Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

KEAMANAN SIBER OTOMOTIF MODERN:

Serangan ECU (Electronic Control Unit) Kendaraan via IoT



YAYASAN PRIMA AGUS TEKNIK

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

KEAMANAN SIBER OTOMOTIF MODERN:

Serangan ECU (Electronic Control Unit) Kendaraan via IoT



YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

KEAMANAN SIBER OTOMOTIF MODERN:
Serangan ECU (Electronic Control Unit) Kendaraan via IoT

Penulis :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

ISBN :

Editor :
Dr. Agus Wibowo, M.Kom, M.Si, MM.

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : info@stekom.ac.id

Hak cipta dilindungi undang-undang
Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara apapun tanpa ijin dari penulis

KATA PENGANTAR

Perkembangan teknologi otomotif modern telah membawa perubahan besar dalam cara kendaraan dirancang, dioperasikan, dan berinteraksi dengan lingkungan sekitarnya. Kendaraan tidak lagi hanya berfungsi sebagai alat transportasi, melainkan telah berkembang menjadi ekosistem digital yang kompleks dan saling terhubung. Integrasi jaringan internal, konektivitas nirkabel, sistem backend, unit kontrol elektronik (ECU), komputer performa tinggi (HPC), sensor pintar, serta kecerdasan buatan (AI) membuka peluang besar bagi inovasi, namun sekaligus memperluas permukaan serangan siber yang harus dikelola secara serius.

Kendaraan modern kini dilengkapi dengan berbagai komponen elektronik dan sistem digital yang saling berkomunikasi. ECU, jaringan internal kendaraan, antarmuka nirkabel, sistem infotainment, layanan berbasis cloud, sensor canggih, serta sistem AI telah menjadi bagian integral dari arsitektur kendaraan. Kompleksitas tersebut menciptakan berbagai titik yang berpotensi menjadi target serangan, baik dari sisi eksternal maupun internal. Oleh karena itu, keamanan siber otomotif tidak dapat lagi dipandang sebagai pelengkap pada tahap akhir pengembangan produk, melainkan harus menjadi bagian mendasar dari perancangan, pengembangan, pengujian, produksi, operasional, dan pemeliharaan kendaraan.

Buku berjudul "*Keamanan Siber Otomotif Modern*" disusun untuk memberikan pemahaman yang menyeluruh mengenai lanskap keamanan siber pada sistem otomotif modern. Buku ini menempatkan keamanan ofensif dan defensif sebagai dua pendekatan yang saling melengkapi. Pendekatan ofensif berfokus pada pemahaman cara kerja serangan, identifikasi kerentanan, pengujian penetrasi, analisis rantai serangan, serta eksplorasi kelemahan pada berbagai lapisan sistem kendaraan. Sementara itu, pendekatan defensif menekankan pentingnya mitigasi risiko, penguatan arsitektur, pengamanan komunikasi, pengelolaan kerentanan, serta pembangunan sistem keamanan yang adaptif dan berkelanjutan.

Pembahasan dalam buku ini diawali dengan tinjauan dasar keamanan ofensif pada sistem otomotif. Pembaca akan diperkenalkan pada perbedaan dan keterkaitan antara pendekatan keamanan ofensif dan defensif, arsitektur E/E kendaraan, komponen utama kendaraan terhubung, ekosistem konektivitas, serta permukaan serangan siber yang perlu dipahami secara komprehensif. Selain itu, buku ini membahas profil ancaman, aspek psikologi dalam keamanan siber, klasifikasi kerentanan, penilaian risiko, metodologi pengujian penetrasi, dan konsep keamanan ofensif otomotif. Pemahaman terhadap elemen-elemen dasar ini menjadi fondasi penting sebelum pembaca melangkah pada pembahasan teknis yang lebih mendalam.

Bagian berikutnya membahas tahapan pengujian penetrasi keamanan otomotif secara sistematis. Pengujian keamanan bukan sekadar aktivitas teknis untuk menemukan celah, tetapi merupakan proses terstruktur yang mencakup perencanaan, penentuan ruang lingkup, identifikasi aset, pengumpulan informasi, penemuan kerentanan, analisis risiko, eksploitasi terkendali, pelaporan, serta tindak lanjut perbaikan. Buku ini juga membahas pentingnya

membangun kumpulan perangkat atau tool arsenal yang sesuai dengan kebutuhan pengujian, mulai dari perangkat untuk sistem backend, antarmuka nirkabel, jaringan internal, ECU berkinerja tinggi, ECU berbasis mikrokontroler, hingga lapisan sensor. Penerapan AI dalam mendukung pengujian keamanan ofensif juga dibahas sebagai bagian dari perkembangan teknologi yang terus berkembang.

Bagian kedua buku ini berfokus pada eksploitasi antarmuka eksternal dan jaringan kendaraan. Salah satu area penting yang dibahas adalah backend kendaraan, yaitu infrastruktur digital yang menghubungkan kendaraan dengan layanan cloud, aplikasi pengguna, sistem manufaktur, pembaruan perangkat lunak, telematika, dan berbagai layanan berbasis internet. Kelemahan pada backend dapat berdampak luas karena tidak hanya menyentuh satu unit kendaraan, tetapi berpotensi memengaruhi banyak kendaraan, pengguna, dan layanan yang saling terhubung. Oleh karena itu, pemahaman terhadap arsitektur backend, protokol konektivitas, kelemahan umum, teknik pengujian penetrasi, studi kasus serangan nyata, serta strategi pengamanan menjadi sangat penting.

Buku ini juga membahas keamanan dan pengujian sistem nirkabel pada kendaraan modern. Teknologi nirkabel seperti Bluetooth, Wi-Fi, jaringan seluler, keyless entry, serta komunikasi jarak jauh memberikan kemudahan bagi pengguna, tetapi sekaligus menjadi salah satu titik masuk yang berpotensi dieksploitasi oleh pihak yang tidak berwenang. Pembahasan mencakup gambaran umum sistem nirkabel, ancaman yang menyertainya, kelemahan yang umum ditemukan, teknik pengujian penetrasi, studi kasus eksploitasi, serta praktik terbaik dalam pengamanan sistem nirkabel. Pemahaman ini menegaskan bahwa keamanan nirkabel bukan hanya persoalan enkripsi, tetapi juga menyangkut desain protokol, manajemen kredensial, autentikasi, segmentasi jaringan, dan pengawasan aktivitas yang mencurigakan.

Aspek lain yang tidak kalah penting adalah keamanan jaringan internal kendaraan. Jaringan internal menjadi tulang punggung komunikasi antar-ECU, sensor, aktuator, sistem infotainment, dan berbagai modul kendali kendaraan. Kelemahan pada lapisan ini dapat berdampak langsung terhadap fungsi kendaraan, termasuk sistem keselamatan, pengendalian mesin, sistem pengereman, kemudi, maupun fitur bantuan pengemudi. Oleh karena itu, buku ini membahas protokol komunikasi pada jaringan internal, arsitektur dan keamanan protokol, kelemahan umum, teknik pengujian penetrasi, studi kasus serangan, serta strategi pengamanan komunikasi internal. Pembahasan ini menegaskan bahwa keamanan kendaraan modern harus dipandang secara menyeluruh, bukan hanya pada permukaan eksternal yang terlihat oleh pengguna.

Bagian ketiga membahas mitos dan realitas keamanan pada arsitektur sistem kendaraan. Salah satu fokus utama adalah keamanan dan eksploitasi ECU berbasis mikrokontroler atau MCU. ECU berperan penting dalam mengendalikan berbagai fungsi kendaraan, mulai dari manajemen mesin, transmisi, keselamatan, hingga sistem hiburan dan konektivitas. Pemahaman terhadap arsitektur, karakteristik, anatomi perangkat keras dan perangkat lunak, kelemahan, rantai serangan, serta strategi penguatan ECU berbasis MCU menjadi sangat penting. Buku ini menekankan bahwa keamanan perangkat keras dan

perangkat lunak harus dikelola secara terpadu, karena celah pada salah satu lapisan dapat menjadi pintu masuk bagi eksploitasi yang lebih luas.

Pembahasan kemudian diperluas ke arsitektur *High-Performance Computer* (HPC) pada kendaraan modern. HPC sering menjadi pusat pemrosesan untuk berbagai fungsi canggih, termasuk sistem infotainment, konektivitas, pemrosesan sensor, fitur bantuan pengemudi, pembaruan perangkat lunak, dan layanan berbasis cloud. Kompleksitas HPC membuatnya menjadi target yang menarik sekaligus kritis dalam konteks keamanan siber otomotif. Buku ini membahas konsep dan arsitektur HPC, anatomi perangkat keras dan perangkat lunak, identifikasi kelemahan, teknik pengujian penetrasi, risiko serangan rantai pasok, studi kasus peretasan dan eksploitasi, serta praktik terbaik dalam penguatan keamanan HPC.

Selain perangkat keras dan jaringan, buku ini juga membahas keamanan sensor dan AI pada kendaraan modern. Sensor berperan penting dalam membantu kendaraan memahami kondisi lingkungan sekitar, sedangkan AI digunakan untuk memproses data, mengenali pola, membuat keputusan, dan mendukung fungsi keselamatan maupun kenyamanan. Namun, sensor dan AI juga memiliki kerentanan tersendiri, termasuk risiko manipulasi data, gangguan sinyal, serangan adversarial, ketidakakuratan model, hingga eksploitasi terhadap antarmuka komunikasi. Oleh karena itu, buku ini membahas arsitektur dan antarmuka sensor, klasifikasi sistem AI, ancaman dan kelemahan, teknik pengujian penetrasi, studi kasus serangan nyata, serta strategi penguatan keamanan sensor dan AI.

Pada bagian akhir, buku ini mengangkat tema transformasi keamanan siber terkelola sebagai arah pengembangan keamanan otomotif masa depan. Pendekatan keamanan yang bersifat ad-hoc sering kali tidak mampu mengikuti kompleksitas sistem kendaraan modern yang terus berkembang. Oleh karena itu, penerapan *Cybersecurity Management System* (CSMS) sepanjang siklus hidup kendaraan menjadi kebutuhan yang semakin mendesak. CSMS membantu organisasi mengelola risiko keamanan secara terstruktur, mulai dari perancangan, pengembangan, produksi, operasional, pemeliharaan, hingga akhir masa pakai kendaraan. Buku ini juga membahas peningkatan standar keamanan, integrasi AI ofensif dan defensif, serta arah masa depan keamanan siber otomotif yang dipengaruhi oleh perkembangan konektivitas, kendaraan otonom, regulasi, dan teknologi kecerdasan buatan.

Buku ini ditujukan bagi mahasiswa, akademisi, peneliti, praktisi keamanan siber, insinyur otomotif, pengembang perangkat lunak, profesional pengujian penetrasi, serta pihak-pihak yang berkepentingan dengan pengembangan kendaraan terhubung dan sistem otomotif modern. Seluruh pembahasan disusun dengan memperhatikan prinsip pengujian yang etis, bertanggung jawab, dan berorientasi pada peningkatan keamanan. Pengetahuan mengenai kerentanan dan teknik serangan harus selalu digunakan untuk memperkuat sistem, melindungi pengguna, mencegah penyalahgunaan, dan mendukung pengembangan teknologi otomotif yang aman serta berkelanjutan.

Penulis menyadari bahwa bidang keamanan siber otomotif merupakan bidang yang sangat dinamis dan terus berkembang seiring kemajuan teknologi. Perubahan pada arsitektur kendaraan, munculnya standar baru, perkembangan regulasi, serta kemajuan AI akan terus membentuk lanskap keamanan otomotif di masa depan. Oleh karena itu, buku ini diharapkan

tidak hanya menjadi sumber referensi, tetapi juga menjadi titik awal bagi pembaca untuk terus belajar, mengeksplorasi, berdiskusi, dan berkontribusi dalam pengembangan ekosistem mobilitas yang lebih aman.

Akhirnya, penulis berharap buku "*Keamanan Siber Otomotif Modern*" dapat memberikan manfaat bagi pembaca, memperkaya wawasan di bidang keamanan siber otomotif, serta menginspirasi pengembangan riset, inovasi, dan praktik terbaik dalam menciptakan kendaraan yang tidak hanya cerdas dan terhubung, tetapi juga aman, andal, dan dapat dipercaya.

Semangat & Selamat Membaca….!!!!!

Semarang, November 2026
Penulis

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

DAFTAR ISI

KATA PENGANTAR ..... **i**
DAFTAR ISI ..... **v**
BAGIAN 1 TINJAUAN DASAR KEAMANAN OFENSIF **1**
BAB 1 DASAR-DASAR KEAMANAN OFENSIF **2**
1.1 Pendekatan Keamanan Ofensif Dan Defensif Pada Sistem Otomotif 2
1.2 Arsitektur E/E Dan Komponen Utama Kendaraan Terhubung 4
1.3 Ekosistem Kendaraan Terhubung Dan Permukaan Serangan Siber 13
1.4 Profil Ancaman Dan Psikologi Dalam Keamanan Siber Otomotif 16
1.5 Klasifikasi Dan Penilaian Kerentanan Pada Sistem Otomotif 20
1.6 Metodologi Dan Pelaksanaan Pengujian Penetrasi Otomotif 25
1.7 Konsep Dan Metodologi Keamanan Ofensif Otomotif 40
BAB 2 TAHAPAN PENGUJIAN PENETRASI **41**
2.1 Perencanaan Pengujian Keamanan Otomotif 41
2.2 Teknik Penemuan Dan Identifikasi Kerentanan Keamanan Otomotif 46
2.3 Rantai Serangan Dan Metode Eksploitasi Pada Kendaraan 64
2.4 Pelaporan Dan Tindak Lanjut Keamanan Otomotif 72
2.5 Tahapan Dan Strategi Pengujian Penetrasi Keamanan Otomotif 76
BAB 3 MEMBANGUN KUMPULAN PERANGKAT (TOOL ARSENAL) **78**
3.1 Perangkat Untuk Sistem *Backend* Kendaraan 78
3.2 Perangkat Antarmuka Nirkabel 86
3.3 Perangkat Jaringan Dalam Kendaraan 95
3.4 Alat ECU Berkinerja Tinggi Dan Berbasis MCU 102
3.5 Perangkat Keamanan Ofensif Pada Lapisan Sensor Kendaraan 108
3.6 Penerapan AI Dalam Pengujian Keamanan Ofensif Kendaraan 112
3.7 Pengujian Keamanan Ofensif Kendaraan Terhubung 113
BAGIAN 2 EKSPLOITASI ANTARMUKA EKSTERNAL DAN JARINGAN KENDARAAN **115**
BAB 4 SERANGAN TERHADAP BACKEND KENDARAAN **116**
4.1 Gambaran Umum Arsitektur *Backend* Kendaraan 116
4.2 Memahami Protokol Konektivitas *Backend* 121
4.3 Kelemahan Umum Dalam Arsitektur *Backend* Kendaraan 124
4.4 Teknik Pengujian Penetrasi 128
4.5 Studi Kasus Serangan Nyata Terhadap *Backend* Kendaraan 133
4.6 Strategi Pengamanan *Backend* Kendaraan 146
4.7 Keamanan *Backend* Kendaraan Terhubung 150
BAB 5 KEAMANAN DAN PENGUJIAN SISTEM NIRKABEL KENDARAAN **152**
5.1 Gambaran Umum Sistem Nirkabel Pada Kendaraan 152
5.2 Antarmuka Nirkabel Dan Ancaman Keamanan Pada Kendaraan Modern 154
5.3 Kelemahan Pada Sistem Nirkabel Kendaraan Modern 160

5.4 Pengujian Penetrasi Pada Sistem Nirkabel Kendaraan 167
5.5 Studi Kasus Eksploitasi Nirkabel Pada Sistem Keamanan Kendaraan Modern .. 174
5.6 Praktik Terbaik Dalam Pengamanan Sistem Nirkabel Kendaraan..... 185
5.7 Pengujian Penetrasi Dan Pengamanan Sistem Nirkabel Kendaraan 190
BAB 6 KEAMANAN DAN PENGUJIAN JARINGAN INTERNAL KENDARAAN 192
6.1 Protokol Komunikasi Pada Jaringan Internal Kendaraan Modern..... 193
6.2 Arsitektur Dan Keamanan Protokol Jaringan Kendaraan 194
6.3 Kelemahan Umum Dalam Komunikasi Di Dalam Kendaraan 199
6.4 Teknik Pengujian Penetrasi Untuk Jaringan Di Dalam Kendaraan..... 216
6.5 Studi Kasus Serangan Siber Pada Jaringan Internal Kendaraan..... 226
6.6 Strategi Pengamanan Komunikasi Jaringan Internal Kendaraan 239
6.7 Keamanan Dan Perlindungan Komunikasi Jaringan Internal Kendaraan 243
BAGIAN 3 MITOS KEAMANAN PADA ARSITEKTUR SISTEM KENDARAAN..... 244
BAB 7 KEAMANAN DAN EKSPLOITASI ECU BERBASIS MIKROKONTROLER (MCU) 245
7.1 Arsitektur Dan Karakteristik ECU Berbasis Mikrokontroler (MCU)..... 246
7.2 Anatomi Perangkat Keras Dan Perangkat Lunak ECU Berbasis MCU 249
7.3 Kelemahan Perangkat Keras Dan Perangkat Lunak Pada ECU Berbasis MCU 265
7.4 Rantai Serangan Dan Tahapan Eksploitasi ECU Berbasis MCU..... 280
7.5 Studi Kasus Kompromi ECU Berbasis MCU..... 286
7.6 Strategi Penguatan Keamanan ECU Berbasis Mikrokontroler..... 295
7.7 Kerentanan Dan Ancaman Keamanan ECU Berbasis Mikrokontroler 299
BAB 8 KEAMANAN DAN PENGUJIAN PENETRASI PADA HPC KENDARAAN 300
8.1 Konsep Dan Arsitektur High-Performance Computer (HPC) 301
8.2 Memahami Anatomi Perangkat Keras Dan Perangkat Lunak HPC 302
8.3 Mengidentifikasi Kelemahan Umum Pada HPC..... 309
8.4 Menjelajahi Teknik Pengujian Penetrasi..... 330
8.5 Serangan Rantai Pasok Dalam Ekosistem Keamanan HPC Kendaraan 336
8.6 Analisis Studi Kasus Peretasan Dan Eksploitasi HPC Kendaraan 337
8.7 Praktik Terbaik Dan Strategi Penguatan Keamanan HPC Kendaraan 355
8.8 Evaluasi Ancaman Dan Strategi Pertahanan Keamanan HPC Kendaraan..... 361
BAB 9 KEAMANAN SENSOR DAN AI PADA KENDARAAN MODERN 364
9.1 Arsitektur Dan Antarmuka Sensor Kendaraan Serta Sistem AI 365
9.2 Klasifikasi Arsitektur Dan Karakteristik Sensor Kendaraan Serta Sistem AI..... 366
9.3 Ancaman Dan Kelemahan Keamanan Pada Sensor Kendaraan Dan Sistem AI .. 373
9.4 Teknik Pengujian Penetrasi Terhadap Sensor Kendaraan Dan AI 384
9.5 Studi Kasus Serangan Nyata Terhadap Sensor Kendaraan Dan Sistem 391
9.6 Prinsip Dan Strategi Penguatan Keamanan Sensor Kendaraan Serta Sistem AI. 404
9.7 Evaluasi Kerentanan Dan Strategi Pertahanan Sensor Serta AI Otomotif..... 408
BAB 10 TRANSFORMASI KEAMANAN SIBER TERKELOLA 410
10.1 Kelemahan Dan Risiko Pendekatan Ad-Hoc Dalam Keamanan Siber Otomotif . 411
10.2 Penerapan CSMS Sepanjang Siklus Hidup Kendaraan..... 414

10.3 Meningkatkan Standar Keamanan Dalam Sistem Otomotif 437
10.4 Integrasi AI Ofensif Dan Defensif Dalam Keamanan Siber Otomotif 440
10.5 Arah Masa Depan Keamanan Siber Otomotif 452
DAFTAR PUSTAKA 454

BAGIAN 1
TINJAUAN DASAR KEAMANAN OFENSIF

Bagian ini membangun landasan konseptual dan praktis yang diperlukan untuk melakukan serangan terhadap kendaraan modern yang terhubung (*connected vehicles*). Anda akan memperoleh pemahaman yang jelas mengenai arsitektur kelistrikan/elektronik otomotif, profil penyerang, serta kerangka kerja klasifikasi kerentanan yang menjadi acuan dalam pemodelan ancaman di dunia nyata.

Bagian ini kemudian menguraikan tahapan terstruktur dalam kegiatan pengujian penetrasi (*penetration testing*) profesional, dilanjutkan dengan tinjauan menyeluruh terhadap ekosistem perangkat ofensif yang mencakup setiap lapisan arsitektur kendaraan. Di akhir bagian ini, Anda akan memiliki kosakata, metodologi, dan pemahaman mengenai perangkat yang diperlukan untuk menangani kegiatan keamanan ofensif otomotif dengan penuh percaya diri.

Bagian ini terdiri dari bab-bab berikut:

- Bab 1, Dasar-Dasar Keamanan Ofensif
- Bab 2, Tahapan Pengujian Penetrasi
- Bab 3, Membangun Perangkat (*Tooling*) untuk Keamanan Ofensif

BAB 1
DASAR-DASAR KEAMANAN OFENSIF

Membangun pertahanan keamanan pada sistem otomotif tanpa pemahaman mengenai bagaimana sistem tersebut dapat diserang akan menempatkan para insinyur keamanan pada posisi yang sangat tidak menguntungkan dibandingkan dengan pihak lawan. Pihak lawan ini menghabiskan sebagian besar hidup mereka untuk mempelajari pertahanan tersebut dan mencari cara untuk melumpuhkannya. Dalam permainan "kucing-kucingan" antara insinyur keamanan siber dan peretas, satu pihak menghabiskan waktu yang tak terhitung jumlahnya untuk merancang pertahanan, sementara pihak lawan mungkin hanya membutuhkan waktu beberapa jam untuk menembusnya.

Asimetri antara pertahanan dan serangan siber berakar pada kenyataan mendasar bahwa membangun sesuatu jauh lebih mahal dan membutuhkan lebih banyak upaya dibandingkan menghancurkannya. Namun, asimetri yang jelas menguntungkan pihak yang melakukan pengujian dan peretasan ini bukan berarti mereka yang bekerja di bidang pertahanan siber harus pasrah menerima kenyataan bahwa sistem mereka pasti akan ditembus, atau menyerah pada kekalahan. Keamanan ofensif (*offensive security*) merupakan perpaduan antara sains dan seni dalam menemukan kerentanan pada sistem komputer serta merancang eksploitasi yang memungkinkan penyerang mencapai tujuan mereka.

Sebelum mendalami bidang keamanan ofensif, bab ini akan diawali dengan meninjau beberapa konsep dasar dalam keamanan siber otomotif, yang akan menjadi landasan untuk memahami bagaimana sistem tersebut dapat disusupi. Pertama, kita akan membahas alasan mengapa keamanan siber ofensif itu penting, kemudian meninjau konsep arsitektur Elektrik/Elektronik (E/E) pada kendaraan modern dalam konteks lingkungan yang terhubung sepenuhnya. Selanjutnya, kita akan mengeksplorasi siapa saja pihak penyerang dan apa motivasi mereka. Setelah itu, kita akan mempelajari konsep-konsep fundamental yang akan membantu dalam bab-bab selanjutnya saat kita menelusuri berbagai kerentanan serta metode serangan yang digunakan untuk mengeksploitasinya guna menembus sistem otomotif.
Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

- Motivasi di balik keamanan ofensif
- Anatomi arsitektur E/E kendaraan
- Memahami lingkungan operasional kendaraan yang terhubung (*connected vehicles*)
- Profil penyerang
- Kelemahan, kerentanan, dan ancaman
- Dasar-dasar rantai serangan (*attack chains*)
- Dasar-dasar pengujian penetrasi (*penetration testing*)

1.1 PENDEKATAN KEAMANAN OFENSIF DAN DEFENSIF PADA SISTEM OTOMOTIF

Para insinyur yang bekerja di bidang keamanan siber biasanya menghabiskan sebagian besar karier mereka untuk menganalisis ancaman dan merancang kendali keamanan siber

guna memitigasi ancaman tersebut. Mereka dengan bangga menganggap diri mereka sebagai insinyur keamanan defensif, yang berfokus pada bagaimana teknologi keamanan terkini dapat menggagalkan berbagai serangan dan memperkuat kendali keamanan. Di industri otomotif, para insinyur keamanan defensif bekerja tanpa kenal lelah untuk memastikan bahwa sistem otomotif yang krusial bagi keselamatan tetap dapat beroperasi dengan aman, bahkan saat berada di bawah serangan.

Beberapa kendaraan mungkin memiliki kemampuan mengemudi otonom terbatas, sementara yang lain mungkin sepenuhnya otonom tanpa kemungkinan intervensi pengemudi jika terjadi pelanggaran keamanan yang berdampak pada keselamatan. Meskipun fokus utama mereka adalah melindungi keselamatan pengguna jalan, tujuan keamanan lainnya mencakup perlindungan kekayaan intelektual serta reputasi perusahaan, yang dapat mengalami kerugian finansial yang parah akibat pelanggaran keamanan. Tujuan lainnya adalah memastikan ketersediaan fungsi kendaraan guna mencegah hilangnya kepercayaan konsumen dan menghindari penarikan kembali produk (*recall*) yang memakan biaya besar.

Jika hanya berfokus pada pertahanan, para profesional keamanan ini dapat dengan mudah memiliki titik buta (*blind spot*) dalam memahami bagaimana sistem mereka mungkin diserang. Sebaliknya, keamanan ofensif menerapkan pola pikir yang berbeda; alih-alih mengasumsikan bahwa suatu kendali keamanan sudah efektif, pendekatan ini justru mengeksplorasi bagaimana kendali tersebut dapat dilumpuhkan untuk mencapai tujuan penyerang. Jika keamanan defensif berupaya membangun penghalang yang kuat, maka keamanan ofensif berupaya mengidentifikasi kelemahan dan celah pada penghalang tersebut. Satu pihak berfokus pada penyempurnaan teknologi keamanan, sementara pihak lainnya berfokus pada penemuan dan eksploitasi terhadap ketidaksempurnaan yang ada.

Mereka yang bekerja di bidang keamanan ofensif mendedikasikan diri untuk memahami bagaimana sistem dibangun, dengan tujuan menemukan kelemahan dan merancang eksploitasi. Oleh karena itu, sudah sewajarnya jika insinyur keamanan defensif juga meluangkan waktu untuk memahami metode, teknik, dan perangkat serangan yang digunakan oleh para profesional keamanan ofensif untuk menembus pertahanan. Tanpa pengetahuan ini, insinyur keamanan siber defensif terutama mereka yang menangani sistem yang krusial bagi keselamatan, seperti yang terdapat pada kendaraan modern akan berada dalam posisi yang sangat dirugikan.

Itulah sebabnya buku ini hadir untuk memberikan wawasan mengenai ranah keamanan ofensif kepada para insinyur keamanan defensif, agar mereka dapat merancang langkah-langkah keamanan yang tepat dan memastikan langkah-langkah tersebut tetap tangguh dalam menghadapi serangan siber. Motivasi lain di balik penulisan buku ini adalah untuk menepis anggapan keliru bahwa kendaraan modern sangat sulit diserang dan bahwa keberhasilan peretasan hanyalah sebatas teori. Meskipun telah banyak upaya dilakukan untuk membantah gagasan tersebut, banyak insinyur otomotif masih meyakini bahwa serangan yang berujung pada dampak berbahaya bagi kendaraan sangat kecil kemungkinannya terjadi, mengingat kompleksitas dan sifat sistem tersebut yang bersifat tertutup (eksklusif/*proprietary*).

Di sepanjang buku ini, kita akan mengkaji serangan nyata yang menyasar setiap lapisan arsitektur kendaraan. Kita akan melihat bagaimana para penyerang berhasil menembus berbagai lapisan keamanan untuk mencapai tujuan mereka. Bagi para insinyur yang berpendapat bahwa sistem mereka sudah cukup aman karena telah dilengkapi dengan langkah-langkah keamanan dasar, kita akan melihat bahwa hasil yang mereka peroleh sering kali tidak lebih baik daripada mereka yang sama sekali mengabaikan aspek keamanan. Namun, sebelum melangkah lebih jauh, mari kita pahami terlebih dahulu dasar-dasar mengenai arsitektur E/E (elektrik/elektronik).

1.2 ARSITEKTUR E/E DAN KOMPONEN UTAMA KENDARAAN TERHUBUNG

Saat merencanakan kampanye keamanan ofensif terhadap suatu sistem, langkah pertama adalah memahami susunan sistem tersebut, komponen-komponennya, serta cara kerjanya. Sistem otomotif pun demikian; alih-alih hanya berfokus pada satu subsistem kendaraan, penting untuk mendapatkan gambaran menyeluruh mengenai semua komponen yang membentuk kendaraan terhubung (*connected vehicle*) beserta lingkungannya. Pada bagian ini, kita akan memperkenalkan lapisan-lapisan utama dalam arsitektur E/E, sementara perincian mengenai setiap lapisan akan dibahas pada bab-bab selanjutnya, saat kita mengulas serangan yang ditargetkan pada lapisan-lapisan tersebut.

Sebelum membahas berbagai titik masuk ke dalam arsitektur elektronik/elektrikal (E/E) kendaraan, penting untuk memahami dua jenis arsitektur E/E kendaraan yang paling umum: arsitektur berbasis domain dan arsitektur berbasis komputer pusat. Jenis pertama merupakan arsitektur warisan (*legacy*) yang telah berkembang selama beberapa dekade terakhir, yang ditandai dengan pemisahan fisik unit kontrol elektronik (ECU) ke dalam domain fungsional yang saling terhubung melalui gerbang pusat (*central gateway*).

Jenis kedua merupakan arsitektur mutakhir yang dianggap sebagai pendukung utama kendaraan berbasis perangkat lunak (*software-defined vehicle*/SDV); dalam arsitektur ini, satu atau lebih unit komputasi pusat menjalankan berbagai aplikasi perangkat lunak, sementara ECU zona dan ECU konvensional menangani sebagian besar tugas penginderaan dan aktuasi. Meskipun arsitektur berbasis komputer pusat kini semakin banyak diadopsi, kedua jenis arsitektur ini kemungkinan besar akan tetap digunakan dalam waktu yang lama. Arsitektur berbasis domain yang terdistribusi akan tetap menjadi pilihan bagi banyak platform lama, di mana kebutuhan bisnis belum menuntut transisi ke sistem berbasis komputer pusat. Masing-masing arsitektur ini menghadirkan tantangan keamanan tersendiri, sehingga memahami kelemahan masing-masing menjadi prasyarat untuk menyusun rantai serangan (*attack chain*) yang efektif.

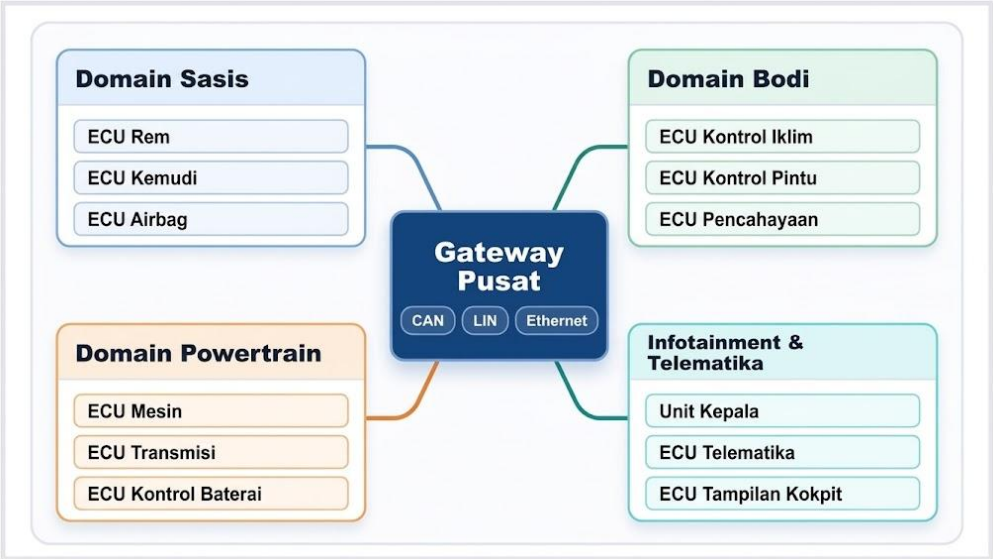
Jenis-Jenis Arsitektur Kendaraan

Arsitektur E/E kendaraan modern kini semakin banyak dikembangkan berdasarkan dua varian utama: arsitektur berbasis multi-domain yang dihubungkan oleh gerbang pusat, serta arsitektur yang lebih terpusat berbasis zona atau server dengan komputer pusat. Dalam praktiknya, banyak kendaraan menggunakan bentuk hibrida dari pendekatan-pendekatan

tersebut. Pada bagian ini, kita akan menelaah masing-masing arsitektur untuk memahami bagaimana sistem tersebut dapat ditembus.

Arsitektur Berbasis Domain

Dalam arsitektur ini, fungsi-fungsi kendaraan dipisahkan ke dalam domain-domain tertentu, di mana setiap domain berbagi perangkat sensor dan aktuator yang sama. Biasanya, terdapat domain sasis yang mengelola sistem pengereman, kemudi, dan kantung udara (*airbag*); *domain powertrain* (sistem penggerak) yang mencakup kontrol mesin dan transmisi; serta domain bodi yang bertanggung jawab atas sistem pemanas, pendingin, pengaturan kursi, lampu, dan akses pintu. Selain itu, terdapat domain infotainment dan telematika yang menyediakan pengalaman kokpit digital serta konektivitas eksternal. Domain-domain ini saling terhubung melalui gerbang pusat yang mendukung berbagai protokol komunikasi, seperti Ethernet, *Controller Area Network* (CAN), dan *Local Interconnect Network* (LIN). Arsitektur E/E berbasis domain, yang diilustrasikan pada Gambar 1.1, hadir dalam berbagai varian berdasarkan kebutuhan OEM untuk menambahkan fungsi-fungsi baru ke dalam berbagai model kendaraan.

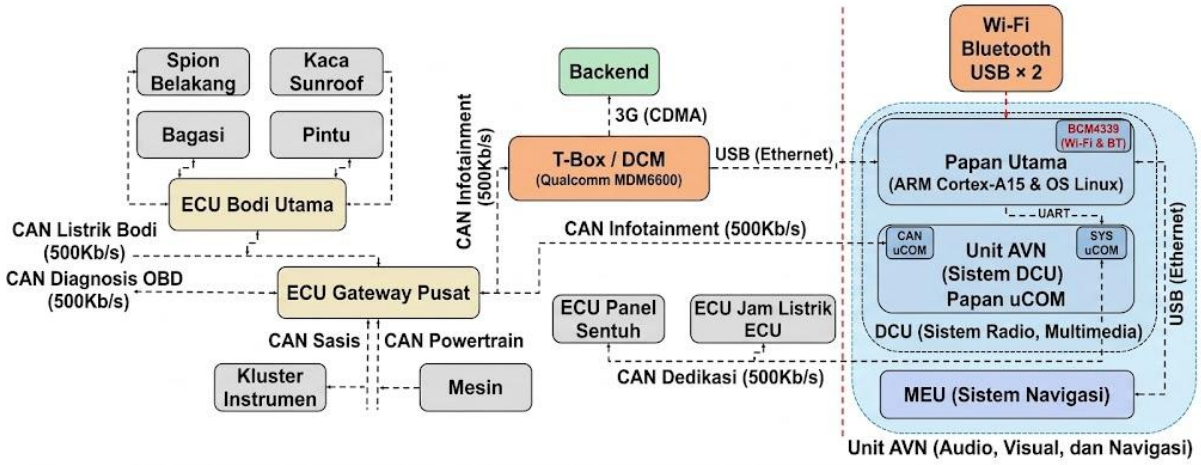


Gambar 1.1: Arsitektur E/E berbasis multi-domain yang terhubung melalui *gateway*

Dalam praktiknya, sering ditemukan komponen sub-bus di mana satu ECU berfungsi sebagai *gateway* lokal, yang mengintegrasikan ECU tambahan ke dalam jaringan kendaraan. Idealnya, ECU disegmentasikan secara rapi berdasarkan fungsi kendaraan, dengan *gateway* yang menyediakan isolasi fisik maupun logis yang kuat. Akan tetapi, sering kali terdapat ECU dengan tingkat kepentingan keselamatan (*safety*) dan keamanan (*security*) yang berbeda-beda dalam satu jaringan yang sama seperti pengontrol kantung udara (*airbag*) yang berbagi jaringan dengan ECU sistem hiburan (*infotainment*), atau pengontrol mesin yang berada dalam satu jaringan dengan unit telematika.

Untuk memahami contoh nyata arsitektur E/E berbasis domain, Anda dapat melihat Gambar 1.2, yang menampilkan topologi pada kendaraan Lexus NX300 tahun 2017. Perhatikan

bagaimana *gateway* pusat memisahkan berbagai domain, seperti domain konektivitas, kontrol bodi, sasis, dan sistem penggerak (*powertrain*).



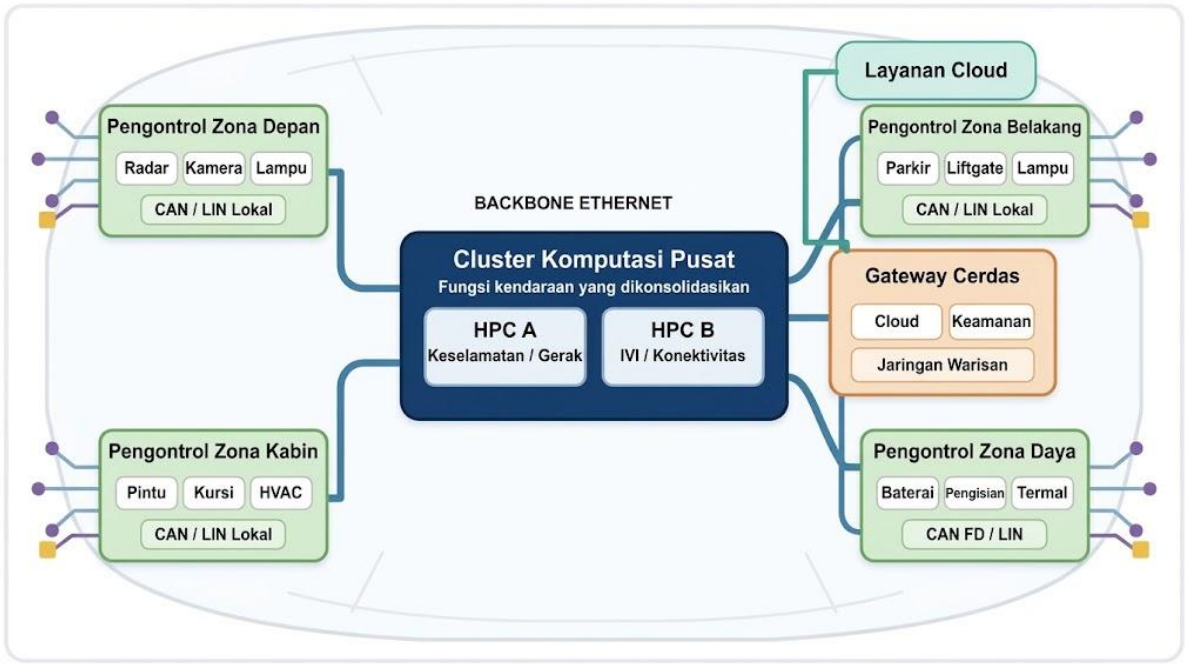
Gambar 1.2: Arsitektur E/E berbasis domain yang terhubung melalui *gateway* pada kendaraan Lexus NX300 tahun 2017

Pada domain kendaraan dengan segmentasi yang buruk, penyerang mungkin hanya perlu membobol satu ECU yang dapat diakses dari jarak jauh untuk berinteraksi secara langsung dengan ECU yang krusial bagi keselamatan, seperti modul kantung udara (*airbag*). Dalam kasus di mana arsitektur kelistrikan/elektronik (E/E) tersegmentasi dengan baik, fokus penyerang akan beralih ke ECU gerbang (*gateway*). Membobol *gateway* menjadi langkah krusial untuk berpindah dari titik akses eksternal seperti unit infotainment menuju lapisan yang lebih dalam dalam arsitektur E/E, sebagaimana akan kita bahas pada bab-bab selanjutnya.

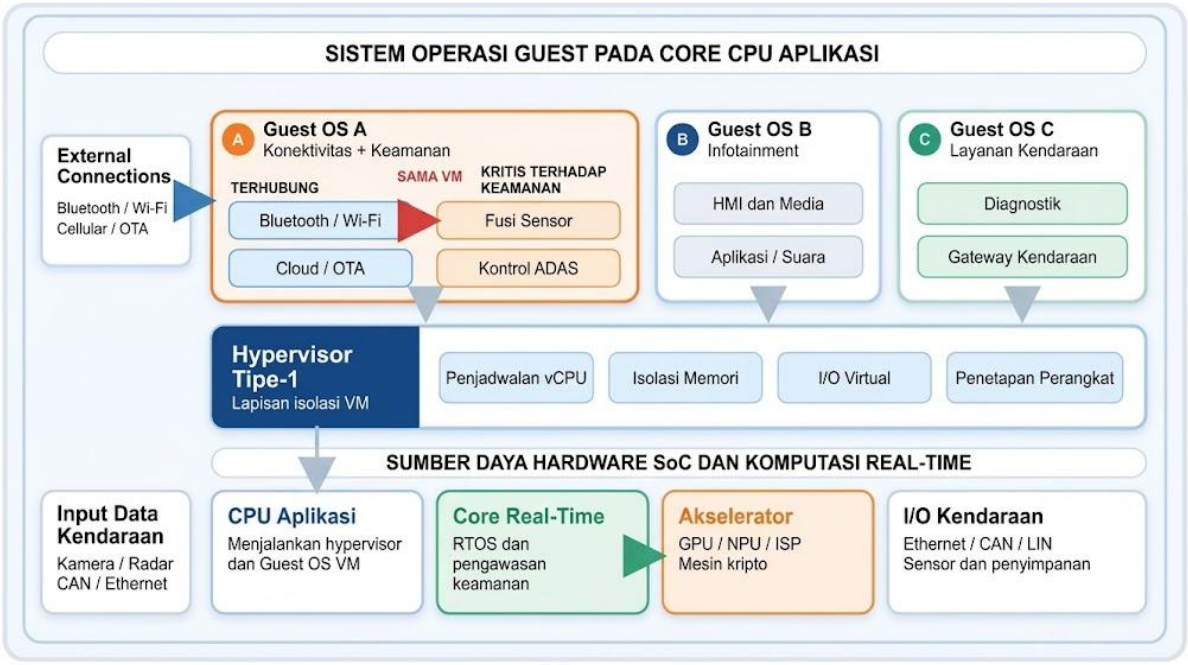
Arsitektur Terpusat Berbasis Zona

Arsitektur kendaraan berbasis komputer pusat merupakan ciri khas dari apa yang kini dikenal luas sebagai kendaraan berbasis perangkat lunak (*software-defined vehicle*). Istilah ini merujuk pada tren migrasi fungsi-fungsi kendaraan dari ECU yang dirancang untuk tujuan khusus ke satu atau beberapa unit komputasi berkinerja tinggi, yang menjalankan fungsi tersebut melalui perangkat lunak yang saling terhubung dalam jaringan. Arsitektur semacam ini memiliki beragam bentuk, namun kita akan berfokus pada model yang kini semakin populer, yaitu arsitektur yang terdiri dari komputer pusat yang terhubung dengan beberapa pengontrol zona (*zone controller*), sebagaimana ditunjukkan pada Gambar 1.3.

Dalam arsitektur ini, platform komputer pusat dapat terdiri dari satu atau lebih *System on Chip* (SoC) yang menampung beberapa sistem operasi tamu (*guest operating system*) dalam lingkungan tervirtualisasi. Hal ini dimungkinkan oleh *hypervisor* yang memvirtualisasikan berbagai komponen perangkat keras sistem serta mengelola sejumlah mesin virtual (*virtual machine*). Biasanya, setiap mesin virtual (VM) menjalankan sistem operasi lengkap pada unit pemrosesan pusat (*central processing unit* atau CPU) aplikasi berkinerja tinggi, sementara inti (*core*) tertanam khusus menjalankan sistem operasi waktu nyata (*real-time operating system*) yang mengelola serangkaian akselerator perangkat keras untuk tujuan tertentu.



Gambar 1.3: Arsitektur E/E dengan komputer pusat



Gambar 1.4: Pemartisian perangkat lunak pada komputer pusat

Akibat adanya konsolidasi fungsi yang sangat padat ke dalam satu platform komputer pusat, produsen peralatan asli (*Original Equipment Manufacturer* atau OEM) sering kali mengintegrasikan fungsi dengan konektivitas tinggi seperti stack Bluetooth/Wi-Fi bersama fungsi yang krusial bagi keselamatan (*safety-critical*), seperti fusi sensor, di dalam satu sistem operasi tamu yang sama. Kelemahan dari pendekatan ini adalah kerentanan pada salah satu

aplikasi yang terhubung ke dunia luar memiliki kemungkinan besar untuk berdampak pada aplikasi yang krusial bagi keselamatan di dalam sistem operasi tamu yang sama.

Arsitektur jenis ini, yang digambarkan dalam Gambar 1.4, sangat berbeda dengan arsitektur berbasis domain, di mana aplikasi yang krusial bagi keselamatan (*safety-critical*) dipisahkan ke dalam unit kontrol elektronik (ECU) yang berdiri sendiri. Meskipun ECU tersebut mungkin memiliki keamanan yang tidak sempurna, eksploitasi jarak jauh tetap sulit dilakukan karena adanya berbagai penghalang fisik. Dalam arsitektur komputer pusat, tantangan isolasi beralih dari pemisahan domain secara fisik menjadi pemisahan perangkat lunak dan perangkat keras di dalam satu platform komputer. Dapat dipahami mengapa jaminan isolasi keamanan yang disediakan oleh perangkat keras harus benar-benar terpenuhi untuk meniru fungsi domain yang terpisah secara fisik.

Idealnya, fungsi-fungsi domain diisolasi dalam mesin virtual (VM) khusus, yang dipisahkan secara spasial dan temporal oleh *hypervisor* Tipe 1. *Hypervisor* memastikan setiap VM mendapatkan alokasi waktu (*time slice*) tersendiri, serta menjamin bahwa memori dan sumber daya perangkat keras suatu VM tidak dapat diakses oleh VM lainnya. Jika diperlukan pembagian sumber daya, *hypervisor* biasanya akan menyediakan layanan khusus yang berfungsi sebagai perantara menuju sumber daya bersama tersebut. Dalam situasi di mana beban pengelolaan VM terlalu besar, sering kali digunakan satu VM tamu (*Guest VM*) yang menampung beberapa kontainer, atau kombinasi antara VM dan container di mana sebagian VM menjalankan beberapa aplikasi, sementara VM lainnya menjalankan beberapa kontainer. Meskipun arsitektur ini menghadirkan tantangan yang lebih besar bagi para insinyur keamanan ofensif, fokus penyerang akan beralih untuk mencari aplikasi yang rentan, yang dapat dijadikan titik awal untuk menyusupi layanan krusial lainnya dalam sistem seperti yang akan kita bahas pada bab-bab selanjutnya.

Baik saat mengevaluasi keamanan arsitektur berbasis domain maupun arsitektur berbasis komputer pusat, seseorang harus terlebih dahulu memahami antarmuka kendaraan yang terbuka ke luar. Antarmuka ini berfungsi sebagai pintu masuk menuju lapisan-lapisan yang lebih dalam dari arsitektur kelistrikan/elektronik (E/E), yang akan menjadi topik pembahasan pada bagian berikutnya.

Antarmuka Eksternal Kendaraan

Pada bagian tepi arsitektur kelistrikan/elektronik (E/E) kendaraan terdapat antarmuka eksternal yang terbuka, seperti Wi-Fi, Bluetooth, jaringan seluler (GSM/4G/5G), antena *Global Positioning System* (GPS), V2X, serta titik akses masuk nirkabel berbasis *Near Field Communication* (NFC) atau *Radio Frequency* (RF). Selain antarmuka nirkabel, terdapat pula berbagai antarmuka fisik, seperti port *Universal Serial Bus* (USB), port *On-board Diagnostics* (OBD), port pengisian daya *Electric Vehicle Supply Equipment* (EVSE), dan antarmuka lain yang dapat dihubungkan untuk mengaitkan kendaraan dengan lingkungannya.

Bergantung pada tujuan penyerang, melakukan enumerasi terhadap antarmuka eksternal baik nirkabel maupun fisik merupakan langkah awal untuk menentukan antarmuka mana yang paling mungkin memberikan celah keberhasilan dalam melakukan peretasan. Sebagai contoh, jika targetnya adalah alur pembaruan perangkat lunak, maka antarmuka Wi-

Fi dan seluler akan menjadi sasaran utama; sementara itu, jika tujuannya adalah memanipulasi sistem akses masuk kendaraan, maka antarmuka NFC dan RF menjadi lebih menarik untuk disasar. Setiap antarmuka menghadirkan serangkaian protokol komunikasi yang harus dipahami sebelum menyusun strategi serangan.

Dengan mengeksplorasi antarmuka eksternal kendaraan, penyerang berupaya memahami perimeter keamanan kendaraan serta mengidentifikasi titik masuk yang kemungkinan besar memiliki kerentanan untuk dieksploitasi. Saat kita menelusuri lebih dalam ke dalam arsitektur E/E, komponen pertama yang akan ditemui adalah ECU (*Electronic Control Unit*), yang akan dibahas pada bagian berikutnya.

***Electronic Control Unit* (ECU)**

Sebagai elemen dasar dalam sistem otomotif, ECU biasanya menjadi target utama dalam setiap operasi keamanan ofensif yang berhasil, karena komponen inilah yang menjalankan kendali langsung atas fungsi-fungsi kendaraan. Baik itu membuka kunci pintu mobil, menyalakan mesin, mengisi daya baterai kendaraan, maupun mengaktifkan rem, ECU merupakan komponen dalam jaringan yang berfungsi untuk mendeteksi, memproses, dan melakukan tindakan (aktuasi).

Namun, tidak semua ECU memiliki tingkat risiko keamanan atau nilai eksploitasi yang sama. Sebagai contoh, ECU telematika memiliki tingkat risiko yang tinggi, tetapi komponen ini saja tidak dapat mengubah pergerakan kendaraan. Sebaliknya, meskipun modul kantung udara (*airbag*) tidak dapat diakses dari luar, komponen ini dapat menyebabkan kerusakan serius pada penumpang jika kantung udara mengembang di luar kondisi operasional normal. Untuk memahami perbedaan antara berbagai ECU ini dengan lebih baik, kita dapat mengelompokkannya ke dalam tiga kategori:

- **Mikrokontroler berkinerja rendah**: Didukung oleh satu mikrokontroler atau *Application-Specific Integrated Circuit* (ASIC), ECU ini mengendalikan satu fungsi utama kendaraan dan terhubung dengan sensor dan/atau aktuator melalui LIN, CAN, atau keduanya. Perangkat lunaknya biasanya berjalan pada satu inti (*single-core*) dengan memori *flash* dan *Random Access Memory* (RAM) tertanam. Sistem ini umumnya menjalankan perangkat lunak khusus (*proprietary*) yang terintegrasi erat dengan perangkat keras demi alasan kinerja. Beberapa sensor cerdas dapat termasuk dalam kategori ini jika memuat perangkat lunak khusus yang dijalankan dalam lingkungan ASIC dan berkomunikasi melalui protokol dengan bandwidth rendah, seperti LIN.
- **Mikrokontroler berkinerja menengah**: Kategori ini mengimplementasikan satu atau lebih fungsi kendaraan tingkat rendah ke dalam satu ECU dengan memanfaatkan mikrokontroler multicore yang memiliki kapasitas *flash* dan RAM lebih besar. Selain berkomunikasi melalui LIN dan CAN, ECU ini biasanya mendukung Ethernet untuk memungkinkan komunikasi dengan ECU berkinerja tinggi, bahkan mungkin dengan lingkungan eksternal kendaraan. Perangkat lunak untuk ECU ini dijalankan dalam lingkungan sistem operasi waktu nyata (*real-time operating system*), yang biasanya diimplementasikan menggunakan platform perangkat lunak terstandarisasi seperti AUTOSAR (*AUTomotive Open System Architecture*) Classic Platform.

- **Pengendali berkinerja tinggi (berbasis SoC)**: Ini adalah jenis ECU yang paling canggih karena dijalankan pada satu atau lebih *System-on-Chip* (SoC). Sistem ini beroperasi menggunakan kombinasi inti aplikasi dan inti waktu nyata (*real-time*), melakukan booting dari memori *flash* eksternal, serta sangat bergantung pada *Dynamic Random-Access Memory* (DRAM) eksternal. *High-Performance Computer* (HPC) biasanya menjalankan kombinasi sistem operasi yang kompleks seperti QNX atau Linux bersamaan dengan satu atau lebih instansi sistem operasi waktu nyata (RTOS). Sistem ini memiliki banyak karakteristik yang serupa dengan sistem tingkat perusahaan (*enterprise*), sehingga menjadi target yang menarik bagi penyerang. Berbeda dengan dua jenis sebelumnya, sistem ini memiliki *Software Bill of Materials* (SBOM) yang kompleks, yang mencakup banyak komponen sumber terbuka (*open source*) maupun komponen siap pakai (*off-the-shelf*).

ECU bergantung pada sejumlah besar sensor untuk menjalankan fungsinya. Namun, sensor menghadirkan serangkaian ancaman keamanannya sendiri. Pada bagian selanjutnya, kita akan meninjau secara singkat jenis sensor yang lazim ditemukan pada kendaraan modern yang terhubung.

Sensor

Sensor pada kendaraan yang terhubung (*connected vehicle*) tersebar luas dan memiliki beragam fungsi, sehingga menciptakan area serangan (*attack surface*) yang besar. Sensor pada bodi kendaraan mencakup kamera, sensor ultrasonik, LiDAR (*Light Detection and Ranging*), Radar (*Radio Detection and Ranging*), serta penerima sinyal RF. Di dalam kabin, sensor berfungsi untuk berbagai keperluan, seperti memantau keberadaan pengemudi, mengatur suhu kabin, mendeteksi posisi kursi, serta fungsi lain yang menunjang keselamatan dan kenyamanan penumpang.

Sensor juga tertanam jauh di dalam ECU (*Electronic Control Unit*) dan berbagai komponen elektromekanis lainnya. Sebagai contoh, sensor di ruang mesin berperan dalam pengendalian emisi, sementara sensor dalam sistem pengereman membantu pengaturan cairan rem. Penyerang dapat menargetkan sensor-sensor ini, bergantung pada bagaimana sensor tersebut mendukung tujuan serangan mereka dan seberapa mudah sensor tersebut untuk diganggu.

Tiga faktor menentukan tingkat kerentanan keamanan suatu sensor: lokasinya di dalam kendaraan, karakteristik fisiknya, dan metode antarmukanya. Sensor yang dipasang di bagian luar kendaraan tentu paling rentan terhadap gangguan langsung. Sensor dengan karakteristik fisik yang membuatnya mudah diganggu juga menjadi target menarik, biasanya melalui serangan jarak dekat (*proximity attacks*). Selain itu, sensor yang dapat diakses melalui jaringan juga menjadi target potensial, karena penyerang hanya perlu berada dalam jaringan lokal kendaraan untuk memengaruhi sensor tersebut. Pada bab-bab selanjutnya, kita akan melihat bagaimana penyerang dapat menyalahgunakan sensor dalam serangan nyata untuk mencapai tujuan mereka.

Protokol Jaringan

Komponen-komponen kendaraan saling terhubung melalui berbagai protokol komunikasi, yang masing-masing dirancang untuk mendukung jenis pertukaran data dan kebutuhan kinerja tertentu. Seiring dengan semakin kompleksnya kendaraan modern dengan banyaknya subsistem yang berinteraksi secara waktu nyata (*real-time*) pemilihan dan penerapan protokol-protokol ini menjadi sangat krusial untuk memastikan pengoperasian kendaraan yang aman, efisien, dan terlindungi. Pada bagian ini, kita akan membahas tiga protokol yang paling umum digunakan dalam jaringan otomotif: LIN, CAN, dan Ethernet, yang masing-masing menghadirkan tantangan keamanan serta manfaat operasional yang unik.

Local Interconnect Network (LIN)

LIN adalah protokol yang hemat biaya, yang terutama digunakan untuk mengendalikan sensor dan aktuator sederhana dalam sistem dengan kebutuhan kinerja rendah. LIN dapat ditemukan di area kendaraan yang tidak kritis, seperti untuk mengendalikan mekanisme jendela dan pengaturan kursi, namun juga di area kritis, misalnya untuk berkomunikasi dengan sensor. Dengan lebar pita (*bandwidth*) yang rendah (hingga 20 kbps) dan penggunaan komunikasi satu kabel, LIN sangat ideal untuk fungsi-fungsi yang tidak memerlukan lebar pita tinggi.

Jaringan LIN juga lebih rentan terhadap gangguan dan memiliki mekanisme pemeriksaan kesalahan bawaan yang lebih sedikit, sehingga kurang tangguh dibandingkan protokol lain seperti CAN. Meskipun memiliki keterbatasan tersebut, LIN tetap memegang peranan penting dalam elektronik otomotif karena kesederhanaan dan biaya rendahnya, terutama untuk aplikasi yang dapat menoleransi batasan kinerjanya. Terkait aspek keamanan, mengamankan setiap bingkai (*frame*) data LIN melalui metode kriptografi bisa menjadi tantangan tersendiri, mengingat kapasitas muatan (*payload*) pesan LIN yang sangat kecil.

Controller Area Network (CAN)

Protokol CAN merupakan salah satu standar komunikasi yang banyak digunakan dalam industri otomotif, yang dikenal karena keandalan, ketangguhan, dan kinerjanya secara waktu nyata (*real-time*). CAN awalnya dikembangkan agar unit kendali elektronik (ECU) di dalam kendaraan dapat berkomunikasi tanpa memerlukan rangkaian kabel yang rumit, sebuah aspek yang sangat krusial mengingat kendaraan modern dapat memiliki 50 hingga 100 ECU untuk mengendalikan berbagai sistem. CAN dirancang untuk mendukung ECU dengan performa menengah, seperti yang digunakan untuk manajemen mesin, kontrol transmisi, sistem pengereman, dan fungsi waktu nyata (*real-time*) lainnya. Protokol ini menyertakan mekanisme deteksi kesalahan dan isolasi gangguan bawaan, yang membuatnya sangat andal, bahkan di lingkungan otomotif yang keras di mana interferensi elektromagnetik atau degradasi sinyal dapat terjadi.

Namun, seiring dengan meningkatnya kompleksitas sistem kendaraan, keterbatasan CAN 2.0 dan CAN FD terutama dalam hal bandwidth menjadi semakin nyata. Protokol-protokol ini kesulitan memenuhi kebutuhan *throughput* data yang tinggi untuk aplikasi-aplikasi baru, seperti sistem bantuan pengemudi tingkat lanjut (ADAS) dan sistem hiburan dalam kendaraan (*infotainment*). Meskipun demikian, rekam jejak yang terbukti dan adopsi luas di industri

memastikan bahwa CAN akan tetap menjadi bagian mendasar dari jaringan kendaraan di masa mendatang, sering kali beroperasi berdampingan dengan protokol yang lebih baru seperti Ethernet.

Terkait keamanan, CAN memiliki kelemahan karena memang tidak dirancang dengan fitur keamanan sejak awal. Upaya untuk mengamankannya biasanya berdampak besar pada performa ECU otomotif yang sumber dayanya memang terbatas. Walaupun begitu, terdapat beberapa langkah keamanan untuk mengatasi kelemahan protokol CAN; hal ini akan dibahas pada bab-bab selanjutnya saat kita mengevaluasi berbagai serangan terhadap jaringan di dalam kendaraan.

Ethernet

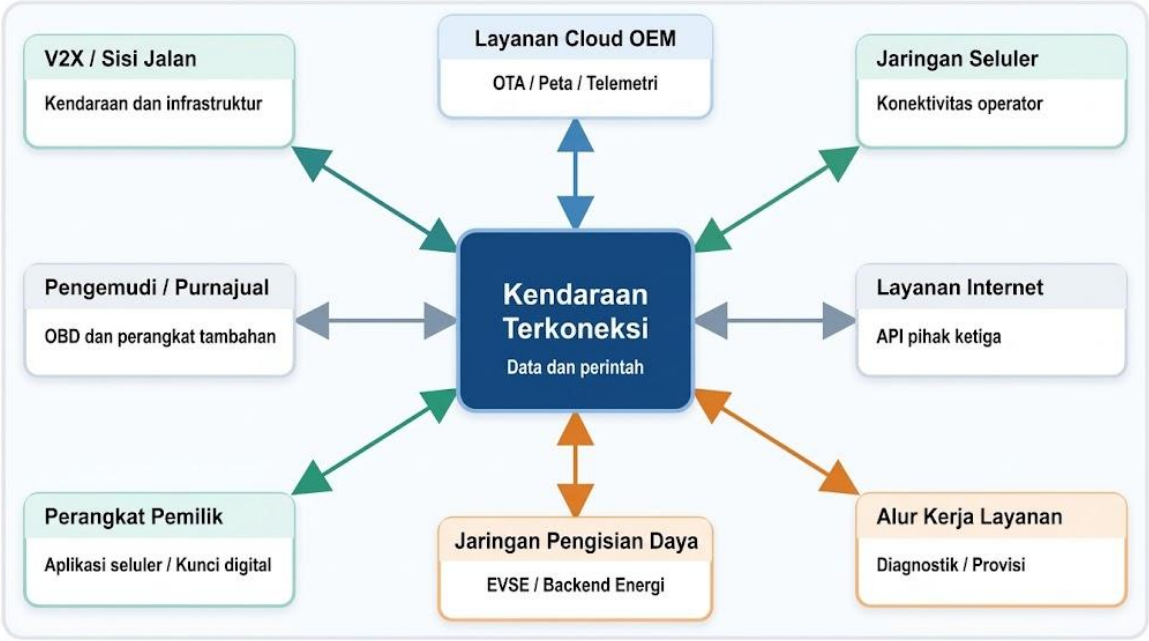
Seiring berkembangnya kendaraan menjadi platform yang sangat terhubung dan intensif data, Ethernet muncul sebagai pendukung utama bagi aplikasi otomotif canggih. Ethernet merupakan protokol yang relatif baru namun berkembang pesat di sektor otomotif, menghadirkan kecepatan data tinggi dan fleksibilitas yang selama ini lazim ditemukan dalam jaringan perusahaan (*enterprise*). Dengan kecepatan mulai dari 100 Mbps hingga 10 Gbps, Ethernet sangat cocok untuk ECU berkinerja tinggi dan sistem kompleks yang membutuhkan komunikasi cepat dan andal seperti sistem untuk kemudi otonom, fusi sensor waktu nyata, serta sistem infotainment.

Berbeda dengan LIN dan CAN yang dirancang khusus untuk penggunaan otomotif, Ethernet berakar pada jaringan perusahaan (*enterprise networking*). Penerapannya dalam industri otomotif menandai pergeseran menuju kerangka komunikasi yang lebih terbuka dan dapat diskalakan. Kemampuan Ethernet dalam menangani data dalam jumlah besar menjadikannya solusi ideal untuk aplikasi ber-bandwidth tinggi, seperti transmisi data video dari kamera, radar, dan sistem LiDAR komponen-komponen yang sangat penting bagi rangkaian teknologi ADAS yang terus berkembang.

Namun, penerapan Ethernet pada kendaraan menghadirkan tantangan tersendiri. Lingkungan otomotif sangat berbeda dengan jaringan TI tradisional, sehingga Ethernet perlu diadaptasi agar memiliki tingkat ketahanan, keamanan, dan kinerja waktu nyata (*real-time*) yang lebih tinggi. Pengembangan *Automotive Ethernet* suatu versi protokol yang dirancang khusus untuk penggunaan pada kendaraan telah menjawab banyak tantangan tersebut, termasuk kebutuhan akan pengaturan waktu yang deterministik, latensi yang lebih rendah, dan keamanan yang ditingkatkan. Selain itu, seiring dengan semakin matangnya penggunaan Ethernet di sektor otomotif, teknologi ini harus diintegrasikan dengan sistem lama (*legacy systems*) yang sudah ada, seperti CAN, guna menciptakan arsitektur jaringan hibrida yang memadukan keunggulan dari kedua sistem tersebut. Terkait aspek keamanan, Ethernet menawarkan berbagai protokol keamanan yang diadopsi dari sistem perusahaan (*enterprise systems*); namun, mengaktifkan seluruh protokol tersebut tidak selalu dimungkinkan karena dapat berdampak pada kinerja sistem yang beroperasi secara waktu nyata.

Arsitektur E/E, platform perangkat keras, lapisan perangkat lunak, dan jaringan komunikasi kendaraan yang telah dijelaskan sejauh ini menentukan bagaimana fungsi-fungsi diimplementasikan di dalam kendaraan. Namun, kendaraan yang terhubung tidak hanya

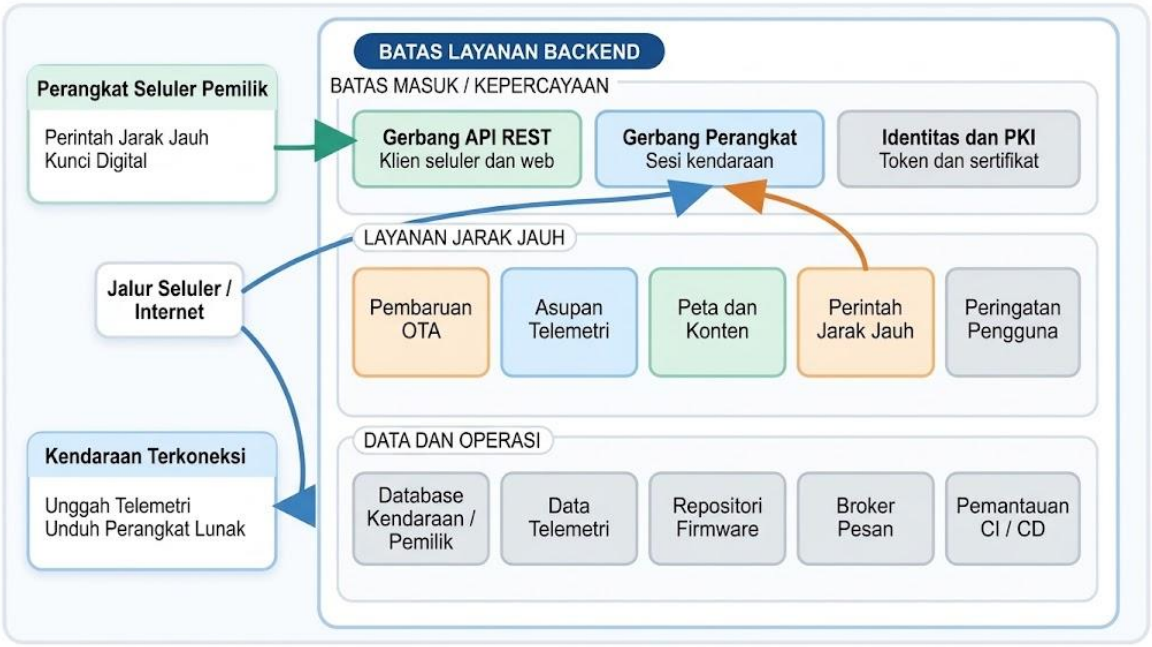
beroperasi dalam lingkup arsitektur internalnya saja. Unit kontrol elektronik (ECU), antarmuka, dan layanan pada kendaraan tersebut berinteraksi dengan berbagai sistem eksternal, seperti sistem *backend* berbasis cloud, peralatan diagnostik, infrastruktur pengisian daya, dan jaringan nirkabel. Oleh karena itu, untuk memahami dari mana serangan dapat berasal dan bagaimana serangan tersebut dapat menyebar, kita perlu memperluas sudut pandang kita dari desain internal kendaraan menuju lingkungan operasional yang lebih luas tempat kendaraan tersebut beroperasi.



Gambar 1.5: Sumber ancaman terhadap kendaraan yang terhubung

1.3 EKOSISTEM KENDARAAN TERHUBUNG DAN PERMUKAAN SERANGAN SIBER

Langkah awal untuk memahami lingkungan operasional kendaraan serta luasnya permukaan serangan (*attack surface*) adalah dengan mengidentifikasi sistem-sistem yang berinteraksi dengan kendaraan di dalam lingkungan operasional tersebut. Sebagaimana ditunjukkan pada Gambar 1.5, kendaraan modern tidak lagi dapat dipandang sebagai entitas yang terisolasi, melainkan merupakan bagian dari ekosistem yang terintegrasi, kompleks, dan saling terhubung. Ekosistem ini menyediakan konektivitas berkelanjutan ke berbagai sistem pendukung, seperti server *backend* yang dioperasikan oleh produsen kendaraan atau penyedia layanan, jaringan seluler, perangkat pribadi pemilik kendaraan, bahkan internet secara lebih luas. Setiap koneksi menghadirkan jalur potensial bagi ancaman keamanan siber.



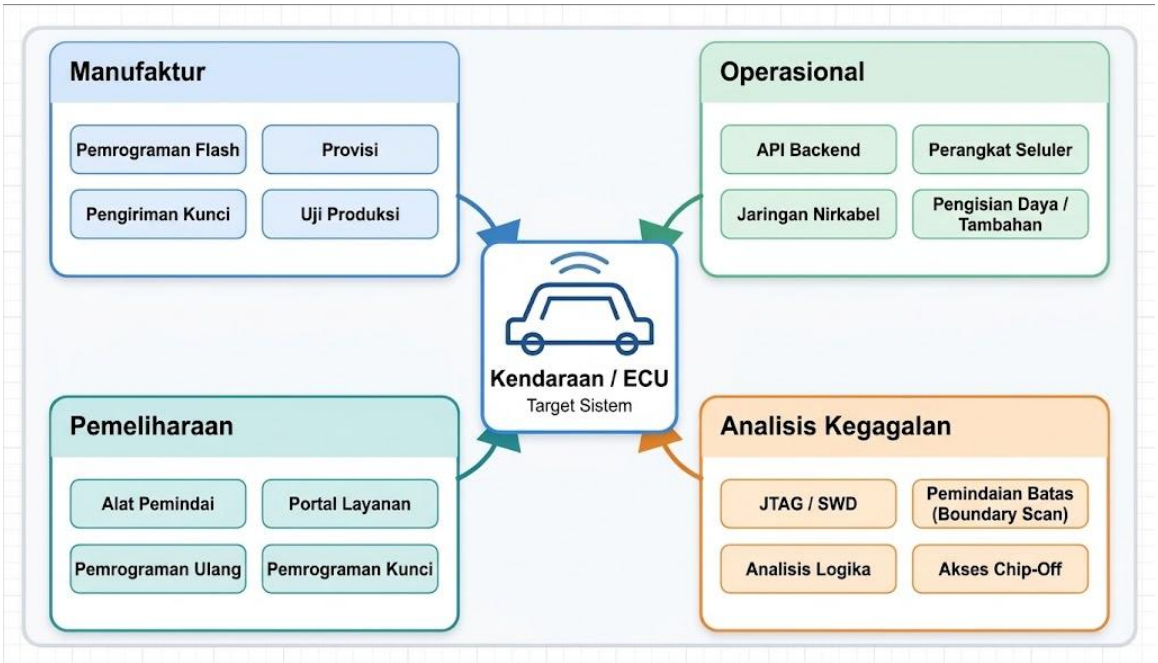
Gambar 1.6: Contoh arsitektur *backend* kendaraan terhubung untuk layanan jarak jauh

Tanpa integrasi dengan server *backend*, pengemudi tidak akan bisa menerima pembaruan perangkat lunak secara berkala, mengunggah data telemetri, mengunduh pembaruan peta, atau mengakses berbagai fungsi yang kini telah menjadi standar dalam kendaraan modern, sebagaimana ditunjukkan pada Gambar 1.6. Meskipun aliran komunikasi yang berkelanjutan ini sangat penting untuk efisiensi operasional dan peningkatan pengalaman pengguna, hal ini secara signifikan memperluas *attack surface* (area yang rentan terhadap serangan), sehingga menciptakan banyak peluang untuk menyusun rangkaian serangan terhadap sistem kendaraan.

Demikian pula, pemilik kendaraan kini terbiasa menghubungkan perangkat seluler mereka ke kendaraan, yang memungkinkan mereka mengakses aplikasi saat mengemudi serta menjalankan fungsi jarak jauh seperti membuka kunci pintu, menyalakan mesin, atau melacak lokasi kendaraan melalui ponsel pintar. Namun, jika saluran-saluran ini tidak diamankan dengan baik, akan muncul risiko keamanan siber tambahan. Di luar operasional normal, berbagai perangkat dan peralatan seperti perangkat diagnostik, pengembangan, dan manufaktur dapat terhubung dengan arsitektur kelistrikan/elektronik (E/E) kendaraan. Jika keamanan perangkat-perangkat ini diabaikan, keamanan lingkungan operasional kendaraan secara keseluruhan tidak dapat dijamin. Selain itu, penggunaan perangkat tambahan (*aftermarket*) dan layanan pihak ketiga menambah tingkat kompleksitas lainnya. Perangkat-perangkat ini, mulai dari dongle OBD hingga sistem hiburan aftermarket yang canggih, mungkin mengandung kerentanan yang tidak diketahui, yang dapat menjadi titik masuk serangan ke dalam jaringan internal kendaraan.

Lebih lanjut, perkembangan kendaraan terhubung kini mencakup kemampuan komunikasi langsung dengan infrastruktur (*Vehicle-to-Infrastructure* atau V2I) dan dengan kendaraan lain (*Vehicle-to-Vehicle* atau V2V). Meskipun kemajuan ini menjanjikan peningkatan

signifikan dalam aspek keselamatan dan manajemen lalu lintas, hal tersebut juga menimbulkan kekhawatiran terkait keamanan siber karena memperluas *attack surface*. Gambar 1.7 mengilustrasikan paparan ancaman pada berbagai tahapan siklus hidup kendaraan.



Gambar 1.7: Paparan ancaman dalam tahapan siklus hidup kendaraan

Selama proses manufaktur, komponen kendaraan berinteraksi dengan perangkat pemrograman *flash*, perangkat lunak penyediaan (*provisioning*), perangkat konfigurasi fuse, serta berbagai layanan pengelolaan kunci yang bertujuan untuk memasang dan mengatur perangkat lunak dan perangkat keras agar berada dalam kondisi yang aman dan terjamin. Setelah kendaraan dirakit dan keluar dari pabrik, kendaraan tersebut memasuki mode operasional. Pada tahap ini, berbagai komponen lingkungan operasional mulai terlibat, seperti *server backend*, perangkat seluler, stasiun pengisian daya, bahkan peralatan tambahan (*aftermarket*) yang menghadirkan fitur konektivitas baru.

Terkadang, kendaraan perlu memasuki mode pemeliharaan (*maintenance mode*) untuk keperluan servis di dealer atau bengkel. Dalam kondisi ini, berbagai perangkat diagnostik berinteraksi dengan kendaraan, yang dapat digunakan untuk mendiagnosis serta mengubah perilaku kendaraan jika diperlukan. Terakhir, terdapat kasus khusus di mana unit kontrol elektronik (ECU) menunjukkan perilaku anomali yang sulit didiagnosis menggunakan perangkat diagnostik standar. Dalam situasi ini, ECU tersebut akan ditarik dari penggunaan normal, dan perangkat khusus seperti debugger berbasis *Joint Test Action Group* (JTAG) atau peralatan uji yang hanya tersedia bagi pihak produsen akan digunakan untuk melakukan analisis kegagalan yang bersifat invasif. Dengan memahami setiap tahapan siklus hidup dan komponen yang terlibat di dalamnya, seorang penyerang dapat memfokuskan diri pada bagian yang paling mungkin memiliki kelemahan yang dapat dieksploitasi. Setelah memahami

berbagai lapisan arsitektur elektronik/elektrikal (E/E) pada kendaraan, kita akan beralih membahas profil penyerang untuk mengetahui siapa saja pihak yang mungkin ingin menyerang sistem otomotif dan apa tujuan mereka.

1.4 PROFIL ANCAMAN DAN PSIKOLOGI DALAM KEAMANAN SIBER OTOMOTIF

Membangun konsensus di dalam organisasi mengenai profil penyerang untuk sistem yang sedang diamankan merupakan salah satu topik terpenting, namun sering kali paling jarang dibahas. Profil penyerang yang dirumuskan dengan jelas akan memengaruhi cara ancaman dinilai serta tingkat ketatnya keamanan siber yang diperlukan untuk menanganinya. Tanpa adanya konsensus, organisasi dapat membuang waktu dan sumber daya untuk memperdebatkan ancaman mana yang perlu dipertimbangkan dan seberapa besar tingkat keamanan yang harus diterapkan.

Di satu sisi spektrum, terdapat organisasi yang menganggap keamanan siber sekadar sebagai pemenuhan daftar periksa (*checkbox exercise*), dengan melakukan upaya minimal demi memenuhi ketentuan kepatuhan. Pada organisasi semacam ini, profil penyerang jarang didokumentasikan. Saat ditanya terhadap pihak mana sistem mereka harus tangguh, para teknisi sering memberikan jawaban yang beragam mulai dari peretas amatir (*script kiddies*) hingga aktor ancaman tingkat lanjut. Tanpa konsep profil penyerang yang jelas, pengembang sering kali hanya mengandalkan intuisi dalam menentukan serangan mana yang perlu dipertimbangkan dan mana yang bisa diabaikan.

Di sisi lain spektrum, terdapat organisasi yang berambisi membangun produk yang mampu bertahan dari penyerang paling canggih, termasuk ancaman persisten tingkat lanjut (*Advanced Persistent Threats*/APT) yang didukung oleh negara. Meskipun ambisi ini patut dipuji, hal tersebut dapat memicu ketegangan jika tidak ada kesepakatan antara pihak manajemen, tim teknis, dan divisi bisnis mengenai tingkat keamanan yang ditargetkan. Sebagai contoh, dalam menghadapi ancaman manipulasi sensor, tim keamanan mungkin mengusulkan penerapan kontrol keamanan berlapis yang ketat bahkan hingga ke pemilihan jenis sensor tertentu sementara divisi bisnis mungkin menolak biaya tambahan untuk pengadaan sensor yang lebih aman. Pihak manajemen pun mungkin keberatan dengan dampak terhadap jadwal kerja dan keterbatasan sumber daya yang diperlukan untuk mendukung sensor tersebut. Akibatnya, terjadi perdebatan yang seolah tak berujung di antara para pemangku kepentingan mengenai seberapa penting risiko tersebut dan tindakan apa yang dianggap tepat untuk memitigasinya. Untuk menghindari perdebatan semacam ini, organisasi sebaiknya mendokumentasikan profil penyerang mereka terlebih dahulu, lalu memastikan seluruh pemangku kepentingan menyepakati tingkat ketahanan keamanan yang diperlukan. Setelah konsensus tercapai, risiko keamanan siber dapat dianalisis secara sistematis untuk menangani risiko-risiko yang termasuk dalam lingkup yang telah disepakati bersama.

Bagi pihak penyerang, mengetahui tingkat kematangan organisasi keamanan target bisa sangat berguna, karena hal tersebut dapat memberikan petunjuk mengenai kapabilitas penyerang apa saja yang telah dipertimbangkan oleh organisasi tersebut, serta di mana letak

keterbatasan keamanan dalam produknya. Gambar 1.8 menyajikan tingkat-tingkat penyerang dan eskalasi kapabilitasnya.



Gambar 1.8: Level penyerang dan peningkatan kemampuan

Sekarang mari kita lihat level-level ini secara detail.

Level Penyerang

Seperti yang ditunjukkan pada Gambar 1.8 kemampuan penyerang meningkat seiring dengan bertambahnya level yang dimulai dengan skrip kiddies dan diakhiri dengan negara bangsa seperti yang akan kita lihat selanjutnya.

Anak-Anak Skrip

Penyerang memiliki tingkat kecanggihan yang bervariasi dan motivasi yang beragam. Di tingkat terendah adalah penghobi dan pecinta skrip, yang tahu cara mengunduh skrip berbahaya dan menjalankannya pada perangkat target. Orang-orang ini sering kali berusaha memodifikasi kendaraannya dengan cara menghindari pembatasan OEM. Misalnya, penyetel mesin mungkin memasang set kalibrasi baru untuk meningkatkan tenaga kuda atau menggunakan alat diagnostik OBD untuk mengurangi jarak tempuh guna menghindari membayar kelebihan sewa atau menjual kendaraan mereka dengan harga lebih tinggi. Meskipun keterampilan teknis mereka terbatas, mereka masih dapat menimbulkan kerugian finansial pada korbannya dengan mengeksploitasi kerentanan sederhana.

Individu yang Termotivasi

Berikutnya adalah individu yang termotivasi, seperti karyawan yang merasa tidak puas atau penyerang yang didorong oleh ideologi tertentu (aktivis peretas/*hacktivist*). Meskipun pendanaan mereka mungkin terbatas, tindakan mereka tetap dapat menimbulkan risiko keamanan siber yang signifikan, terutama bagi produsen peralatan asli (OEM). Individu-individu ini mungkin meneliti peretasan dan kerentanan untuk menyusun rangkaian serangan,

membeli perangkat dari dark web, atau memanfaatkan pengetahuan khusus yang mereka peroleh dari tempat kerja mereka.

Pelaku ancaman dari dalam (*insider threat*) sangat berbahaya karena mereka memiliki akses ke informasi penting dan dapat menyebabkan kerusakan parah, seperti meluncurkan pembaruan perangkat lunak yang tidak kompatibel atau menyisipkan kode berbahaya ke dalam basis kode selama tahap pengembangan. Ancaman dari dalam meskipun tidak tersebar luas juga dapat mencakup kegiatan spionase demi keuntungan finansial, di mana oknum menjual kekayaan intelektual yang berharga kepada pemerintah asing atau pihak pesaing.

Kelompok Kriminal

Kelompok kriminal mewakili tingkat kecanggihan berikutnya. Kelompok ini mempekerjakan peretas terampil untuk merancang serangan khusus, sering kali menawarkan layanan "peretas sewaan" (*hackers-for-hire*). Di sektor otomotif, kelompok semacam ini memproduksi peralatan untuk pencurian kendaraan, dengan terus mengembangkan perangkat mereka agar tetap selangkah lebih maju dari sistem pertahanan keamanan. Mereka memiliki pemahaman mendalam tentang teknologi informasi (TI) dan sistem tertanam (*embedded systems*), serta mampu menemukan dan mengeksploitasi kerentanan.

Peneliti dan Peretas Etis

Peneliti dan peretas etis yang biasanya termotivasi oleh pengakuan akademis atau imbalan finansial juga harus diperhitungkan dalam profil penyerang. Meskipun mereka umumnya tidak berniat menimbulkan kerugian, pengungkapan kerentanan yang tidak tepat dapat berdampak serius secara finansial maupun reputasi jika organisasi terkait tidak segera menangani masalah tersebut.

Kejahatan Terorganisasi

Pada tingkat kecanggihan yang lebih tinggi terdapat kelompok kejahatan terorganisasi yang beroperasi secara multinasional. Kelompok ini menggunakan teknik seperti rekayasa sosial (*social engineering*) dan ransomware. Meski keberadaan mereka di sektor otomotif belum banyak didokumentasikan, dapat dibayangkan skenario di mana mereka menargetkan kendaraan tertentu demi tujuan politik atau finansial. Sebagai contoh, klien yang bersedia membayar mahal mungkin ingin melumpuhkan model kendaraan tertentu secara "tidak sengaja" dan tanpa meninggalkan jejak, guna menekan produsen kendaraan (OEM) agar membayar uang tebusan.

Aktor Negara

Penyerang yang paling canggih adalah kelompok ancaman tingkat lanjut (*Advanced Persistent Threats* atau APT) yang didukung oleh negara, dengan motivasi yang didorong oleh agenda politik maupun ekonomi. Para penyerang ini memiliki sumber daya yang besar serta akses ke perangkat khusus yang bersifat eksklusif. Mereka bahkan bisa menyamar sebagai aktivis peretas (*hacktivists*) untuk mengaburkan keterlibatan mereka. Kemampuan yang mereka miliki menjadikan mereka lawan yang sangat tangguh, dengan serangan yang sering kali dirancang secara rahasia layaknya strategi pertahanan nasional.

Kelompok penyerang ini juga mungkin memiliki kemampuan untuk menyusup ke dalam rantai pasok melalui serangan canggih yang sulit dideteksi, seperti penyisipan sensor palsu,

kloning cip, atau perangkat keras jaringan tiruan. Mengingat kompleksitas rantai pasok otomotif, sangat mungkin bagi kelompok APT untuk menargetkan titik lemah dalam rantai pasok tersebut guna mencapai tujuan mereka mirip dengan kasus SolarWinds, di mana penargetan dilakukan untuk berdampak pada perusahaan swasta maupun entitas pemerintah yang menggunakan perangkat lunak tersebut.

Dalam spektrum antara peretas amatir (*script kiddies*) hingga aktor negara, sebagian besar perusahaan otomotif merancang sistem mereka agar tangguh dalam menghadapi kelompok kriminal. Hal ini tercermin dari tingkat kendali keamanan yang mereka terapkan. Sebagai contoh, saat merancang perlindungan untuk kode yang dijalankan di dalam lingkungan eksekusi tepercaya (*trusted execution environment*), beberapa perusahaan mungkin memutuskan bahwa penyerang yang mampu menembus sistem khusus semacam itu berada di luar cakupan, sehingga analisis keamanan mereka dihentikan pada tingkat tersebut. Di sisi lain, organisasi yang bertujuan untuk memberikan perlindungan terhadap ancaman tingkat negara harus melangkah lebih jauh, dengan memastikan ketahanan bahkan terhadap serangan yang sangat canggih dan berlapis. Setelah profil penyerang disepakati, para insinyur keamanan defensif harus tetap konsisten dengan profil tersebut, terutama selama fase pemodelan ancaman.

Psikologi Pengembang

Keamanan bukan sekadar masalah teknologi, melainkan juga berkaitan erat dengan psikologi manusia. Dengan memahami psikologi pengembang, kita dapat memprediksi jenis kelemahan keamanan yang mungkin terdapat dalam produk mereka. Begitu pula, dengan memahami psikologi penyerang, kita dapat mengantisipasi sejauh mana mereka akan bertindak untuk mencapai tujuan mereka, serta apa saja tujuan tersebut.

Pengembang menghadapi berbagai tantangan saat merancang dan membangun sistem otomotif, dan penambahan batasan keamanan pada sistem tersebut justru meningkatkan kompleksitas pengembangan. Sebagai contoh, melakukan debugging pada jaringan CAN yang mengalami kesalahan *timeout* sinyal secara berkala bisa menjadi tugas yang sangat sulit. Bayangkan jika jaringan tersebut telah menerapkan autentikasi pesan CAN, sementara pengembang yang mencoba memperbaiki masalah tersebut tidak memiliki kunci kriptografi yang diperlukan untuk mengirim pesan yang sah dan terautentikasi. Dalam situasi ini, pengembang yang harus menyelesaikan masalah di tengah tenggat waktu yang ketat mungkin terpaksa menonaktifkan autentikasi tersebut dengan membuat *backdoor* yang kemudian tetap ada di dalam sistem bahkan setelah produk masuk ke tahap produksi!

Skenario fiktif ini lazim ditemukan dalam bidang rekayasa otomotif, di mana psikologi dalam pemecahan masalah dapat menjadi sumber kelemahan keamanan baru yang tertinggal dan menunggu untuk dieksploitasi. Dalam contoh di atas, pengembang seharusnya bekerja sama secara erat dengan insinyur keamanan untuk memungkinkan penonaktifan autentikasi melalui mekanisme yang aman seperti opsi *build* yang dihapus saat sistem memasuki tahap produksi. Para profesional keamanan ofensif yang memahami psikologi pengembang dapat mengantisipasi keberadaan kerentanan semacam itu, karena *backdoor* (pintu belakang) dan jalan pintas merupakan hal yang mungkin terjadi.

Demikian pula, sering ditemukan fitur-fitur sisa yang sebenarnya hanya ditujukan untuk pengembangan seperti log terperinci (*verbose logs*) yang mengungkap data sensitif, driver pengujian khusus, atau bahkan kunci kriptografi yang hanya untuk pengembangan namun masih ada dalam sistem yang ditujukan untuk produksi. Dengan memahami faktor-faktor psikologis ini, para profesional keamanan ofensif dapat mempersempit area pencarian mereka saat berupaya menemukan kelemahan keamanan.

Taktik Psikologis

Penyerang sering mengeksploitasi kelemahan psikologis manusia melalui rekayasa sosial (*social engineering*), dengan memanfaatkan rasa takut, kelangkaan, dan urgensi untuk mendorong korban agar mengambil keputusan yang buruk. Sebagai contoh, penyerang mungkin mengirim surel atau pesan berisi tautan berbahaya yang menciptakan kesan urgensi palsu, sehingga pengguna terdorong untuk mengabaikan pemikiran kritis. Meskipun rekayasa sosial langsung memiliki cakupan yang lebih terbatas dalam sistem otomotif dibandingkan dalam lingkungan TI perusahaan, penyerang tetap dapat memanfaatkan urgensi misalnya saat respons insiden, pembaruan perangkat lunak, interaksi dengan pemasok, atau operasional layanan untuk memengaruhi pengambilan keputusan dan melewati prosedur pengamanan standar.

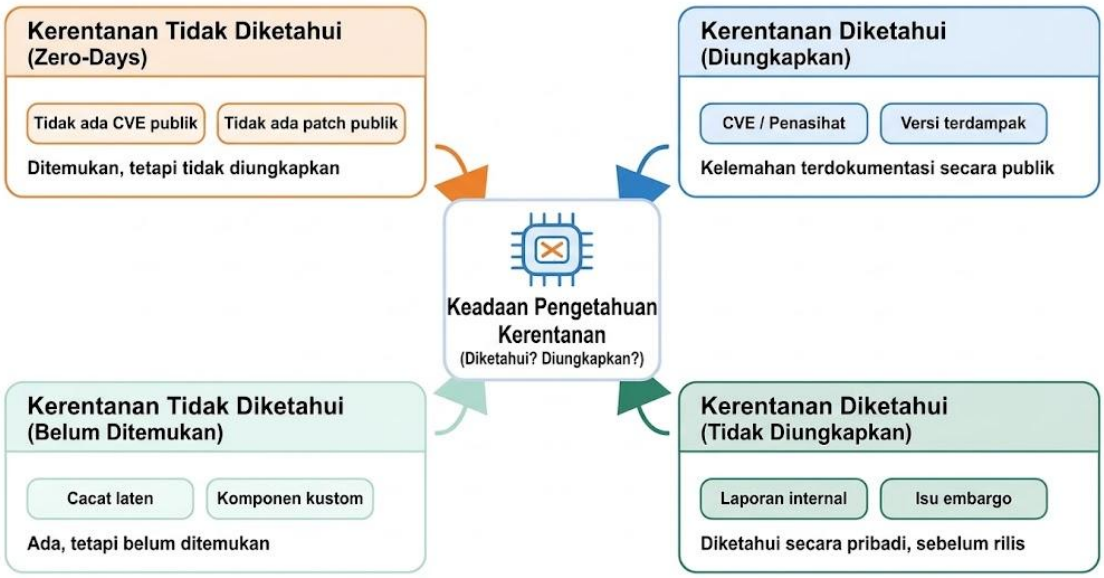
Kelangkaan adalah taktik lain yang digunakan penyerang untuk menekan korban dengan menciptakan kesan bahwa waktu, akses, dukungan, atau sumber daya yang tersedia sangat terbatas. Dalam lingkungan otomotif, hal ini bisa berupa tindakan menyamar sebagai pemasok, penyedia layanan, atau pemangku kepentingan internal, lalu mengeklaim bahwa tindakan segera harus diambil sebelum jendela pembaruan ditutup, akses kedaluwarsa, atau sistem krusial menjadi tidak tersedia.

Rasa takut juga dapat digunakan untuk memanipulasi pengambilan keputusan. Penyerang mungkin melebih-lebihkan konsekuensi dari tidak mengambil tindakan guna menekan personel agar menyetujui perubahan yang tidak aman, mengungkapkan informasi sensitif, atau mengabaikan prosedur yang telah ditetapkan. Di sisi lain, organisasi harus berhati-hati agar budaya keamanan internal tidak hanya didorong oleh rasa takut, karena hal ini dapat menyebabkan tim hanya berfokus pada ancaman yang paling terlihat, sementara risiko yang lebih samar namun tetap berbahaya justru terabaikan. Setelah memahami berbagai kapabilitas penyerang dan pentingnya menyusun profil penyerang yang tepat, kini saatnya menelaah konsep dasar mengenai kelemahan (*weaknesses*), kerentanan (*vulnerabilities*), dan ancaman (*threats*), serta bagaimana istilah-istilah tersebut saling berkaitan.

1.5 KLASIFIKASI DAN PENILAIAN KERENTANAN PADA SISTEM OTOMOTIF

Menurut NIST, kerentanan (*vulnerability*) adalah "*kelemahan dalam sistem informasi, prosedur keamanan sistem, pengendalian internal, atau implementasi yang dapat dieksploitasi atau dipicu oleh sumber ancaman*". Untuk sistem otomotif, definisi yang sama berlaku, dengan sedikit penyesuaian bahwa kelemahan dapat ditemukan pada komponen mana pun di dalam kendaraan atau sistem pendukungnya. Sementara itu, kelemahan

(*weakness*) didefinisikan sebagai "cacat atau karakteristik yang dapat menyebabkan perilaku yang tidak diinginkan".



Gambar 1.9: Jenis-jenis kerentanan berdasarkan status penemuannya

Tentu saja, adanya kelemahan membuat suatu sistem lebih rentan terhadap penyalahgunaan oleh penyerang. Dari perspektif ini, setiap kerentanan dapat dianggap sebagai kelemahan, namun tidak semua kelemahan serta-merta merupakan kerentanan. Terkait eksploitasi, para insinyur keamanan defensif berfokus untuk menghilangkan kelemahan sejak tahap perancangan guna mengurangi risiko eksploitasi, sedangkan para profesional keamanan ofensif mencari kelemahan yang luput dari perhatian tim defensif dan berupaya mengeksploitasinya. Kerentanan tidak hanya dibedakan berdasarkan penyebab teknisnya, tetapi juga berdasarkan status pengetahuannya: siapa yang mengetahui cacat tersebut, apakah cacat itu telah ditemukan, dan apakah informasi mengenainya telah diungkapkan kepada publik, sebagaimana ditunjukkan dalam Gambar 1.9.

Kategori pertama adalah kerentanan yang tidak diketahui dan belum ditemukan, yang pada dasarnya merupakan cacat laten dalam perangkat lunak, perangkat keras, protokol, logika integrasi, atau kondisi operasional yang tidak lazim pada ECU yang belum diidentifikasi oleh peneliti, pihak pertahanan, maupun penyerang. Ini adalah kelemahan tersembunyi yang menjadi sasaran pengujian keamanan untuk diungkap.

Setelah suatu cacat ditemukan, statusnya pun berubah. Jika cacat tersebut hanya diketahui oleh penyerang atau peneliti dan belum diungkapkan kepada pihak pertahanan yang terdampak, maka hal ini umumnya dikategorikan sebagai *zero-day*. Keistimewaan *zero-day* bukan sekadar karena adanya cacat keamanan, melainkan karena pihak pertahanan memiliki "nol hari" (*zero days*) untuk bersiap mengingat tidak adanya pemberitahuan publik, entri CVE, maupun *patch* atau mitigasi yang tersedia secara luas. Menemukan dan mengeksploitasi celah semacam ini sering kali memerlukan upaya penelitian yang besar; oleh karena itu, penyerang

cenderung menyimpannya untuk target bernilai tinggi, di mana keuntungan yang diperoleh sepadan dengan biaya dan risiko keterpaparan yang dihadapi.

Sebuah kerentanan juga bisa saja sudah diketahui namun belum diungkapkan ke publik misalnya, jika kerentanan tersebut dilacak secara privat oleh OEM, pemasok, peneliti, atau penyerang. Contohnya mencakup laporan bug internal, masalah pada komponen pemasok yang belum diperbaiki, atau temuan penelitian yang masih dalam masa embargo. Sebaliknya, kerentanan yang sudah diketahui dan diungkapkan bersifat publik: kerentanan ini muncul dalam bentuk pemberitahuan keamanan (*advisories*), entri CVE, daftar versi yang terdampak, atau panduan mitigasi. Kerentanan jenis ini sering kali menjadi pilihan utama bagi penyerang karena mereka dapat mengidentifikasi versi yang rentan, menggunakan kembali detail teknis yang sudah tersedia untuk publik, serta menggabungkan beberapa kelemahan yang sudah diketahui semuanya dilakukan sebelum pihak pertahanan sempat menerapkan perbaikan (*patch*) secara menyeluruh pada sistem mereka.

Mencari kerentanan yang sudah diketahui merupakan teknik yang umum digunakan ketika sistem target mengintegrasikan komponen siap pakai (*off-the-shelf*). Secara tradisional, sistem otomotif sangat bergantung pada solusi kustom, sehingga eksploitasi terhadap kerentanan yang sudah diketahui hanya memberikan hasil yang terbatas. Namun, seiring dengan pergeseran industri menuju kendaraan berbasis perangkat lunak (*Software-Defined Vehicles*/SDV) yang dilengkapi sistem komputasi berkinerja tinggi, penggunaan komponen siap pakai dan sumber terbuka (*open-source*) dalam kendaraan modern pun kian marak. Hal ini membuat eksploitasi terhadap kerentanan yang sudah diketahui menjadi lebih efektif. Meski demikian, terlepas dari meluasnya penggunaan komponen komersial siap pakai (*Commercial Off-The-Shelf*/COTS) dan perangkat lunak sumber terbuka (*Open-Source Software*/OSS), perangkat lunak maupun perangkat keras kustom masih banyak ditemukan; kondisi ini menuntut penemuan kerentanan *zero-day* agar sistem tersebut dapat ditembus secara efektif.

Demikian pula, upaya dalam industri otomotif untuk menstandardisasi platform perangkat lunak dan perangkat keras, serta berbagai protokol, membuat penemuan kerentanan *zero-day* menjadi jauh lebih menguntungkan. Pasalnya, penyerang dapat menemukan kerentanan pada komponen standar dan mengeksploitasinya di mana pun komponen tersebut digunakan.

Seni keamanan ofensif bukan sekadar menemukan kelemahan, melainkan juga memprioritaskan kelemahan yang paling mudah dieksploitasi. Seperti dalam profesi lainnya, pihak penyerang cenderung memilih jalan dengan hambatan paling minim, berupaya mencapai tujuan mereka dengan waktu dan upaya sesedikit mungkin. Jika suatu sistem memiliki banyak kelemahan yang mudah dieksploitasi, penyerang akan menyasar kelemahan-kelemahan tersebut terlebih dahulu untuk melihat apakah itu cukup untuk mencapai tujuan mereka. Hanya setelah kelemahan yang mudah dieksploitasi habis dimanfaatkan, penyerang akan beralih ke metode yang lebih canggih. Penyerang yang terampil bahkan dapat memanfaatkan kelemahan yang tampak mustahil untuk dieksploitasi guna menyusun rangkaian serangan yang efektif.

Sebagai contoh, salah satu eksploitasi yang sangat sulit dibayangkan yang pernah dievaluasi oleh *Project Zero* dari Google melibatkan penggunaan kerentanan dalam penguraian (*parsing*) format *Portable Document Format* (PDF) untuk membajak iMessage melalui trik berkas GIF palsu. Penyerang memanfaatkan bug pada pengurai PDF CoreGraphics milik Apple melalui standar kompresi lama, yaitu *Joint Bi-level Image Group 2* (JBIG2). Kerentanan ini memungkinkan mereka untuk menyusun eksploitasi kompleks berbasis gerbang logika yang meniru komputer *Turing-complete* di dalam kode pemrosesan gambar. Mereka kemudian menggunakan mekanisme ini untuk mengeksekusi perintah semuanya hanya melalui satu berkas gambar.

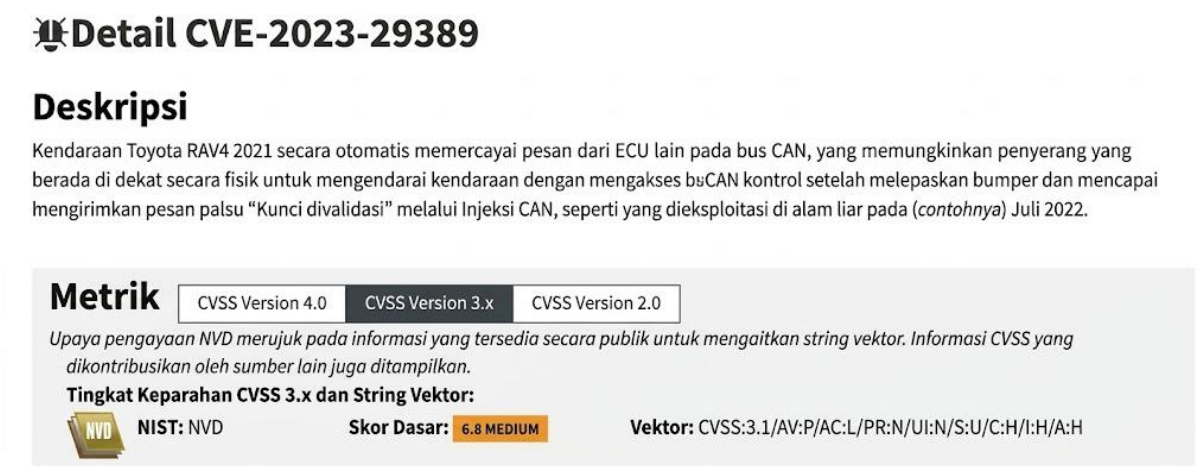
Tingkat kecanggihan dan kreativitas dalam mengubah teknik kompresi menjadi sebuah eksploitasi sungguh luar biasa. Sangatlah wajar jika seorang insinyur keamanan defensif menjumpai standar kompresi lama seperti JBIG2 dan tidak menyadari bahwa standar tersebut mengandung kelemahan yang dapat memicu serangan secanggih itu; namun, hal ini tidak menghalangi para profesional keamanan ofensif untuk memanfaatkannya dalam rangkaian serangan tingkat lanjut.

Skor Kerentanan

Tidak semua kerentanan memiliki tingkat yang sama; oleh karena itu, metode penilaian yang objektif sangat penting untuk menentukan tingkat keparahan suatu kerentanan. Penyerang biasanya lebih tertarik pada kerentanan dengan skor tinggi, karena kerentanan semacam itu cenderung lebih mudah dieksploitasi dan dapat menimbulkan dampak yang lebih parah. Salah satu sistem penilaian kerentanan yang umum digunakan adalah *Common Vulnerability Scoring System* (CVSS). Skema penilaian CVSS mempertimbangkan dampak terhadap integritas dan ketersediaan sistem berdasarkan kriteria berikut: "*Eksploitasi kerentanan akan mengakibatkan dampak pada integritas yang dapat menyebabkan cedera serius atau lebih buruk (kategori 'Marginal' atau lebih buruk, sebagaimana dijelaskan dalam standar IEC 61508) terhadap manusia atau pengguna*."

Common Vulnerabilities and Exposures (CVE) dan laporan kerentanan sangat penting untuk melakukan pengujian keamanan ofensif yang sukses. Setelah fase penemuan, penyerang dapat mengidentifikasi kerentanan apa saja yang ada dalam suatu produk dan mulai menyiapkan eksploitasi. Beberapa platform yang sudah mapan merupakan sumber informasi yang berguna mengenai kerentanan. *Cybersecurity and Infrastructure Security Agency* (CISA) menyediakan peringatan dan saran terkait ancaman serta kerentanan yang baru muncul. *Japan Computer Emergency Response Team* (JPCERT) menawarkan laporan kerentanan yang spesifik untuk wilayah Jepang. *National Vulnerability Database* (NVD), yang dikelola oleh NIST, menyediakan daftar lengkap mengenai kerentanan yang telah diketahui.

Kerentanan ini biasanya memiliki nomor CVE yang menunjukkan tahun ditemukannya, serta metrik yang digunakan untuk menentukan skor, sebagaimana ditunjukkan pada Gambar 1.10.



Gambar 1.10: Contoh CVE yang menampilkan ID dan skor

Di sisi lain, kerangka kerja seperti Common Weakness Enumeration (CWE) dan *Common Attack Pattern Enumeration and Classification* (CAPEC) dari MITRE menawarkan metode terstruktur untuk memahami serta mengategorikan kerentanan dan pola serangan. Contoh CWE yang umum dikenal adalah CWE-120: "Penyalinan Buffer tanpa Memeriksa Ukuran Input" (*Buffer Copy without Checking Size of Input*), yang menjelaskan kelemahan umum di mana data disalin dari input ke *buffer output* tanpa memverifikasi apakah ukuran data input lebih kecil atau sama dengan kapasitas *buffer output*. Akibatnya, input yang terlalu besar dapat menyebabkan *buffer overflow* (peluapan buffer). Sebelum kita mengakhiri bagian ini, penting untuk mengalihkan perhatian kita pada konsep mendasar lainnya, yaitu ancaman (*threats*).

Ancaman

Sementara kelemahan dan kerentanan bersifat internal terhadap suatu sistem, ancaman berasal dari sumber eksternal. NIST mendefinisikan ancaman sebagai "*suatu keadaan atau peristiwa yang berpotensi memberikan dampak buruk terhadap operasional organisasi (termasuk misi, fungsi, citra, atau reputasi), aset organisasi, atau individu melalui sistem informasi baik berupa akses tanpa izin, perusakan, pengungkapan, modifikasi informasi, dan/atau penolakan layanan (denial of service). Ancaman juga mencakup potensi bagi sumber ancaman untuk berhasil mengeksploitasi kerentanan tertentu dalam sistem informasi*." Dalam konteks sistem otomotif, ancaman dapat dipahami secara sederhana sebagai tindakan yang bertujuan untuk mengeksploitasi kerentanan guna mencapai tujuan serangan tertentu.

Sebuah sistem mungkin memiliki beberapa kerentanan, namun tanpa adanya pelaku ancaman yang aktif, sistem tersebut bisa saja tetap aman dari peretasan untuk waktu yang lama. Hal ini dapat menimbulkan anggapan keliru bahwa tingkat keamanan sistem tersebut sudah tangguh. Sebaliknya, sebuah sistem mungkin terpapar berbagai ancaman, tetapi jika kelemahan yang ada tidak dapat dieksploitasi secara praktis, peretasan mungkin tidak akan pernah terjadi. Kondisi ini dapat menciptakan rasa aman yang semu, yang mendorong organisasi untuk melonggarkan langkah-langkah keamanan karena tidak adanya insiden keamanan yang terjadi. Kedua kesalahpahaman ini perlu diluruskan, dan pengujian keamanan

ofensif (*offensive security testing*) merupakan salah satu cara untuk membuktikan kondisi suatu sistem melalui validasi keamanan yang ketat.

Dalam sistem otomotif, ancaman dapat berasal dari *backend*, perangkat pendukung, antarmuka diagnostik internal kendaraan (*onboard diagnostic interface*), perangkat seluler, serta berbagai sumber lain yang ada dalam lingkungan operasional. Ancaman-ancaman ini bertujuan untuk merusak aspek-aspek aset system seperti integritas, autentisitas, dan ketersediaan melalui tindakan manipulasi (*tampering*), penyamaran (*spoofing*), dan penolakan layanan (*denial of service*). Setelah memahami dasar-dasar mengenai kelemahan, kerentanan, dan ancaman, kini kita siap untuk mendalami ranah pengujian penetrasi (*penetration testing*), termasuk berbagai tahapan dan tingkat kedalamannya.

1.6 METODOLOGI DAN PELAKSANAAN PENGUJIAN PENETRASI OTOMOTIF

Meskipun sistem manajemen keamanan siber (*Cybersecurity Management System* atau CSMS), sebagaimana didefinisikan dalam standar ISO/SAE 21434, bertujuan untuk mencapai tingkat jaminan keamanan siber yang memadai bagi produk otomotif, cara terbaik untuk mengukur keamanan suatu sistem adalah dengan melihat perilakunya dalam dua skenario: saat menjalani pengujian penetrasi independen dan saat menghadapi serangan nyata. Pengujian penetrasi dilakukan berdasarkan ketentuan dari pemilik produk, dengan keunggulan berupa temuan yang dapat ditangani sebelum terjadi kerusakan nyata. Sementara itu, serangan nyata dilakukan oleh pihak lain-misalnya, penjahat siber dan dapat menimbulkan konsekuensi nyata, seperti kerugian finansial atau dampak buruk terhadap keselamatan. Oleh karena itu, saat membahas keamanan siber ofensif, kita dapat meninjaunya dari dua perspektif: perspektif penguji penetrasi dan perspektif penyerang di dunia nyata. Mungkin muncul pertanyaan, apa perbedaan di antara keduanya?

Jawaban sederhananya terletak pada aspek izin, niat, cakupan, dan akses terhadap informasi:

- Penguji penetrasi (*penetration tester*) memperoleh izin dari pemilik sistem target sebelum melancarkan kampanye serangan, sedangkan penyerang di dunia nyata tidak menunggu izin apa pun.
- Tujuan mereka pun berbeda. Satu kelompok berupaya melakukan kompromi terhadap sistem guna mengidentifikasi kelemahan yang dapat diperbaiki sebelum kelemahan tersebut dieksploitasi di dunia nyata, sementara kelompok lainnya melakukan kompromi demi keuntungan pribadi, finansial, politik, atau strategis. Perbedaan lainnya terletak pada cara masing-masing kelompok menangani temuan mereka. Penguji penetrasi (*penetration tester)* melaporkan kerentanan secara etis dan dalam kerangka hukum yang berlaku. Sebaliknya, penyerang di dunia nyata mungkin tidak akan pernah mengungkapkan temuan mereka, bisa jadi menjual informasi tersebut, atau justru mengungkapkannya hanya melalui pemerasan atau tuntutan tebusan.
- Ruang lingkup (*scope*) merupakan perbedaan penting lainnya. Penguji penetrasi (*penetration tester*) terikat oleh ruang lingkup yang telah disepakati, yang menentukan komponen, antarmuka, dan sistem pendukung mana saja yang boleh mereka uji. Hal ini dapat membatasi kemampuan mereka dalam menemukan jalur serangan

menyeluruh (*end-to-end*) yang melintasi batasan sistem. Sebagai contoh, pengujian yang dibatasi pada satu ECU (*Electronic Control Unit*) mungkin melewatkan kelemahan yang baru bisa dieksploitasi melalui layanan *backend*, ECU lain, atau aplikasi seluler. Demikian pula, penilaian terhadap kendaraan yang tidak mencakup stasiun pengisian daya (*charging station*) atau sistem *backend* pengisian daya mungkin gagal mendeteksi kerentanan yang muncul melalui komponen eksternal tersebut. Penyerang di dunia nyata tidak terikat oleh batasan ruang lingkup seperti itu dan dapat menempuh jalur apa pun yang menawarkan peluang keberhasilan terbesar.

- Sumber daya dan akses terhadap informasi sensitif juga berbeda. Penguji penetrasi biasanya bekerja dalam jangka waktu yang terbatas, serta terkendala oleh biaya dan ketersediaan tenaga ahli. Sebaliknya, penyerang di dunia nyata mungkin menghabiskan waktu berbulan-bulan atau bahkan bertahun-tahun untuk mencoba menembus suatu sistem, tergantung pada tingkat kecanggihan dan dukungan yang mereka miliki. Penguji penetrasi mungkin juga menerima informasi sensitif dari klien melalui pendekatan *gray-box* atau *white-box* (yang akan kita bahas lebih lanjut nanti dalam bab ini) agar pengujian menjadi lebih efektif. Sebaliknya, penyerang di dunia nyata umumnya memulai dari posisi *black-box*, tanpa akses resmi sebelumnya terhadap informasi semacam itu. Oleh karena itu, kita harus tetap menyadari keterbatasan pengujian penetrasi dalam mengungkap kelemahan keamanan secara menyeluruh jika dibandingkan dengan serangan yang terjadi di dunia nyata.

Meskipun terdapat perbedaan-perbedaan tersebut, kedua kelompok ini sering kali mengandalkan metodologi, perangkat, dan teknik serangan yang serupa.

Memanfaatkan Kerangka Kerja NIST

Karena pengujian penetrasi merupakan bidang yang sudah mapan, kita akan mempelajari keamanan ofensif dari sudut pandang penguji penetrasi. Namun, pertama-tama kita perlu menetapkan dasar mengenai apa itu pengujian penetrasi dalam konteks sistem otomotif. Pada bab-bab selanjutnya, kita akan membahas setiap fase dalam pelaksanaan pengujian penetrasi secara mendalam; untuk saat ini, kita akan memperkenalkan konsep-konsep dasarnya agar kita memiliki peta jalan untuk menjalankan inisiatif keamanan ofensif dengan sukses. Mengacu pada panduan NIST SP 800-115, kami mendefinisikan pengujian penetrasi (*penetration testing*) sebagai pendekatan sistematis untuk menilai keamanan suatu produk terhadap serangkaian tujuan keamanan melalui penggunaan berbagai metode, perangkat, dan teknik sebaiknya dilakukan sebelum sistem tersebut diterapkan di lapangan. Meskipun standar NIST menggunakan istilah "penilaian keamanan informasi" (*information security assessment*), kami tidak akan menggunakan istilah tersebut agar tidak tertukar dengan penilaian kepatuhan proses (*process compliance assessments*) yang lazim dilakukan dalam konteks standar ISO/SAE 21434 untuk sistem otomotif.

NIST Special Publication 800-115, "*Technical Guide to Information Security Testing and Assessment*," menyajikan pendekatan terstruktur untuk penilaian keamanan yang juga sangat relevan bagi keamanan ofensif pada sistem otomotif tertanam (*embedded automotive systems).* Dengan mengadaptasi konsep-konsep intinya, para profesional keamanan dapat

mengidentifikasi dan mengeksploitasi kerentanan dalam kendaraan modern secara lebih sistematis, sehingga meningkatkan postur keamanan siber secara keseluruhan.

Perencanaan penilaian keamanan merupakan langkah dasar. Proses ini dimulai dengan menetapkan ruang lingkup penilaian secara jelas, dengan fokus khusus pada elemen-elemen krusial seperti ECU, antarmuka komunikasi, dan perangkat lunak tertanam. Dengan menerapkan konsep-konsep tersebut pada keamanan ofensif dalam sistem otomotif, para profesional dapat mengadopsi pendekatan yang lebih disiplin dan efektif untuk mengidentifikasi serta mengeksploitasi kerentanan yang spesifik pada kendaraan.

ISO 21434 dan Pengujian Penetrasi (*Pentesting*)

ISO/SAE 21434 mendefinisikan pengujian penetrasi sebagai "*pengujian di mana serangan dunia nyata ditiru untuk mengidentifikasi cara-cara dalam mengompromikan tujuan keamanan siber*." Dengan demikian, ISO/SAE 21434 mengandalkan pengujian penetrasi sebagai metode untuk mengidentifikasi kerentanan serta sebagai praktik yang direkomendasikan untuk memvalidasi tujuan keamanan siber.

Catatan

***Pentesting* sebagai metode validasi:**
Kesalahan yang sering terjadi adalah menyamakan kegiatan validasi keamanan dengan pengujian verifikasi keamanan. Verifikasi keamanan harus dilakukan terlebih dahulu untuk memastikan bahwa kontrol keamanan yang telah ditetapkan telah diimplementasikan dengan benar dan berfungsi sebagaimana mestinya—seperti memastikan bahwa *secure boot* menolak perangkat lunak yang tidak sah, atau bahwa pembatasan akses diagnostik benar-benar ditegakkan. *Penetration testing* (pengujian penetrasi) kemudian melengkapi verifikasi tersebut dengan mengevaluasi sistem dari sudut pandang penyerang, serta mencari cara untuk menggagalkan tujuan keamanan siber meskipun kontrol keamanan telah diterapkan. Alih-alih menggantikan verifikasi, *penetration testing* membantu mengungkap jalur serangan yang tidak terduga, kelemahan konfigurasi, asumsi yang keliru, serta kerentanan yang sebelumnya tidak diketahui yang mungkin luput dari proses perancangan, implementasi, dan verifikasi yang lazim dilakukan.

Meskipun didefinisikan sebagai sebuah rekomendasi, terdapat ekspektasi umum dalam industri otomotif agar pengujian penetrasi (*penetration testing*) dilakukan pada tingkat tertentu, bergantung pada tingkat kekritisan produk yang tercermin dari tingkat paparan risikonya. Sudah menjadi hal yang lumrah bagi produsen peralatan asli (OEM) untuk mewajibkan pemasok melakukan pengujian penetrasi terhadap komponen-komponen tertentu yang dianggap sangat kritis; sebagai contoh, ECU telematika sering kali menjadi objek pengujian karena memiliki permukaan serangan (*attack surface*) yang menjangkau jarak jauh.

Demikian pula, OEM diharapkan untuk melakukan pengujian penetrasi (*penetration testing*) terhadap jaringan kendaraannya sendiri serta sistem pendukungnya, seperti sistem *backend*, lingkungan manufaktur, dan lingkungan layanan. Saat melaksanakan pengujian penetrasi, pihak terkait dapat memanfaatkan berbagai dokumen kerja lain yang ditetapkan dalam standar ISO/SAE 21434 seperti definisi item (*item definition*), model ancaman (*threat model*), spesifikasi keamanan siber, dan laporan analisis kerentanan sebagaimana ditunjukkan dalam Gambar 1.11.



Gambar 1.11: Pengujian penetrasi (*pentesting*) dalam konteks sistem manajemen keamanan siber

Sebagai contoh, selama fase penemuan (*discovery*), saat menentukan cakupan untuk pengujian penetrasi, definisi item dapat menjadi titik awal yang berguna karena memungkinkan kita untuk memahami elemen-elemen lingkungan operasional, batasan item, dan sistem target. Berbagi detail ini dengan penguji penetrasi dapat menghasilkan temuan yang lebih mendalam.

Analisis Ancaman dan Penilaian Risiko (*Threat Analysis and Risk Assessment*/TARA) merupakan sumber informasi bermanfaat lainnya yang dapat membantu dalam mengidentifikasi aset yang menjadi target pengujian penetrasi. Dalam hal ini, aset yang dianggap krusial bagi pengoperasian sistem yang tepat dan aman harus diprioritaskan dibandingkan aset lain yang kurang krusial. Tujuan keamanan siber yang diidentifikasi selama proses TARA dapat diterjemahkan menjadi tujuan serangan yang harus dicoba untuk dicapai dalam rangkaian pengujian penetrasi. Sederhananya, penguji penetrasi harus berupaya membuktikan bahwa pelanggaran terhadap tujuan keamanan sistem dapat terjadi dengan menemukan eksploitasi yang mampu melumpuhkan kemampuan sistem dalam memitigasi risiko-risiko tertentu.

Tergantung pada jenis pengujian penetrasi (*penetration test)*, spesifikasi keamanan siber dapat digunakan sebagai sumber masukan tambahan untuk mengidentifikasi kelemahan dalam desain keamanan serta kesenjangan dalam persyaratan keamanan. Selain melakukan pengujian penetrasi pada suatu produk, sebuah organisasi mungkin perlu menguji kemampuannya dalam merespons laporan kerentanan secara tepat waktu guna menambal (*patch*) masalah yang bersifat kritis. Latihan berbasis simulasi (*tabletop exercise*) seperti ini dirancang untuk meniru serangan di dunia nyata, dengan tujuan memastikan bahwa tim rekayasa keamanan defensif mengikuti prosedur manajemen insiden yang tepat, serta mampu menangani laporan tersebut dalam jangka waktu yang diharapkan agar masalah dapat segera diperbaiki. Selain itu, melakukan audit terhadap hasil analisis kerentanan dan proses

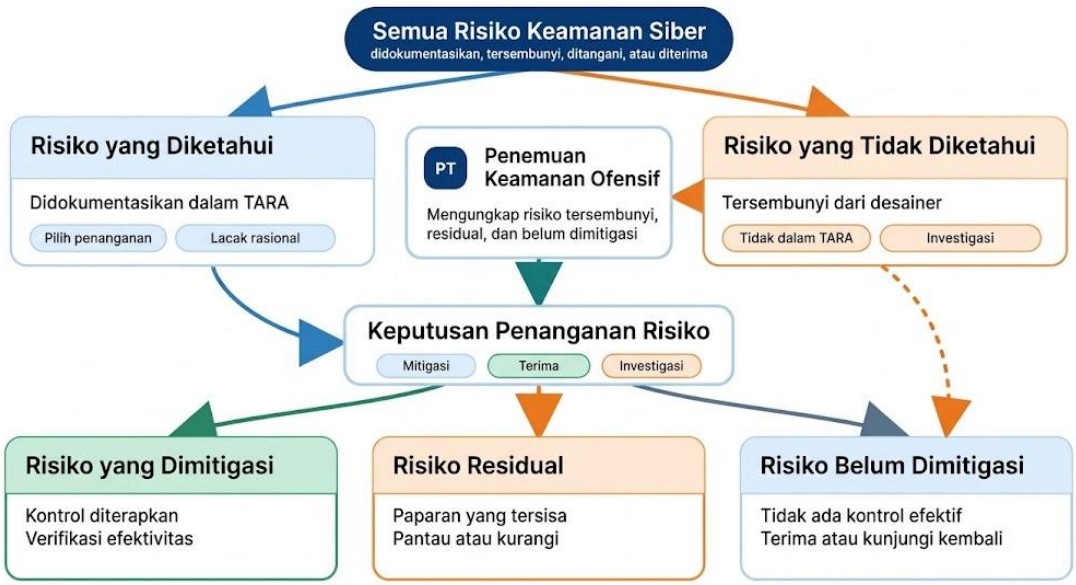
manajemennya merupakan praktik yang baik untuk memastikan bahwa kerentanan dinilai secara akurat dan tindakan perbaikan yang dipilih sudah tepat.

Menjamin Keamanan Siber Melalui Pengujian Penetrasi

Anda merancang sistem berdasarkan standar keamanan siber yang sistematis. Anda siap meluncurkan produk tersebut, namun belum ada pihak independen yang memvalidasi klaim keamanan Anda. Setelah produk diluncurkan, dalam hitungan minggu atau bulan, Anda mulai menerima laporan mengenai insiden keamanan siber, beberapa di antaranya bahkan dapat berdampak secara hukum, finansial, atau bahkan mengancam nyawa. Anda bertanya-tanya, bagaimana hal itu bisa terjadi? Setelah menghabiskan waktu berminggu-minggu menganalisis laporan kerentanan, Anda menyadari adanya serangkaian kelalaian yang tidak terdeteksi oleh pengujian keamanan rutin Anda. Apakah ada sesuatu yang bisa dilakukan untuk mencegahnya?

Jika membangun kontrol keamanan siber bertujuan untuk mengurangi risiko keamanan sistem hingga ke tingkat yang wajar, maka pengujian penetrasi berfungsi untuk memastikan bahwa risiko keamanan tersebut telah benar-benar ditekan hingga ke tingkat yang diinginkan. Mengingat sistem yang dikembangkan dengan tingkat jaminan keamanan tinggi sekalipun tetap akan memiliki risiko residual (sisa) serta risiko yang belum dimitigasi yang mungkin tidak diketahui oleh perancang sistem, diperlukan suatu metode untuk mengungkap risiko-risiko tersebut secara terkendali.

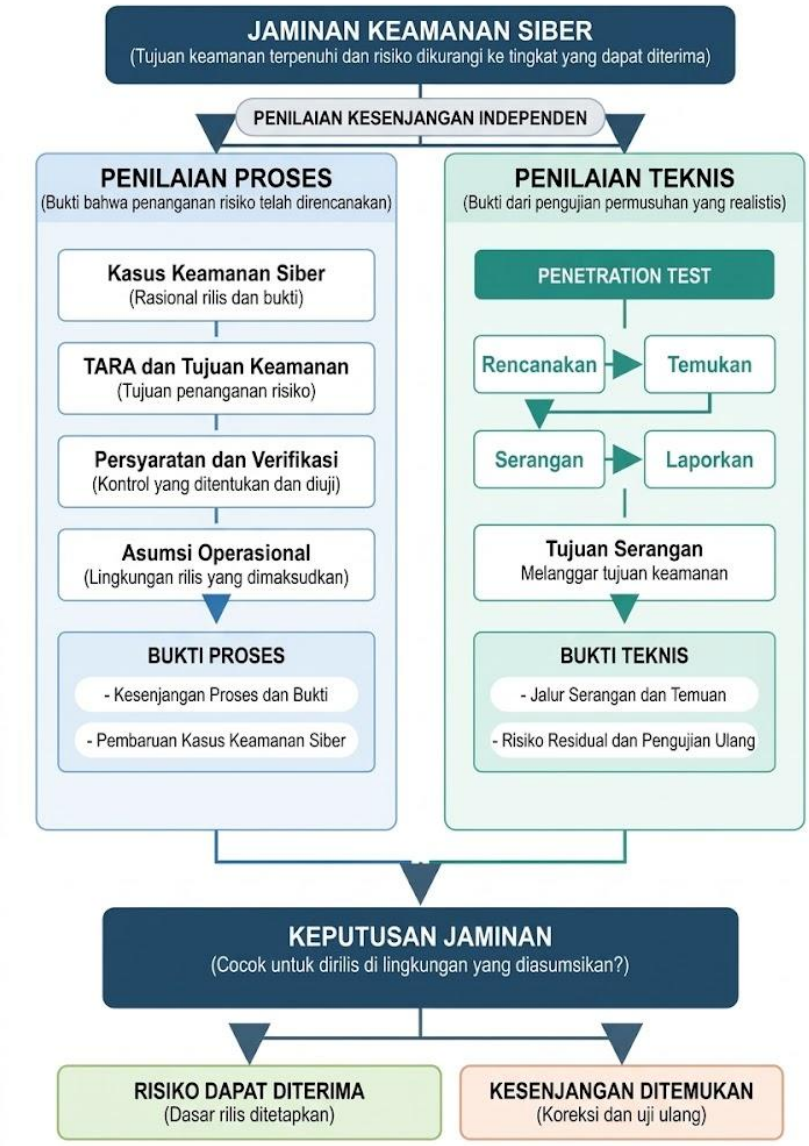
Risiko yang diketahui (*known risks*) adalah risiko yang didokumentasikan selama fase TARA (*Threat Analysis and Risk Assessment*). Risiko yang tidak diketahui (*unknown risks*) adalah risiko yang tersembunyi dari perancang sistem, mungkin karena proses TARA yang tidak lengkap. Risiko residual (*residual risks*) adalah risiko yang muncul akibat langkah mitigasi yang tidak sepenuhnya efektif dalam menangani risiko awal. Sebagai contoh, sistem yang menerapkan pemeriksaan *secure boot* untuk mendeteksi manipulasi pada *binary* perangkat lunak di media penyimpanan masih memiliki risiko residual, di mana pemeriksaan pengukuran *boot* itu sendiri dapat menjadi sasaran serangan *glitching* yang mampu memintas proses *secure boot* tersebut. Berbagai jenis risiko ini diilustrasikan dalam Gambar 1.12.



Gambar 1.12: Jenis-jenis risiko keamanan siber berdasarkan status mitigasinya

Di sisi lain, risiko yang tidak dimitigasi adalah risiko yang tidak dapat ditangani melalui mitigasi apa pun. Jika risiko tersebut diketahui, perancang sistem mungkin menganggap tingkat risikonya terlalu rendah atau memilih untuk menerima risiko tersebut karena alasan teknis maupun performa. Sebagai contoh, perancang sistem mungkin memutuskan untuk menerima kerentanan *speculative execution* (eksekusi spekulatif) pada CPU karena penerapan mitigasi akan menyebabkan penurunan performa yang signifikan. Baik diketahui maupun tidak, keberadaan risiko sisa (*residual risks*) dan risiko yang tidak dimitigasi menciptakan peluang untuk dieksploitasi oleh pihak lawan yang gigih, yang memiliki keahlian serta sumber daya yang memadai. Tugas para profesional di bidang keamanan ofensif (*offensive security*) adalah mengungkap bagaimana risiko-risiko ini dapat dieksploitasi sebelum ditemukan oleh penyerang sungguhan.

Jaminan keamanan siber (*cybersecurity assurance*) merupakan keyakinan bahwa suatu sistem telah memenuhi tujuan keamanan yang ditetapkan dan berhasil mengurangi risiko keamanan siber hingga ke tingkat yang dapat diterima. Saat mengevaluasi tingkat jaminan keamanan siber pada suatu produk otomotif, terdapat dua pendekatan yang dapat digunakan: pendekatan berbasis proses dan pendekatan berbasis teknis.



Gambar 1.3: Jaminan keamanan siber melalui penilaian proses dan teknis

Pendekatan-pendekatan ini, meskipun berbeda dalam cakupan dan metodologinya, memiliki prinsip dasar yang sama, yaitu keduanya dilaksanakan oleh tim independen yang bertujuan untuk menemukan kesenjangan teknis maupun kesenjangan yang berkaitan dengan proses. Pendekatan pertama berfokus pada penilaian produk berdasarkan sistem manajemen keamanan siber untuk menentukan apakah aktivitas keamanan siber yang telah dilakukan sudah memadai dalam mengurangi risiko hingga ke tingkat yang wajar. Dalam konteks ISO/SAE 21434, penilaian ini akan memeriksa kasus keamanan siber beserta bukti-bukti yang disertakan untuk memastikan bahwa produk tersebut layak dirilis, dengan mempertimbangkan lingkungan operasional yang diasumsikan.

Pendekatan kedua melibatkan pelaksanaan pengujian penetrasi untuk memverifikasi bahwa risiko keamanan siber telah ditangani dengan cukup, melalui paparan sistem terhadap kondisi-kondisi berbahaya yang mungkin ditemui di dunia nyata. Hasil pengujian ini memberikan wawasan kepada tim teknis mengenai celah keamanan pada produk mereka, baik

sebelum produk tersebut diluncurkan maupun sebelum risiko tersebut terekspos ke publik (jika produk sudah telanjur didistribusikan). Setelah memahami cakupan *penetration testing* serta keterkaitannya dengan ISO/SAE 21434, kita akan membahas berbagai jenis metode *penetration testing*.

Jenis-Jenis Pengujian Penetrasi

Pengujian penetrasi dapat bervariasi dalam cakupannya, bergantung pada tingkat informasi yang tersedia bagi penguji. Pada satu sisi spektrum terdapat pengujian *black box*, di mana penguji hanya diberikan informasi yang sangat terbatas mengenai target; sementara di sisi lain spektrum terdapat pengujian *white box*, di mana penguji diberikan informasi yang hampir tidak terbatas mengenai target tersebut. Setiap jenis pengujian memiliki kelebihan dan kekurangan, yang akan kita bahas selanjutnya.

Pengujian Penetrasi White Box

Pengujian penetrasi white box adalah metodologi penilaian keamanan komprehensif di mana penguji memiliki pengetahuan lengkap mengenai struktur internal sistem, termasuk kode sumber (*source code*), arsitektur, dan dokumen desain. Pendekatan ini memungkinkan pemeriksaan mendalam terhadap potensi kerentanan dengan menganalisis kode untuk mencari kelemahan keamanan, meninjau konfigurasi sistem, serta memahami interaksi antarberbagai komponen sistem.

Proses ini biasanya mencakup peninjauan kode secara mendetail, analisis statis dan dinamis, serta simulasi skenario serangan dengan memanfaatkan seluruh informasi yang tersedia. Tingkat akses seperti ini memungkinkan penguji untuk menemukan kerentanan yang mungkin terlewatkan dalam jenis pengujian penetrasi lainnya, mengingat diperlukannya pemahaman teknis yang mendalam. Terlepas dari efektivitasnya, pengujian *white box* menghadirkan beberapa tantangan, termasuk waktu dan sumber daya yang signifikan yang dibutuhkan, serta kebutuhan untuk menangani informasi rahasia yang sensitif secara bertanggung jawab.

Selain itu, ketika ditugaskan oleh OEM, pengujian *white box* mungkin tidak praktis jika pemasok tidak bersedia berbagi kode atau dokumentasi desain. Meskipun dapat dimengerti bahwa organisasi mungkin ragu untuk mempekerjakan penguji pihak ketiga untuk pengujian penetrasi *white box* karena kekhawatiran tentang informasi rahasia, hal ini mungkin tidak dapat dihindari ketika menargetkan komponen bernilai tinggi yang memerlukan pengetahuan keamanan khusus yang tidak tersedia secara internal. Sebagai contoh, sebuah perusahaan mungkin melibatkan pihak ketiga untuk melakukan analisis keamanan mendalam terhadap kode bootROM sebelum kode tersebut diterapkan ke dalam silikon, atau untuk menganalisis IP yang aman (*secure IP*) sebelum melisensikannya kepada berbagai vendor cip.

Informasi yang umumnya disediakan untuk pengujian *white box* atau *gray box* mencakup berkas basis data CAN, topologi CAN dan diagram pengabelan, skema ECU, diagram blok perangkat keras/perangkat lunak, catatan aplikasi keamanan, informasi pembongkaran kendaraan untuk menemukan akses ke jaringan yang tepat, TARA (*Threat Analysis and Risk Assessment*), citra *firmware* atau berkas biner, serta kode sumber (*source code*).

Uji Penetrasi Black Box

Uji penetrasi *black box* adalah metodologi penilaian keamanan di mana penguji mengevaluasi sistem elektronik kendaraan tanpa memiliki pengetahuan awal mengenai arsitektur internal, kode, atau protokol khusus (*proprietary protocols*) yang digunakan. Dalam artian tertentu, jenis uji penetrasi ini menyimulasikan perspektif penyerang eksternal yang memulai aksinya dengan pengetahuan yang sangat terbatas mengenai target, sebelum berupaya menemukan kerentanan yang dapat dieksploitasi.

Perbedaan utama antara keduanya adalah bahwa penyerang di dunia nyata mungkin bersedia meluangkan waktu dan sumber daya yang jauh lebih banyak dibandingkan tim penguji profesional yang terikat oleh batasan waktu dalam kontrak. Pendekatan ini mencakup tahapan seperti pengintaian (*reconnaissance*) untuk mengidentifikasi antarmuka eksternal (seperti titik akses bus CAN atau port OBD-II), pemindaian kerentanan melalui analisis protokol dan ekstraksi *firmware*, upaya eksploitasi seperti *fuzz testing* dan serangan nirkabel, serta analisis pasca-eksploitasi untuk peningkatan hak akses (*privilege escalation*) dan mekanisme persistensi.

Biasanya, uji *black box* menghadirkan tantangan unik, seperti aksesibilitas perangkat keras, kurangnya pemahaman mengenai protokol khusus, serta risiko keselamatan yang tidak diketahui dalam operasi pengujian tertentu. Oleh karena itu, pengujian sebaiknya dilakukan di lingkungan yang terkendali, seperti sistem *Hardware-in-the-Loop* (HIL), sembari berkolaborasi dengan produsen untuk memaksimalkan manfaat dari pengujian tersebut. Baik produsen peralatan asli (OEM) maupun pemasok sering kali menggunakan uji *black box* ketika waktu sebelum peluncuran produk sangat terbatas dan staf internal tidak dapat memberikan dukungan ekstensif kepada tim penguji. Dengan demikian, mereka mengandalkan uji *black box* sebagai langkah terakhir sebelum meluncurkan produk guna memastikan keamanan produk tersebut.

Uji Penetrasi Gray Box

Dalam uji penetrasi *gray box*, penguji memiliki pengetahuan parsial mengenai struktur internal sistem seperti diagram arsitektur kelistrikan/elektronik (E/E) kendaraan, spesifikasi protokol, atau potongan kode. Hal ini memungkinkan penguji untuk berfokus pada area-area krusial, sembari tetap menyimulasikan skenario serangan yang realistis dari sudut pandang pihak luar. Penguji juga dapat diberikan akses ke peralatan dan perangkat khusus, misalnya rangkaian kabel (*wiring harness*) khusus yang memungkinkan akses ke jaringan internal kendaraan. Metode ini sangat efektif karena mampu menyeimbangkan antara ketelitian dan efisiensi; pendekatan ini memfasilitasi identifikasi kerentanan yang mungkin terlewatkan dalam uji *black box* akibat kompleksitas serta sifat sistem otomotif yang bersifat eksklusif (*proprietary*) tanpa harus menghabiskan sumber daya yang sangat besar sebagaimana yang diperlukan dalam penilaian *white box* secara menyeluruh.

Proses ini mencakup analisis dokumentasi yang tersedia, pemetaan antarmuka utama, pelaksanaan pemodelan ancaman untuk mengidentifikasi area berisiko tinggi, serta melakukan pengujian dan eksploitasi terarah berdasarkan pengetahuan parsial. Selain itu, tantangan seperti keterbatasan akses informasi dan kompleksitas sistem diatasi dengan

berkolaborasi bersama para pemangku kepentingan serta menangani informasi sensitif secara etis. Dengan demikian, *gray box testing* (pengujian kotak abu-abu) memungkinkan OEM dan pemasok untuk menemukan lebih banyak kerentanan, sekaligus menyimulasikan penyerang di dunia nyata baik itu ancaman dari pihak internal maupun penyerang yang sangat canggih yang meluangkan banyak waktu untuk memperoleh pengetahuan yang memang disediakan dalam lingkup *gray box testing* tersebut. Meskipun waktu pengujian mungkin terbatas, dengan pengetahuan yang memadai mengenai target, hasil yang diperoleh dapat menjadi lebih bermakna dibandingkan dengan *black box testing* (pengujian kotak hitam), tanpa membebani staf OEM atau pemasok secara berlebihan dalam mendukung tim *penetration testing*. Pada tahap ini, Anda telah membangun landasan yang kuat mengenai apa itu *penetration testing* dan langkah-langkah yang terlibat di dalamnya. Kini, saatnya untuk menelusuri fase-fase dalam *penetration testing*.

Fase-Fase Penetration Testing

Standar keamanan menyediakan berbagai kerangka kerja untuk melaksanakan *penetration testing*. Meskipun terdapat perbedaan dalam pendekatan spesifiknya, standar-standar tersebut selalu mencakup empat fase utama sebagaimana ditunjukkan pada Gambar 1.14:

- Perencanaan dan penentuan lingkup (*scoping*) sistem target
- Penemuan (*discovery*) melalui pengumpulan informasi dan identifikasi kerentanan
- Penyusunan serangan dan eksploitasi
- Pelaporan dan komunikasi

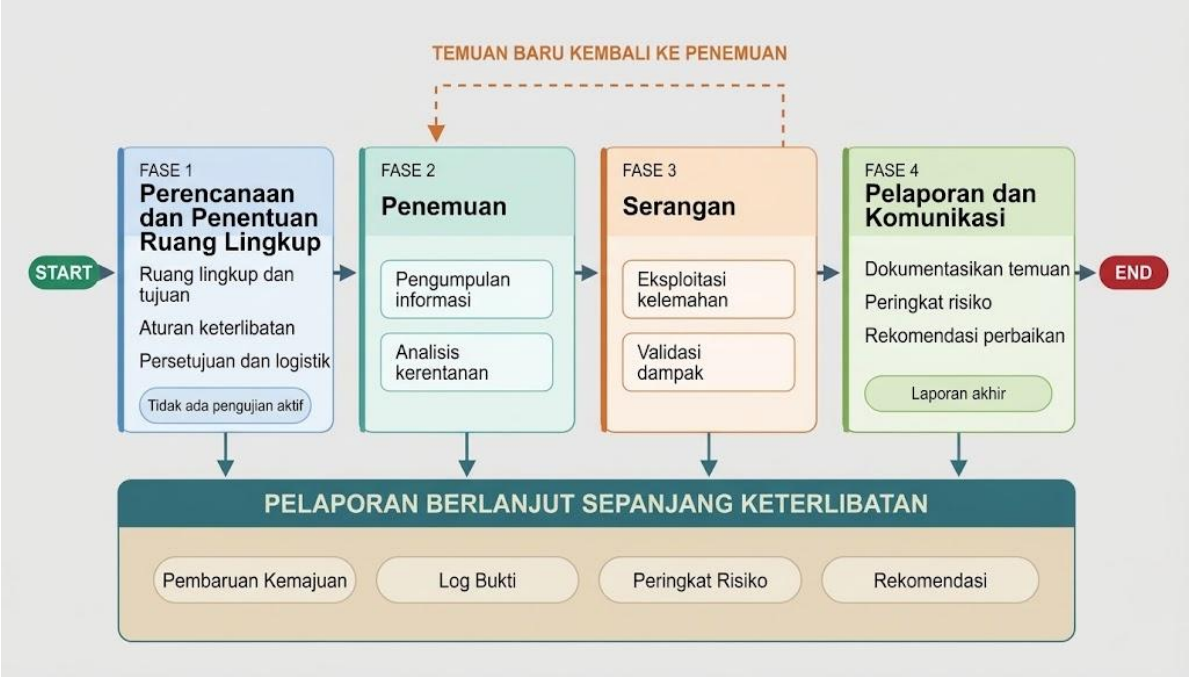
Kita akan membahas masing-masing fase tersebut sekarang.

Fase Perencanaan

Fase ini mencakup persiapan dasar untuk keberhasilan pengujian penetrasi dengan menetapkan aturan, merampungkan persetujuan manajemen, dan menentukan tujuan pengujian. Fase perencanaan tidak melibatkan kegiatan pengujian yang sesungguhnya. Sebaliknya, fase ini berfokus pada pengembangan strategi yang komprehensif, termasuk menentukan ruang lingkup, tujuan, batasan, dan aturan keterlibatan (*rules of engagement*/ROE) untuk pengujian tersebut. Logistik terperinci, seperti penjadwalan, sumber daya, dan koordinasi dengan pihak-pihak terkait, juga ditetapkan selama fase ini.

Fase Penemuan (Discovery)

Fase penemuan mencakup dua komponen utama: pengumpulan informasi serta pemindaian dan enumerasi.



Gambar 1.14: Empat fase utama dalam pengujian penetrasi (*penetration testing*)

Pengumpulan Informasi

Ini merupakan bagian awal dari fase penemuan yang melibatkan pengumpulan data mengenai arsitektur E/E (elektrik/elektronik) kendaraan, termasuk antarmuka kendaraan eksternal, layanan *backend*, perangkat lingkungan operasional, protokol jaringan di dalam kendaraan, ECU (*Electronic Control Unit*), serta fungsionalitas ECU tersebut. Bergantung pada jenis pengujian penetrasi yang dilakukan seperti pengujian *white box, gray box*, atau *black box* proses pengumpulan informasi dapat memakan porsi yang cukup besar dari keseluruhan rangkaian pengujian.

Pemindaian dan Enumerasi

Tahap ini merupakan kelanjutan dari pengumpulan informasi dengan mengidentifikasi komponen yang dapat dijangkau, layanan yang terbuka, perilaku komunikasi, versi perangkat lunak atau *firmware*, fungsi diagnostik, pesan jaringan, serta karakteristik lain yang dapat diamati dari sistem target. Pemindaian dapat bersifat pasif maupun aktif. Pemindaian pasif melibatkan pemantauan komunikasi yang sudah ada tanpa mengirimkan lalu lintas uji seperti mengamati interaksi pada CAN, Ethernet, jaringan nirkabel, atau sistem *backend* untuk menyimpulkan perilaku sistem. Pemindaian aktif melibatkan pengiriman pesan atau permintaan, lalu menganalisis respons dari modul-modul terkait; contohnya termasuk melakukan pemeriksaan pada layanan diagnostik, API *backend*, atau ECU yang terhubung ke jaringan.

Alat pemindaian kerentanan otomatis sangat berguna pada tahap ini, terutama untuk mengidentifikasi kelemahan yang sudah diketahui, kesalahan konfigurasi, layanan yang terbuka, versi perangkat lunak yang rentan, atau masalah yang tercatat dalam sumber publik seperti NVD. Namun, alat-alat ini tidak dapat menemukan kerentanan yang belum terdokumentasi atau belum diketahui sebelumnya. Penemuan kerentanan yang tidak

diketahui sangat bergantung pada pengalaman, kreativitas, dan kemampuan penguji dalam menafsirkan hasil, mendeteksi perilaku yang tidak lazim, menyusun hipotesis, serta melakukan investigasi lebih mendalam pada area tertentu.

Dalam pengujian penetrasi otomotif, keterbatasan waktu sering kali membuat fokus utama tetap tertuju pada kerentanan yang sudah diketahui dan permukaan serangan (*attack surface*) yang terlihat jelas. Investigasi lebih mendalam terhadap kerentanan yang belum diketahui menjadi lebih penting ketika masalah yang ada terbatas, alat pemindaian standar tidak dapat diterapkan secara efektif, atau jika lingkup pengujian secara khusus menuntut riset kerentanan tingkat lanjut.

Tahap Serangan

Tahap serangan melibatkan eksploitasi kerentanan atau perilaku mencurigakan yang ditemukan selama tahap penemuan, guna menentukan dampaknya serta memverifikasi apakah kerentanan tersebut dapat dieksploitasi. Hal ini dapat mencakup upaya memperoleh akses tanpa izin, meningkatkan hak akses (*privilege escalation*), memintas kontrol keamanan, mengakses informasi sensitif, atau mencoba melakukan pengendalian kendaraan secara berbahaya semuanya dilakukan sesuai dengan aturan keterlibatan (*rules of engagement*) yang telah disepakati.

Dalam praktiknya, tahap ini bersifat sangat iteratif. Penguji jarang sekali melakukan penemuan hanya sekali lalu melancarkan serangan satu kali saja. Sebaliknya, proses eksploitasi sering kali mengungkap informasi baru yang mengharuskan penguji kembali ke tahap penemuan. Keberhasilan dalam memintas fungsi diagnostik mungkin akan mengungkap fungsi ECU tambahan, sementara akses ke satu layanan *backend* bisa menyingkap API baru. Demikian pula, rekayasa balik (*reverse engineering*) terhadap satu citra *firmware* mungkin akan mengungkap komponen lain yang layak untuk diuji.

Proses ini sering kali berputar antara tahap penemuan, serangan, penemuan kembali, dan serangan yang lebih terarah, hingga penguji merasa telah cukup mengeksplorasi jalur-jalur serangan yang dapat dijangkau. Masalah-masalah umum yang diperiksa selama fase ini mencakup kelemahan perangkat keras, kesalahan konfigurasi jaringan, *buffer overflow*, validasi input yang tidak memadai, *race condition*, diagnostik yang tidak aman, autentikasi yang lemah, serta cacat pada perlindungan prosesor atau memori.

Tahap Pelaporan

Tahap pelaporan terintegrasi dengan tiga tahap lainnya dan mencakup pendokumentasian temuan serta kemajuan secara berkelanjutan. Laporan biasanya memuat catatan aktivitas, berbagai tangkapan layar (*screenshot*), kerentanan yang ditemukan, peringkat risiko, dan rekomendasi untuk memitigasi kelemahan yang teridentifikasi. Penguji penetrasi (*penetration tester*) juga dapat membantu dalam analisis akar masalah (*root cause analysis*), tidak hanya untuk menentukan eksploitasi yang digunakan, tetapi juga untuk menyarankan langkah-langkah mitigasi. Sebuah laporan komprehensif akhir disusun pada akhir proses pengujian, yang merinci semua temuan, metodologi, serta saran untuk perbaikan.

Sejauh ini, kita telah mempelajari jenis-jenis pengujian penetrasi dan tahapan pelaksanaannya yang menggambarkan bagaimana sebuah pengujian penetrasi

diorganisasikan. Kualitas dan relevansi temuan yang dihasilkan juga bergantung pada siapa yang melakukan pengujian tersebut serta kemampuan lawan (*adversary*) seperti apa yang diharapkan untuk direpresentasikan oleh penguji hal ini akan kita bahas pada bagian selanjutnya.

Memilih Penguji Penetrasi

Meskipun kampanye pengujian penetrasi bervariasi dalam gaya dan cakupannya, faktor penentu yang penting saat memilih penguji penetrasi adalah profil penyerang yang menjadi fokus pengujian. Sebagai contoh, jika suatu sistem dirancang agar tangguh terhadap ancaman persisten tingkat lanjut (*Advanced Persistent Threats*/APT) yang dilakukan oleh negara tertentu, maka kampanye pengujian penetrasi harus melibatkan profesional keamanan ofensif berpengalaman yang memiliki kemampuan setara dengan anggota kelompok APT. Hal ini memastikan bahwa pengujian tersebut dilakukan secara menyeluruh dan secanggih serangan nyata yang mungkin dilancarkan oleh kelompok APT.

Jika Anda bekerja di organisasi berskala besar, Anda mungkin memiliki tim keamanan ofensif khusus yang berfokus sepenuhnya pada evaluasi keamanan semua produk yang terkait dengan keamanan siber. Anggota tim ini telah menghabiskan karier mereka untuk mengembangkan keahlian teknis yang mendalam dalam mengidentifikasi kerentanan pada sistem tertanam (*embedded systems*). Mereka memahami arsitektur perangkat keras maupun perangkat lunak hingga ke detail terkecil mulai dari cara fuse bit dibaca di tingkat perangkat keras hingga bagaimana kernel sistem operasi mengelola tabel halaman memori (*memory page tables*). Berbekal berbagai perangkat untuk melakukan rekayasa balik (*reverse engineering*) pada binari dan melancarkan serangan intrusif baik di tingkat perangkat keras maupun perangkat lunak, mereka menyediakan lapisan keahlian keamanan yang krusial.

Namun, jika Anda tidak memiliki tim seperti itu, terdapat dua opsi: menyewa tim penguji penetrasi pihak ketiga selama durasi proyek, atau membentuk tim penguji independen internal yang mungkin menjadi salah satu alasan Anda membeli buku ini. Teruslah membaca jika Anda berniat membangun tim semacam itu secara internal. Mengetahui keahlian dan sumber daya yang diperlukan untuk tim pentesting (pengujian penetrasi) merupakan langkah dasar dalam merekrut tim yang tepat atau membangun tim internal. Karena pengujian penetrasi menuntut tingkat spesialisasi yang tinggi, penting untuk memahami kapan harus mencari bantuan dari pihak luar dan kapan sebaiknya berinvestasi pada sumber daya internal. Sebagai contoh, jika Anda adalah vendor semikonduktor yang ingin mengidentifikasi kerentanan keamanan dalam desain perangkat keras sebelum tahap tapeout (produksi) cip, sering kali disarankan untuk melibatkan tim pentest eksternal sejak awal proses desain. Tim eksternal ini harus memiliki rekam jejak dalam menemukan kelemahan nyata serta menyusun laporan terperinci mengenai analisis dan temuan mereka.

Perlu dicatat bahwa keberhasilan sebuah pengujian penetrasi tidak boleh dinilai hanya berdasarkan jumlah kerentanan yang ditemukan. Laporan komprehensif yang menunjukkan cakupan dan kedalaman pengujian meskipun tidak ditemukan kerentanan besar tetap dianggap sebagai hasil positif. Hal ini menunjukkan bahwa siklus hidup rekayasa yang aman (*secure engineering lifecycle*) telah berjalan dengan efektif. Namun, jika tim yang tepat telah

direkrut dan sistem diperiksa secara menyeluruh dalam jangka waktu yang memadai, kemungkinan besar akan ditemukan sesuatu yang signifikan.

Peretas White Hat vs. Black Hat

Meskipun peretas *white hat* dan *black hat* mungkin menggunakan keterampilan teknis yang serupa, keduanya berbeda dalam hal niat, otorisasi, serta cara mereka menangani kerentanan yang ditemukan. Peretas black hat mengeksploitasi kerentanan untuk tujuan jahat, seperti pencurian, gangguan, akses tanpa izin, atau keuntungan pribadi. Dalam sistem otomotif tertanam (*embedded automotive systems*), tindakan ini bisa mencakup upaya mengakses data kendaraan yang sensitif, memanipulasi fungsi kendaraan, mencegat komunikasi, mengeksploitasi kelemahan perangkat lunak, atau menyisipkan kode berbahaya. Aktivitas mereka menunjukkan mengapa langkah-langkah keamanan siber yang tangguh, penilaian keamanan berkala, dan proses penanganan kerentanan yang jelas sangatlah diperlukan.

Sebaliknya, peretas white hat menggunakan pengetahuan keamanan mereka untuk mengidentifikasi kelemahan agar dapat diperbaiki sebelum pihak jahat mengeksploitasinya. Kelompok ini mencakup tim keamanan internal, penguji penetrasi pihak ketiga profesional yang disewa oleh organisasi, peserta program *bug bounty*, serta peneliti keamanan independen yang mengikuti praktik pengungkapan yang bertanggung jawab (*responsible disclosure*). Dalam penugasan pengujian penetrasi formal, aktivitas white hat secara eksplisit diizinkan dan diatur oleh cakupan yang jelas, aturan keterlibatan (*rules of engagement*), serta proses pelaporan yang ditetapkan. Peneliti independen mungkin tidak selalu memiliki izin sebelumnya, namun mereka tetap dapat bertindak dengan etika yang baik (*white hat*) dengan cara menghindari kerugian, membatasi pengujian sebatas yang diperlukan, melindungi informasi sensitif, serta mengungkapkan temuan mereka secara bertanggung jawab melalui saluran yang tepat.

Ketika pengujian penetrasi (*penetration test*) atau kegiatan penelitian yang bertanggung jawab mengungkap kerentanan serius, temuan tersebut harus ditangani dengan hati-hati. Tim penguji atau peneliti hanya boleh membagikan detail temuan kepada pihak-pihak tepercaya yang telah ditentukan di dalam organisasi terkait, serta harus menghindari pengungkapan publik yang tidak disengaja. Jika sebuah tim penguji penetrasi bekerja dengan beberapa klien, mereka wajib menjaga agar informasi setiap klien tetap terpisah guna mencegah kontaminasi informasi, kebocoran data, atau kekeliruan antar-penugasan.

Organisasi dapat melibatkan komunitas *white hat* (peretas etis) yang lebih luas melalui program bug bounty, program pengungkapan kerentanan terkoordinasi, ajang seperti Pwn2Own, serta hackathon privat. Program bug bounty menawarkan imbalan finansial atau pengakuan kepada peneliti yang melaporkan celah keamanan dalam perangkat lunak kendaraan, perangkat keras, sistem *backend*, atau aplikasi seluler. Ajang seperti Pwn2Own dan kompetisi peretasan privat menyediakan lingkungan terkendali di mana para peserta berupaya mengeksploitasi target otomotif tertentu sesuai dengan aturan yang telah disepakati. Eksploitasi yang berhasil sering kali berlanjut pada kolaborasi dengan vendor untuk memvalidasi, menentukan prioritas, dan memperbaiki masalah yang ditemukan.

Inisiatif-inisiatif ini membantu mengungkap masalah keamanan kritis sekaligus membangun kepercayaan antara produsen dan komunitas peneliti keamanan. Tesla merupakan contoh produsen otomotif ternama yang secara rutin memanfaatkan program bug bounty dan ajang peretasan publik untuk melibatkan para peneliti *white hat* serta memperkuat keamanan siber kendaraan.

Tim Merah vs. Tim Biru

Sebagai alternatif untuk menyewa penguji penetrasi pihak ketiga, organisasi dapat membentuk tim penguji internal yang sering disebut sebagai *red team*. *Red team* beroperasi secara independen di dalam organisasi, biasanya ditempatkan dalam divisi yang terpisah dari tim keamanan defensif yang dikenal sebagai *blue team*. Pemisahan ini sangat penting untuk menjaga objektivitas dan memastikan bahwa penilaian keamanan tetap tidak bias. Peran *red team* adalah mengambil perspektif calon penyerang, menyimulasikan ancaman nyata dengan mengidentifikasi serta mengeksploitasi kerentanan dalam produk otomotif beserta sistem pendukungnya. Hal ini mencakup pengujian terhadap kerentanan yang sudah diketahui maupun upaya untuk menemukan kelemahan yang belum diketahui (*zero-day*). *Red team* memiliki keahlian tinggi dalam teknik keamanan ofensif, menggunakan perangkat dan metodologi yang sering kali serupa dengan yang digunakan oleh pihak-pihak berbahaya. Misi mereka adalah menemukan celah keamanan sebelum penyerang dapat memanfaatkannya, sehingga membantu organisasi dalam memperkuat pertahanan secara proaktif.

Sebaliknya, *blue team* bertanggung jawab untuk mempertahankan sistem. Hal ini mencakup pemodelan ancaman, penetapan konsep keamanan siber, pelaksanaan analisis kerentanan baik selama maupun setelah pengembangan, serta pemeliharaan postur keamanan secara menyeluruh. Sementara tim merah (*red team*) berupaya untuk menyusup ke dalam sistem, tim biru (*blue team*) bekerja untuk mencegah, mendeteksi, dan memitigasi penyusupan tersebut.

Kolaborasi antara tim merah dan tim biru sangat penting untuk meningkatkan keamanan produk. Temuan tim merah baik berupa kerentanan, kesalahan konfigurasi, maupun kelemahan dalam pertahanan sistem segera disampaikan kepada tim biru melalui saluran pelaporan formal. Proses ini memastikan bahwa kerentanan kritis dapat ditangani tepat waktu. Bergantung pada struktur organisasi, hal ini dapat melibatkan sesi evaluasi (*debrief*) rutin atau sesi serah terima, di mana kedua tim mendiskusikan temuan, metode yang digunakan untuk mengeksploitasi kerentanan, serta rekomendasi untuk perbaikan.

Di banyak organisasi, kombinasi antara latihan tim merah dan tim biru menghasilkan apa yang disebut sebagai purple teaming, di mana kedua tim berkolaborasi secara erat. Hal ini berujung pada peningkatan berkelanjutan terhadap langkah-langkah keamanan, karena para ahli di bidang ofensif dan defensif bekerja sama untuk mengidentifikasi celah serta memastikan celah tersebut ditutup seefisien mungkin. Dengan menumbuhkan budaya pengujian keamanan internal melalui tim merah, organisasi dapat memperkuat pertahanan mereka, menyelaraskan diri dengan praktik terbaik industri, serta mengurangi ketergantungan pada pihak eksternal untuk menjamin keamanan.

1.7 KONSEP DAN METODOLOGI KEAMANAN OFENSIF OTOMOTIF

Dalam bab ini, kita telah melihat mengapa keamanan ofensif sangat penting untuk memahami cara melindungi kendaraan modern dari ancaman siber yang semakin canggih. Jika keamanan defensif berfokus pada pembangunan dan pemeliharaan pertahanan yang kuat, keamanan ofensif lebih menekankan pada pemahaman serta eksploitasi kerentanan potensial agar dapat selangkah lebih maju dari pihak lawan. Kita telah membahas bagaimana pendekatan sistematis terhadap keamanan ofensif memungkinkan para insinyur untuk mengidentifikasi kelemahan dalam arsitektur kelistrikan/elektronik (E/E), protokol, dan lingkungan operasional kendaraan, sebelum pihak yang berniat jahat dapat memanfaatkannya.

Bab ini menyoroti kompleksitas arsitektur berbasis domain maupun berbasis komputer pusat, yang dipadukan dengan luasnya permukaan serangan pada kendaraan terhubung (*connected vehicles*). Hal ini menuntut para profesional keamanan siber untuk terus mengembangkan strategi pertahanan mereka. Kita juga telah mencapai kesepakatan mengenai tingkat penyerang yang perlu diantisipasi mulai dari peretas amatir (*script kiddies*) dan kelompok kriminal, hingga aktor yang didukung oleh negara guna menyelaraskan prioritas keamanan di seluruh organisasi. Pentingnya penyusunan profil penyerang yang komprehensif juga dibahas, yang berfungsi sebagai landasan dalam membangun pertahanan yang tangguh serta menetapkan tujuan yang realistis bagi upaya keamanan ofensif.

Selain itu, kita telah mengulas berbagai metodologi keamanan ofensif yang memberikan wawasan krusial mengenai bagaimana penyerang dapat membobol keamanan kendaraan. Melalui pengujian penetrasi *black box, gray box*, maupun *white box*, metode-metode ini mampu mengungkap kerentanan di setiap tahapan rantai serangan mulai dari akses awal, peningkatan hak akses (*privilege escalation*), hingga eksekusi tujuan jahat. Kami berharap, dengan mengintegrasikan konsep keamanan ofensif ke dalam strategi defensif, organisasi dapat mengantisipasi ancaman dengan lebih baik, meningkatkan keamanan kendaraan, serta memastikan sistem mereka tetap tangguh terhadap serangan siber sepanjang siklus hidupnya.

Sepanjang bab ini, kita telah meletakkan dasar bagi banyak konsep yang akan kita perlukan di sepanjang buku ini. Meskipun kita telah memperkenalkan berbagai tahapan dalam pengujian penetrasi (*penetration testing*), kita belum membahas secara rinci bagaimana cara melaksanakan tahapan-tahapan tersebut. Hal inilah yang akan menjadi fokus pada bab berikutnya.

BAB 2
TAHAPAN PENGUJIAN PENETRASI

Seiring berjalannya waktu, pengujian penetrasi (*penetration testing*) telah berkembang menjadi proses sistematis untuk mengidentifikasi komponen sistem yang krusial bagi pengujian keamanan, menemukan kerentanan pada komponen-komponen tersebut, menyusun eksploitasi terhadap komponen-komponen itu, serta melaporkan kerentanan tersebut secara bertanggung jawab disertai dengan saran mitigasi kepada para pemangku kepentingan yang terdampak. Pada bab 1, kami telah memperkenalkan kerangka dasar pengujian penetrasi yang dimulai dari tahap perencanaan dan penemuan (*discovery*), dilanjutkan dengan pembuatan serangan, dan diakhiri dengan pelaporan. Dalam bab ini, kita akan menelaah lebih dalam setiap tahapan tersebut dan mengevaluasi penerapannya pada kendaraan yang terhubung (*connected vehicle*) beserta lingkungannya.

Dalam prosesnya, kita akan mempelajari cara menyusun rencana komprehensif yang mencakup ruang lingkup pengujian penetrasi serta aturan keterlibatan (*rules of engagement*). Selanjutnya, kita akan mengeksplorasi tahap penemuan melalui pendekatan pengintaian (*reconnaissance*) pasif maupun aktif. Kemudian, kita akan membahas beberapa teknik serangan untuk menjalankan eksploitasi secara berulang hingga tujuan serangan tercapai.

Terakhir, kita akan membahas aktivitas pasca-eksploitasi, termasuk pelaporan dan remediasi. Kami menyadari bahwa banyak topik yang diperkenalkan dalam buku ini sebenarnya dapat dibahas secara mendalam dalam bab tersendiri. Oleh karena itu, tujuan kami dalam bab ini adalah memperkenalkan konsep-konsep yang menjadi prasyarat untuk membangun rantai serangan (*attack chains*) praktis, yang akan menjadi fokus utama pada bagian kedua buku ini.

Dalam bab ini, kita akan membahas topik-topik utama berikut:

- Perencanaan pengujian penetrasi
- Penemuan kelemahan keamanan
- Tahap serangan
- Pelaporan kerentanan

Kami berharap, setelah menyelesaikan bab ini, Anda akan siap untuk melaksanakan seluruh tahapan dalam siklus hidup pengujian penetrasi, dimulai dari tahap perencanaan.

2.1 PERENCANAAN PENGUJIAN KEAMANAN OTOMOTIF

Saat mengikuti proses rekayasa keamanan siber yang sistematis, seperti ISO/SAE 21434, kita diwajibkan untuk merencanakan pengujian keamanan independen selama fase perencanaan keamanan siber. Fase perencanaan menetapkan arah strategis untuk pengujian penetrasi dengan menguraikan lingkup evaluasi dan menentukan bagaimana pelaksanaan pengujian tersebut akan dilakukan. Perencanaan yang cermat memastikan keselarasan antara penguji dan para pemangku kepentingan produk, aspek yang sangat krusial mengingat

kompleksitas ketergantungan dalam sistem tertanam (*embedded system*) otomotif, antarmuka komunikasinya, serta sistem pendukung eksternal lainnya. Namun, menentukan waktu yang tepat untuk melakukan pengujian penetrasi bisa menjadi tantangan tersendiri. Idealnya, pengujian dijadwalkan menjelang akhir pengembangan, saat tidak ada lagi perubahan perangkat lunak atau desain yang diperkirakan akan memengaruhi kinerja keamanan sistem yang diuji.

Pendekatan ini mengharuskan adanya periode jeda (*buffer*) antara tanggal pembekuan desain (*design freeze*) dan tanggal peluncuran produk akhir, guna memberikan waktu untuk pengujian serta penerapan perbaikan yang diperlukan. Dengan munculnya kendaraan berbasis perangkat lunak (*software-defined vehicles*), perangkat lunak otomotif sering kali dirilis secara berkelanjutan, sehingga sulit untuk menentukan waktu yang paling tepat bagi pengujian penetrasi. Meskipun model rilis berkelanjutan ini bermanfaat untuk menangani kerentanan keamanan, pembaruan perangkat lunak yang terus-menerus menghadirkan tantangan unik bagi sistem yang krusial terhadap keselamatan (*safety-critical systems*).

Sebagai contoh, kegiatan kualifikasi keselamatan harus dikoordinasikan secara cermat dengan perilisan tambalan keamanan (*security patch*) untuk mencegah munculnya masalah kualitas atau keselamatan secara tidak sengaja saat memperbaiki kerentanan keamanan. Hal ini menuntut kolaborasi erat antara tim keselamatan (*safety*) dan tim keamanan (*security*) guna memastikan bahwa pembaruan yang meningkatkan keamanan tidak mengorbankan integritas keselamatan, dan sebaliknya. Dengan demikian, fase perencanaan tidak hanya terbatas pada penetapan tujuan dan lingkup, tetapi juga mencakup penyepakatan jadwal yang mempertimbangkan berbagai persyaratan produk lainnya, seperti penilaian keselamatan, pengujian kualitas, dan perencanaan rilis secara keseluruhan.

Menentukan Lingkup Pelaksanaan (*Engagement*)

Perencanaan pengujian penetrasi dimulai dengan menentukan lingkup pelaksanaan yaitu proyek pengujian penetrasi spesifik yang dirancang untuk memenuhi kebutuhan produsen atau pemasok otomotif. Tujuan pengujian harus ditetapkan secara eksplisit, dengan memprioritaskan sistem dan komponen yang paling kritis. Aturan Keterlibatan (*Rules of Engagement*/RoE) menguraikan bagaimana tim penguji harus beroperasi, termasuk kapan dan di mana pengujian dapat dilakukan seperti saat kendaraan dalam kondisi diam atau di dalam laboratorium pengujian serta sejauh mana penguji diperbolehkan untuk menyimulasikan serangan. Sebagai contoh, sebuah pengujian penetrasi mungkin menargetkan sistem hiburan (*infotainment*) kendaraan, namun tidak diperbolehkan untuk mengutak-atik sistem pengereman atau kemudi.

Menentukan Ruang Lingkup

Ruang lingkup menguraikan sistem, komponen, dan jaringan mana saja yang akan diuji. Hal ini dapat mencakup berbagai elemen, mulai dari *Electronic Control Unit* (ECU) internal kendaraan, unit berkinerja tinggi seperti modul telematika dan sistem infotainment, hingga protokol komunikasi eksternal yang menghubungkan kendaraan dengan *server back-end* serta sistem pendukung lainnya, seperti klien diagnostik. Penetapan ruang lingkup yang tepat

memastikan semua pihak memahami batasan pengujian dengan jelas, sehingga mencegah perluasan cakupan yang tidak direncanakan (*scope creep*) selama fase pelaksanaan pengujian.

Pertimbangkan skenario di mana Anda telah menyepakati untuk menguji ketahanan *Hardware Security Module* (HSM) tertanam terhadap serangan *side-channel* yang bertujuan mengekstrak bit kunci kriptografis. Setelah hasil pengujian dilaporkan, pemasok komponen mungkin menyadari bahwa mereka juga ingin mengevaluasi perilaku HSM tersebut terhadap serangan injeksi kesalahan (*fault injection*), seperti *electromagnetic glitching*. Jika ruang lingkup terkait serangan *side-channel* tidak didefinisikan dengan jelas selama fase perencanaan, ketidakselarasan ini dapat berujung pada perlunya melakukan pekerjaan tambahan yang tidak direncanakan, atau membuat klien merasa tidak puas.

Menentukan Kondisi Pengujian

Selain menentukan area yang akan diuji, penting juga untuk memperjelas kondisi pelaksanaan pengujian tersebut. Sebagai contoh, berdasarkan ilustrasi di atas, Anda harus memutuskan apakah HSM akan dievaluasi menggunakan *firmware* produksi yang dilengkapi dengan fitur mitigasi *side-channel*, atau menggunakan *firmware* khusus yang mengisolasi perilaku perangkat keras. Keputusan ini dapat sangat memengaruhi tingkat keahlian yang diperlukan serta keseluruhan upaya pengujian. Selain itu, Anda disarankan untuk menggunakan sistem pengembangan (*development*) maupun sistem produksi guna melaksanakan pengujian penetrasi secara menyeluruh.

Perangkat pengembangan mungkin sengaja dibiarkan memiliki akses seperti akses SSH atau port debug yang tidak terkunci untuk memfasilitasi eksplorasi lebih dalam terhadap potensi eksploitasi. Sebaliknya, perangkat produksi dapat digunakan untuk memastikan bahwa antarmuka semacam itu telah dibatasi secara efektif, sesuai dengan prinsip pertahanan berlapis (*defense-in-depth*). Mengingat produk otomotif sering kali digunakan selama satu dekade atau lebih, bijaklah untuk mengasumsikan bahwa lapisan terluar dari strategi pertahanan berlapis apa pun pada akhirnya dapat ditembus. Oleh karena itu, melakukan pengujian pada perangkat pengembangan akan menyimulasikan skenario tersebut, sehingga Anda dapat memverifikasi apakah pertahanan berlapis pada produk tetap mampu bertahan di tengah serangan yang lebih canggih.

Di sisi lain, saat mengevaluasi kendaraan yang terintegrasi, sangat penting untuk menentukan apakah pengujian akan dilakukan saat kendaraan bergerak atau di dalam lingkungan terkendali, seperti di dalam bengkel. Ruang lingkup pengujian harus mempertimbangkan semua batasan teknis yang melindungi fungsi-fungsi krusial bagi keselamatan, guna memastikan fungsi tersebut tetap beroperasi selama pengujian berlangsung. Jika pengujian dilakukan di atas meja kerja (*bench test*) dalam lingkungan laboratorium, langkah-langkah pengamanan harus diterapkan untuk mencegah kerusakan permanen pada sistem. Terakhir, untuk komponen yang krusial bagi keselamatan, sangat disarankan untuk mengandalkan simulasi dan pengujian di meja kerja guna menghindari pemicuan kejadian berbahaya yang tidak diinginkan, seperti terpicunya kantung udara (*airbag*) secara tidak sengaja atau terganggunya fungsi pengereman.

Berbagi Informasi dan Jenis Pengujian

Salah satu hal yang sering terlewatkan selama tahap perencanaan adalah menentukan metode dan cakupan informasi yang akan dibagikan setelah pengujian selesai. Sebagai contoh, jika Anda diminta untuk melakukan uji penetrasi (*penetration test*) pada ECU gerbang pusat (*central gateway ECU*) guna mengidentifikasi kerentanan perangkat lunak yang dapat dieksploitasi untuk mendapatkan akses ke bus *Controller Area Network* (CAN) kendaraan, maka kesepakatan awal mengenai jenis uji penetrasi Kapakah *black box, white box*, atau *gray box* menjadi sangat krusial. Hal ini memastikan alokasi sumber daya yang tepat dan estimasi waktu pengerjaan yang akurat.

Jika tim produk menyetujui pengujian *white box*, mereka harus siap untuk membagikan dokumen teknis terperinci, seperti skema, dokumen desain, berkas basis data CAN, bahkan kode sumber (*source code*). Sebaliknya, jika pengujian dibatasi pada metode *black box*, sistem tersebut harus diberikan kepada tim penguji beserta instruksi dasar untuk pengaturan dan penggunaannya. Sangat disarankan agar informasi yang dibagikan kepada penguji penetrasi disesuaikan dengan jenis pengujian yang diminta. Dengan demikian, tim keamanan ofensif dapat menggunakan waktu berharga mereka untuk menemukan kerentanan, alih-alih menghabiskan waktu mencari informasi yang sebenarnya sudah bisa diakses dengan mudah oleh penyerang dalam skenario dunia nyata.

Selain itu, mengingat beberapa jenis uji penetrasi berisiko membuat sistem tidak dapat berfungsi sama sekali atau rusak total (*bricked*) misalnya setelah melakukan *flashing firmware* tim pengembangan harus memberikan informasi penting mengenai cara menghindari kerusakan pada ECU. Hal ini mencegah terbuangnya waktu berharga untuk melakukan debugging atas masalah yang tidak berkontribusi pada penemuan kerentanan atau eksploit yang nyata. Penting juga untuk dicatat bahwa tim pengembang harus memastikan sistem yang diberikan kepada tim penguji penetrasi (*pentesting*) dapat berfungsi, serta memberikan dukungan yang diperlukan agar sistem target tetap dalam kondisi operasional.

Pertimbangan Hukum dan Dokumentasi

Sangatlah penting untuk menetapkan kerangka kerja yang sah secara hukum sebelum pengujian penetrasi (*penetration testing*) dimulai. Memperoleh persetujuan tertulis biasanya melalui *Master Service Agreement* (MSA), *Statement of Work* (SOW), dan perjanjian kerahasiaan (*Non-Disclosure Agreement* atau NDA) tidak hanya mencegah komplikasi hukum, tetapi juga menetapkan ekspektasi yang jelas mengenai ruang lingkup, jadwal, dan hasil kerja (*deliverables*). Dokumen-dokumen ini merinci tugas-tugas yang harus dilaksanakan, jadwal pengerjaan, hasil kerja (seperti laporan kerentanan yang mendetail), serta ketentuan mengenai penanganan data sensitif, termasuk informasi kepemilikan terkait kendaraan.

Meskipun pengujian dilakukan oleh tim internal dalam organisasi, ruang lingkupnya tetap harus didokumentasikan untuk menghindari ketidakselarasan antara ekspektasi dan hasil. Sebagai contoh, saat bekerja dengan tim pengujian penetrasi internal, perlu ada kesepakatan mengenai tingkat akses yang akan diberikan ke dalam sistem, durasi pelaksanaan pengujian, serta tingkat kedetailan yang diperlukan dalam laporan akhir.

Sebagaimana akan terlihat pada fase penemuan (*discovery phase*), kegiatan pengintaian tertentu berisiko melanggar hak kekayaan intelektual atau mengganggu privasi pengguna. Ketika pengujian melibatkan interaksi langsung dengan sistem kendaraan yang sedang aktif, terdapat risiko memicu malfungsi yang berdampak pada keselamatan. Oleh karena itu, Anda harus menetapkan batasan yang jelas bagi penguji penetrasi, terutama jika terdapat sistem kritis yang berada di luar ruang lingkup pengujian.

Melibatkan penasihat hukum sejak awal sangatlah penting untuk memastikan kontrak mencakup klausul ganti rugi (*indemnity*) dan pembatasan tanggung jawab (*limitation of liability*), khususnya untuk pengujian yang bersifat lebih invasif yang berpotensi mengekspos data kepemilikan atau merusak sistem yang sedang beroperasi. Jika menggunakan penguji eksternal, tim hukum dari kedua belah pihak harus bekerja sama untuk merampungkan perjanjian kerahasiaan (*Non-Disclosure Agreement* atau NDA) dan menangani masalah privasi, termasuk kepatuhan terhadap regulasi perlindungan data seperti GDPR. Bagi produsen otomotif global, aliran data lintas negara harus dikelola agar tetap mematuhi hukum pengendalian ekspor serta berbagai standar privasi internasional yang berlaku.

Selain itu, adanya regulasi baru seperti aturan di Tiongkok yang melarang pengungkapan data pengguna jalan setempat kepada entitas asing menegaskan pentingnya menyelaraskan praktik pengujian penetrasi dengan ketentuan hukum lokal maupun internasional. Jika kerentanan yang ditemukan di kemudian hari menyebabkan kebocoran data saat kendaraan digunakan di lapangan, kontrak harus mengatur tanggung jawab dan kewajiban hukum dari pihak-pihak yang terlibat. Dengan demikian, meskipun mengamankan kendaraan tetap menjadi tujuan utama, landasan hukum yang kuat juga sama pentingnya untuk meminimalkan risiko hukum dan finansial.

Menentukan Prioritas Komponen Untuk Pengujian

Aspek krusial lainnya dalam fase perencanaan adalah memilih komponen mana yang akan diuji. Mengingat pengujian penetrasi sering kali dibatasi oleh waktu dan biaya, penting untuk memprioritaskan komponen serta sistem yang vital bagi operasional dan keselamatan kendaraan. Dalam hal ini, tingkat paparan risiko memegang peranan kunci dalam menetapkan prioritas pengujian. Selama proses Analisis Ancaman dan Penilaian Risiko (*Threat Analysis and Risk Assessment*/TARA), beberapa komponen mungkin telah diidentifikasi sebagai bagian yang menyimpan aset bernilai tinggi atau memiliki arsitektur kompleks yang rentan terhadap kerentanan perangkat lunak maupun perangkat keras. Informasi ini sangat berguna untuk memprioritaskan komponen-komponen tersebut dalam pengujian penetrasi. Selain itu, fitur-fitur baru yang belum pernah dievaluasi secara independen sebelumnya harus diberikan prioritas lebih tinggi, karena fitur-fitur tersebut lebih mungkin mengandung kerentanan yang belum diketahui.

Menentukan Hasil Kerja (*Deliverables*) dan Pelaporan

Komunikasi yang efektif sangatlah penting selama pengujian sistem yang kompleks dan beroperasi secara waktu nyata (*real-time*), seperti kendaraan. Tim penguji penetrasi harus menjaga komunikasi rutin dengan pihak tepercaya di dalam perusahaan manufaktur atau laboratorium pengujian. Pihak ini berperan sebagai penghubung yang memastikan tim penguji

mendapatkan sumber daya yang diperlukan, sekaligus menjaga agar para pemangku kepentingan utama tetap mendapatkan informasi mengenai perkembangan dan temuan pengujian. Tanpa kontak yang tepat, tim penguji mungkin membuang waktu untuk menghubungi pihak yang tidak relevan saat mencari informasi krusial seperti prosedur pemrograman *flash* atau urutan penyalaan (*power-up sequence*) pada unit kontrol elektronik (ECU) yang kompleks yang merupakan prasyarat untuk melanjutkan pengujian.

Sebelum fase perencanaan berakhir, hasil kerja (*deliverables*) dari penugasan ini harus ditetapkan dengan jelas. Biasanya, ini mencakup laporan komprehensif yang merinci kerentanan apa pun yang ditemukan, potensi dampaknya terhadap kendaraan, serta rekomendasi untuk memitigasi risiko tersebut. Format dan isi laporan harus disepakati oleh semua pemangku kepentingan guna memastikan hasil tersebut dapat dipahami dengan baik dan dapat ditindaklanjuti. Selain itu, fase perencanaan juga harus menentukan bagaimana dan kapan hasil pengujian penetrasi (*penetration test*) akan disampaikan. Sebagai contoh, para pemangku kepentingan mungkin meminta agar setiap kerentanan yang ditemukan segera dilaporkan.

Meskipun dapat dipahami dari sudut pandang manajemen risiko, beralih ke fase pelaporan sebelum pengujian penetrasi selesai dapat mengalihkan fokus tim keamanan ofensif dan menghambat kemampuan mereka dalam menyusun rangkaian serangan (*attack chains*) yang komprehensif yang mungkin memanfaatkan kerentanan yang telah ditemukan. Hal ini menegaskan pentingnya mengalokasikan waktu yang cukup untuk menyelesaikan pengujian sebelum pengembangan produk dibekukan (*frozen*), sehingga perbaikan dapat dilakukan jika diperlukan. Kini, setelah fase perencanaan selesai, kita siap melangkah ke salah satu fase terpenting dalam pengujian penetrasi, yaitu fase penemuan (*discovery phase*).

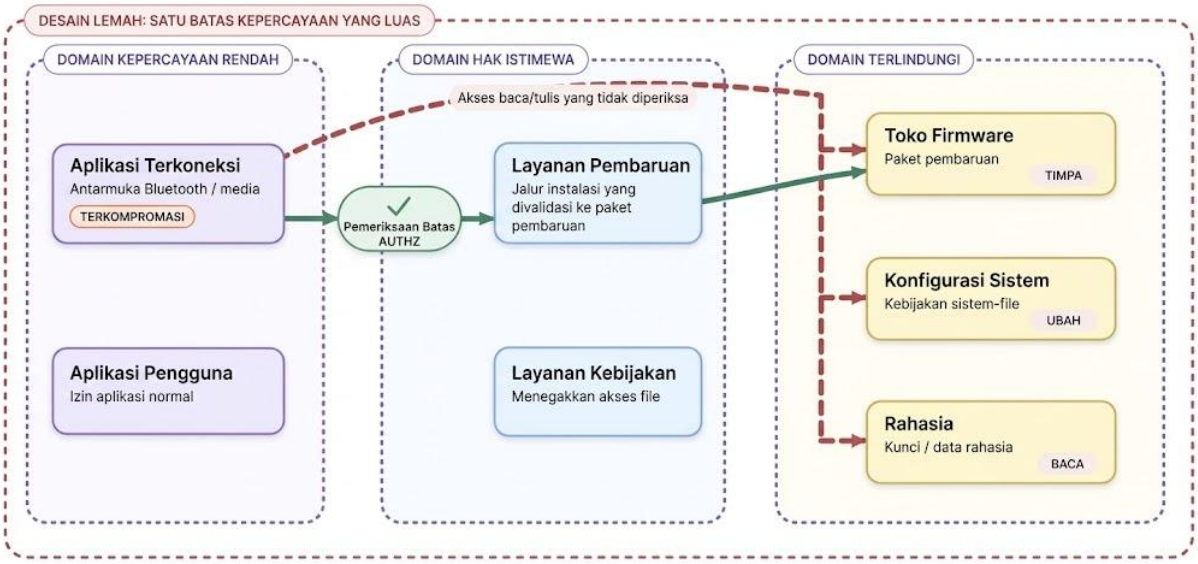
2.2 TEKNIK PENEMUAN DAN IDENTIFIKASI KERENTANAN KEAMANAN OTOMOTIF

Sebelum melancarkan serangan terhadap sistem otomotif apa pun, langkah dasar yang harus dilakukan adalah memahami secara menyeluruh arsitektur sistem, kasus penggunaan (*use cases*), serta interaksinya dengan lingkungan operasional. Sederhananya, kita tidak mungkin menyerang sesuatu secara efektif jika kita tidak memahaminya. Di sinilah peran fase penemuan. Ini merupakan proses sistematis di mana para profesional keamanan ofensif mengumpulkan informasi mengenai sistem target, memetakan permukaan serangan (*attack surface*), serta mengidentifikasi potensi kerentanan dengan memeriksa perangkat lunak, perangkat keras, komponen komunikasi, dan interaksi sistem dengan lingkungannya. Sebelum kita membahas lebih dalam mengenai teknik-teknik untuk menemukan kelemahan keamanan, kita perlu memahami pentingnya memetakan hubungan kepercayaan (*trust relations*) dalam sistem target.

Pemetaan Kepercayaan Dalam Sistem Tertanam (*Embedded Systems*)

Sudah jelas bahwa kepercayaan merupakan landasan bagi setiap sistem yang aman. Pemeriksaan yang cermat terhadap hubungan kepercayaan di dalam kendaraan membantu mengidentifikasi potensi kerentanan yang muncul akibat kepercayaan yang tidak terverifikasi, penerapan batasan yang tidak tepat, atau hubungan kepercayaan yang terlalu luas.

Menemukan kerentanan yang belum diketahui terkadang ibarat mencari jarum dalam tumpukan jerami. Memahami dinamika kepercayaan dapat membantu mempersempit ruang pencarian, sehingga fokus tertuju pada komponen-komponen yang paling mungkin menghasilkan temuan yang signifikan. Batas kepercayaan dapat dipandang sebagai garis pemisah virtual di antara entitas-entitas dalam sistem yang saling tidak memercayai satu sama lain, sebagaimana ditunjukkan pada Gambar 2.1.



Gambar 2.1: Contoh batasan kepercayaan (*trust boundaries*) di dalam sebuah SoC

Batasan kepercayaan yang umum ditemukan adalah batasan antara dua ECU, misalnya antara gerbang pusat (*central gateway*) dan pengontrol mesin (*engine controller*). Tautan jaringan di antara keduanya dianggap sebagai pihak yang tidak tepercaya (*untrusted*). Jika dalam fase penemuan (*discovery phase*) diketahui bahwa ECU-ECU dalam segmen jaringan yang sama menganggap tautan tersebut tepercaya sehingga mengabaikan dukungan untuk autentikasi pesan maka bagian ini menjadi titik yang jelas untuk diteliti terkait kelemahan keamanannya.

Dengan menyimulasikan pelanggaran kepercayaan, melalui serangan *man-in-the-middle* atau *replay attack*, kita dapat mengungkap kerentanan yang muncul akibat penerapan kebijakan kepercayaan yang tidak tepat. Tentu saja, sistem yang dirancang dengan batasan kepercayaan yang luas akan memberikan lebih banyak peluang untuk eksploitasi dibandingkan dengan sistem yang batasan kepercayaannya mencerminkan tingkat isolasi keamanan yang tinggi. Ketika penguji penetrasi (*penetration tester*) menemukan hubungan kepercayaan yang lemah, mereka dapat menargetkannya dengan input yang tidak valid atau telah dimanipulasi, dengan harapan dapat menemukan kerentanan yang tersembunyi. Dengan demikian, pemeriksaan kepercayaan memungkinkan kita untuk mendeteksi kapan komponen kritis berinteraksi dengan mitra komunikasi yang tidak terautentikasi, atau kapan komponen tersebut menerima input berbahaya dari sistem yang menghadap ke luar (*external-facing systems*).

Selain itu, pemetaan penyebaran kepercayaan (*trust propagation*) dapat menyoroti hubungan kepercayaan yang tidak perlu atau terlalu luas, sehingga membantu mengidentifikasi skenario di mana hak akses yang berlebihan dapat secara tidak sengaja membuka jalan bagi akses tidak sah ke data sensitif. Sebagai contoh, bayangkan sebuah skenario di mana sebuah ECU berkinerja tinggi menganggap semua aplikasi yang berjalan di dalam *System-on-Chip* (SoC) berada dalam satu batasan kepercayaan yang sama.

Kesalahan konfigurasi pada kontrol akses sistem berkas seperti memberikan hak akses baca/tulis kepada setiap aplikasi terhadap berkas konfigurasi sistem memungkinkan aplikasi berbahaya atau aplikasi yang telah disusupi untuk menimpa pembaruan *firmware* atau mengekstrak data rahasia yang seharusnya hanya dapat diakses oleh proses dengan hak akses lebih tinggi. Akibatnya, penyerang yang berhasil menyusup ke satu aplikasi berhak akses rendah dapat meningkatkan hak aksesnya (*privilege escalation*) di seluruh sistem, yang melanggar prinsip hak akses minimum (*principle of least privilege*) dan membahayakan keamanan secara keseluruhan. Dengan memeriksa izin sistem berkas selama fase penemuan (*discovery phase*), penguji penetrasi (*penetration tester*) dapat memahami hubungan kepercayaan di dalam SoC (*System-on-Chip*) sebagai persiapan untuk mengeksploitasi batasan kepercayaan yang terlalu luas.

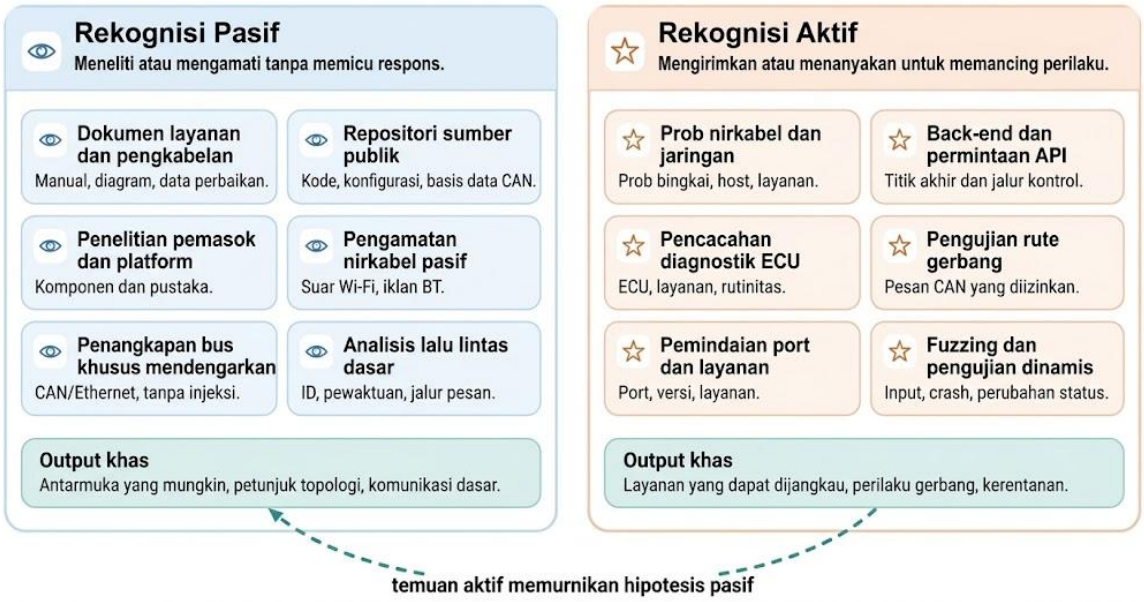
Fase Penemuan Dalam Praktik

Fase penemuan sering kali menjadi bagian yang paling menyita waktu dalam pengujian penetrasi, karena melibatkan proses yang bergantian antara pengumpulan informasi dan upaya eksploitasi. Keberhasilan pengujian penetrasi sangat bergantung pada kualitas informasi yang dikumpulkan mengenai sistem target. Hal ini menjadi sangat krusial saat melakukan pengujian penetrasi *black-box* atau *grey-box*, di mana informasi yang diberikan oleh pemilik komponen sangat terbatas.

Pertimbangkan skenario di mana Anda bertujuan untuk menyusup ke dalam ECU (*Electronic Control Unit*) yang tertanam jauh di dalam jaringan CAN (*Controller Area Network*) kendaraan. Tujuan serangan Anda adalah untuk membangun kemampuan kendali (*command-and-control*) atas jaringan CAN dari luar kendaraan. Dengan memanfaatkan pengetahuan Anda tentang jaringan otomotif, Anda berupaya untuk berkomunikasi langsung dengan ECU target atau mengidentifikasi ECU lain yang dapat meneruskan pesan Anda. Aktivitas penemuan Anda mencakup pemetaan arsitektur elektrik dan elektronik (E/E) kendaraan untuk mengidentifikasi titik akses, seperti antarmuka OBD (*On-Board Diagnostics*). Anda mungkin menemukan *gateway* jaringan pusat yang memisahkan ECU target dari antarmuka OBD, yang pada dasarnya berfungsi sebagai proksi. Temuan ini mendorong investigasi lebih lanjut untuk mengidentifikasi pesan CAN spesifik yang diizinkan oleh *gateway* pusat untuk diteruskan menuju target Anda.

Pada akhir proses ini, Anda telah mengumpulkan informasi yang cukup mengenai pengidentifikasi pesan CAN (*CAN message identifiers*) dan pemetaannya ke saluran CAN individual untuk mencapai tujuan serangan Anda. Contoh sederhana ini menegaskan pentingnya fase penemuan dalam merencanakan dan melaksanakan serangan dengan sukses. Untuk menjalankan fase penemuan secara efektif, kita perlu memahami dua pendekatan

dalam fase ini: pengintaian pasif *(passive reconnaissance*) dan pengintaian aktif (*active reconnaissance*), sebagaimana diilustrasikan dalam Gambar 2.2.



Gambar 2.2: Metode penemuan pasif dan aktif

Pengintaian pasif (*passive reconnaissance*) adalah pengumpulan informasi tanpa melakukan pemeriksaan langsung, memberikan perintah, mengirimkan data, atau berinteraksi dengan sistem target dengan cara yang bertujuan untuk memicu respons. Tujuannya adalah untuk membangun pemahaman awal mengenai arsitektur target, antarmuka yang terbuka, komponen-komponen yang mungkin ada, dependensi perangkat lunak, jalur komunikasi, dan potensi permukaan serangan (*attack surface*), sebelum beralih ke analisis artefak secara offline atau pengujian aktif.

Dalam pengujian penetrasi otomotif, pengintaian pasif biasanya terbagi dalam dua bentuk. Pertama, penelitian berdasarkan informasi yang tersedia untuk umum, tersedia secara komersial, atau telah dikumpulkan sebelumnya. Ini dapat mencakup manual servis, diagram kelistrikan, dokumentasi perbaikan, informasi penarikan kembali (*recall*), paten, pengumuman dari pemasok, repositori kode sumber publik, forum teknis, makalah penelitian, serta panduan pemasangan suku cadang aftermarket. Kedua, pengamatan pasif terhadap lingkungan target. Dalam hal ini, penguji mungkin memiliki akses ke kendaraan, perangkat pengujian di meja kerja (*bench setup*), lingkungan nirkabel, atau penyadapan jaringan (*network tap*), namun hanya mengamati komunikasi atau emisi yang sudah ada. Penguji tidak mengirimkan frame, permintaan diagnostik, upaya autentikasi, atau tindakan lain yang menyebabkan target memberikan respons.

Perbedaan ini sangat penting saat membahas pemindaian (*scanning*). Mendengarkan beacon Wi-Fi, iklan Bluetooth, aktivitas NFC, siaran V2X, atau emisi frekuensi radio lainnya termasuk dalam kategori pasif. Sebaliknya, mengirimkan permintaan probe (*probe requests*), frame nirkabel yang dirancang khusus, pesan diagnostik, permintaan API, atau upaya

autentikasi merupakan pengintaian aktif. Demikian pula, mengamati lalu lintas CAN, Ethernet, atau FlexRay hanya dianggap pasif jika penguji menggunakan konfigurasi "hanya mendengar" (*listen-only*) atau alat penyadap pasif yang tidak mengubah perilaku bus. Begitu penguji menyuntikkan pesan, meminta layanan, melakukan enumerasi ECU, atau mencoba melintasi segmen jaringan, aktivitas tersebut berubah menjadi pengintaian aktif.

Pengintaian pasif sangat berharga karena membantu penguji menyusun hipotesis sebelum berinteraksi dengan target. Sebagai contoh, dokumentasi servis mungkin mengungkap keberadaan *gateway* pusat, pengumuman pemasok dapat mengidentifikasi chipset telematika, repositori publik mungkin menampilkan file konfigurasi atau basis data CAN, dan pengamatan nirkabel secara pasif dapat menunjukkan bahwa kendaraan tersebut menawarkan layanan Wi-Fi atau Bluetooth. Temuan-temuan ini tidak membuktikan adanya kerentanan yang dapat dieksploitasi, namun membantu mempersempit ruang lingkup pencarian dan memberikan panduan untuk pengujian tahap selanjutnya.

Sumber Informasi Publik dan Komersial

Ketika informasi mengenai target terbatas, tim keamanan ofensif akan meneliti sistem target dengan memanfaatkan sumber informasi yang tersedia untuk publik maupun secara komersial. Salah satu teknik yang populer adalah menelusuri portal layanan OEM yang dirancang bagi mekanik maupun pengguna yang melakukan perbaikan sendiri (DIY) untuk mendapatkan skema kelistrikan, panduan servis, bahkan citra (*image*) *firmware* untuk pembaruan perangkat lunak resmi dari OEM. Sebagai contoh, Škoda, merek mobil yang merupakan bagian dari grup Volkswagen, menyediakan dokumen teknis, diagram kelistrikan, protokol diagnostik, dan panduan perbaikan yang dapat dibeli untuk akses jangka pendek maupun jangka panjang melalui portal Škoda Erwin.

Sumber lain yang sering digunakan adalah forum pengguna dan blog teknis, yang dapat mengungkap detail krusial mengenai pembaruan sistem atau kerentanan yang berguna dalam menyusun rangkaian serangan (*attack chain*). Sering kali ditemukan dokumen teknis di forum pengguna yang dibagikan oleh para penggemar otomotif; mereka kerap berbagi kiat, seperti cara memodifikasi berkas kalibrasi atau mengganti komponen OEM dengan suku cadang dari pihak ketiga (*aftermarket*).

Jika tujuannya adalah untuk menyusupi sistem *backend* berbasis cloud pada kendaraan, kueri DNS publik dapat mengungkap domain yang digunakan untuk pembaruan *over-the-air* (OTA) atau layanan telemetri, yang berpotensi mengekspos kerentanan terkait. Sumber informasi lain yang sering terabaikan adalah media sosial dan komunikasi perusahaan. Para insinyur mungkin secara tidak sengaja mengungkapkan detail penting mengenai sistem kendaraan terhubung (*connected vehicle*) saat membuat lowongan pekerjaan, profil LinkedIn, atau bahkan tulisan di blog. Hal ini dapat memberikan wawasan mengenai arsitektur perangkat lunak dan perangkat keras produk tersebut.

Sumber informasi lain yang tidak lazim adalah siaran pers yang mengumumkan kemitraan antara pemasok ECU dan vendor komponen; informasi ini dapat mengungkap detail mengenai komponen jaringan, cip memori, dan pustaka perangkat lunak (*software libraries*) yang digunakan dalam ECU tertentu. Beberapa vendor perangkat lunak sering kali dengan

antusias menonjolkan pemasok otomotif ternama yang menggunakan produk mereka, termasuk membagikan kisah sukses yang menyebutkan secara spesifik produk yang mengintegrasikan komponen mereka.

Informasi ini bisa sangat berguna jika, misalnya, mereka menawarkan pustaka yang mengandung kerentanan yang dapat Anda eksploitasi. Dalam beberapa kasus, produsen kendaraan (OEM) akan mengiklankan bahwa kendaraan mereka terintegrasi dengan merek aftermarket tertentu, seperti pemasok unit telematika atau sistem infotainment. Mendapatkan unit-unit ini sering kali lebih mudah, dan vendornya pun cenderung terbuka dalam menyediakan dokumentasi pemasangan yang tidak hanya mengungkap detail mengenai unit itu sendiri, tetapi juga arsitektur kelistrikan/elektronik (E/E) tempat unit tersebut diintegrasikan.

Sumber daya lain yang berguna adalah repositori kode sumber (*source code*) publik. Dalam beberapa tahun terakhir, terdapat kejadian di mana OEM otomotif secara tidak sengaja memberikan akses ke repositori publik akibat pengaturan akses yang keliru. Sebagai contoh, pada tahun 2023, sebuah OEM besar secara tidak sengaja mengekspos kunci akses (*keys*) mereka di repositori publik, sehingga kode tersebut dapat dilihat oleh siapa saja. Meskipun pengembang otomotif mungkin menganggap kode mereka bersifat rahasia, sifat rantai pasok otomotif yang sangat terdistribusi meningkatkan risiko kebocoran kode. Selain kebocoran melalui repositori publik, kita mungkin dapat menemukan basis data CAN untuk arsitektur E/E kendaraan tertentu langkah awal yang krusial untuk menyusun frame CAN berbahaya saat melakukan penyamaran identitas (*spoofing*) terhadap suatu node CAN.

Setelah menelusuri sumber-sumber di atas, Anda juga dapat mempertimbangkan makalah penelitian, presentasi, atau blog yang membahas model kendaraan lama atau platform serupa. Informasi tersebut dapat memberikan wawasan berharga mengenai kerentanan keamanan yang mungkin masih relevan untuk model yang lebih baru, atau model yang dikembangkan oleh perusahaan afiliasi. Banyak produsen peralatan asli (OEM) berbagi komponen dengan anak perusahaan mereka; oleh karena itu, mengetahui platform kendaraan mana yang dibangun di atas platform bersama dapat berguna untuk mengidentifikasi potensi kerentanan yang mungkin berlaku pada sistem target Anda.

Pengamatan Nirkabel dan Frekuensi Radio (RF) Secara Pasif

Pengamatan nirkabel secara pasif mencakup pemantauan aktivitas frekuensi radio tanpa melakukan transmisi. Dalam konteks otomotif, hal ini dapat meliputi pemantauan suar *(beacon*) Wi-Fi, iklan Bluetooth, perilaku NFC, sinyal yang terkait dengan kunci nirkabel (*key-fob*), perilaku modem seluler yang dapat dideteksi melalui peralatan uji, sinyal pemantauan tekanan ban, atau siaran V2X (komunikasi kendaraan dengan segala sesuatu di sekitarnya) jika tersedia. Tujuannya adalah untuk mengidentifikasi antarmuka nirkabel apa saja yang ada, kapan antarmuka tersebut aktif, pengenal (*identifier*) apa yang diungkapkan, serta apakah terdapat metadata berguna mengenai kendaraan atau komponen-komponennya.

Sebagai contoh, seorang penguji mungkin mengamati bahwa sistem infotainment mengiklankan titik akses Wi-Fi, antarmuka Bluetooth menampilkan nama perangkat yang terkait dengan model kendaraan, atau perangkat yang terhubung ke *backend* di sekitar

kendaraan berkomunikasi pada waktu-waktu tertentu yang dapat diprediksi. Pengamatan ini membantu mengidentifikasi kemungkinan titik masuk, namun tetap bersifat pasif selama penguji tidak mengirimkan probe (permintaan pemeriksaan), mencoba melakukan pemasangan (*pairing*), melakukan autentikasi, bergabung dengan jaringan, atau mengirimkan paket data yang telah dimanipulasi.

Penyadapan Pasif dan Pengambilan Lalu Lintas Dasar (*Baseline Traffic Capture*)

Penyadapan (*eavesdropping*), yang juga dikenal sebagai network sniffing, merupakan tindakan mencegat dan menganalisis komunikasi secara pasif tanpa mengubah aliran data. Pada kendaraan yang terhubung (*connected vehicles*), hal ini dapat mencakup pengamatan lalu lintas pada protokol CAN, Ethernet, FlexRay, LIN, V2X, Wi-Fi, Bluetooth, atau saluran yang terhubung ke *backend*. Penyadapan pasif dapat mengungkap pengidentifikasi pesan, pola waktu, ketergantungan layanan, perilaku diagnostik, aliran data sensor, serta hubungan antar-ECU (*Electronic Control Unit*).

Saat memeriksa jaringan di dalam kendaraan, penguji dapat menggunakan perangkat seperti penganalisis CAN, Ethernet tap, alat penangkap paket, atau penganalisis logika untuk mengumpulkan lalu lintas dasar. Untuk CAN, antarmuka harus dikonfigurasi dalam mode *listen-only* (hanya mendengar) atau mode senyap (*silent mode*) jika pengamatan pasif yang ketat diperlukan; jika tidak, node CAN biasa mungkin akan memberikan konfirmasi (*acknowledge*) terhadap frame data dan memengaruhi perilaku bus komunikasi. Untuk Ethernet, pengambilan data secara pasif harus dilakukan melalui tap, mirror port, atau pengaturan serupa yang tidak menyuntikkan lalu lintas data ke dalam jaringan.

Pengambilan lalu lintas dasar sering kali menjadi jembatan antara pengintaian pasif (*passive reconnaissance*) dan pengujian aktif yang dilakukan kemudian. Dengan mengamati komunikasi normal terlebih dahulu, penguji dapat mengidentifikasi tingkat pesan yang diharapkan, urutan saat sistem dinyalakan (*startup sequences*), sesi diagnostik, koneksi ke *backend*, serta ketergantungan antar-ECU. Nantinya, data dasar ini dapat digunakan untuk merancang pengujian replay, kampanye *fuzzing*, probe diagnostik, atau eksperimen penyaringan *gateway* selama tahap pengintaian aktif.

Pengintaian Aktif (*Active Reconnaissance*)

Setelah menyelesaikan kegiatan pengintaian pasif, para profesional keamanan siber (dari sisi ofensif) beralih ke fase krusial, yaitu pengintaian aktif. Tahap ini melibatkan interaksi langsung dengan sistem kendaraan untuk memperoleh wawasan yang lebih mendalam mengenai potensi kerentanan. Fase ini mencakup pemeriksaan aktif terhadap sistem *backend* kendaraan, antarmuka eksternal, jaringan di dalam kendaraan, serta penelusuran lebih dalam terhadap antarmuka perangkat keras ECU, API perangkat lunak, dan layanan yang ada. Dengan mendata dan memeriksa komponen-komponen tersebut, penguji penetrasi berupaya mengidentifikasi pintu belakang (*backdoor*) yang tersembunyi, konfigurasi yang tidak aman, serta protokol komunikasi yang dapat dieksploitasi. Tahap ini sangat penting untuk menentukan apakah sistem menyimpan kerentanan yang nantinya dapat dimanfaatkan saat menyusun rangkaian serangan.

Membangun Perangkat Uji (*Test Bench*)

Cara membangun perangkat uji sering kali bergantung pada jenis tim yang melakukan pekerjaan tersebut. Dalam kegiatan keamanan ofensif professional baik yang dilakukan oleh tim *red team* internal maupun tim pengujian penetrasi (*penetration testing*) pihak eksternal perusahaan otomotif biasanya perlu menyediakan ECU target, komponen kendaraan yang representatif, diagram kelistrikan, kebutuhan daya, dokumentasi antarmuka, serta perangkat pendukung yang diperlukan. Hal ini memastikan bahwa lingkungan pengujian mencerminkan platform kendaraan yang dituju, serta menjaga agar pengujian tetap berada dalam lingkup dan batasan keamanan yang telah disepakati.

Peneliti keamanan independen, pelajar, atau penghobi mungkin tidak memiliki akses ke komponen yang disediakan oleh produsen. Dalam situasi seperti ini, mereka sering memperoleh ECU, *gateway*, unit infotainment, modul telematika, rangkaian kabel (*wiring harness*), atau kendaraan rusak dari tempat barang rongsokan (*salvage yard*) maupun pasar daring. Membeli kendaraan yang rusak parah dari tempat barang rongsokan terkadang menjadi cara yang hemat biaya untuk membangun perangkat uji yang realistis, karena rangkaian kabel dan banyak komponen elektronik mungkin masih utuh meskipun bodi atau sistem penggerak kendaraan sudah tidak dapat digunakan lagi.

Memiliki perangkat uji khusus memungkinkan penguji untuk mengeksplorasi kerentanan dengan aman, memutar ulang lalu lintas jaringan, melakukan pembaruan *firmware* berulang kali, menjalankan pengujian diagnostik, serta menerapkan teknik-teknik yang bersifat intrusif tanpa risiko merusak kendaraan yang layak jalan. Perangkat ini juga mendukung aspek reprodusibilitas: penguji dapat mengembalikan sistem ke kondisi yang diketahui, mereproduksi temuan, dan memvalidasi langkah mitigasi dalam kondisi yang terkendali.

Perangkat uji sangat berguna karena beberapa fungsi kendaraan tidak dapat diuji jika ECU diisolasi. Banyak ECU mengharapkan pesan berkala dari modul lain; jika pesan-pesan tersebut tidak ada, ECU mungkin masuk ke dalam status penurunan fungsi (*degraded state*) atau *timeout*, sehingga penguji tidak dapat mengamati perilaku yang seharusnya hanya muncul selama operasional kendaraan normal. Perangkat uji yang dilengkapi dengan ECU pendukung yang tepat, simulasi pesan, catu daya, dan kondisi jaringan yang sesuai dapat mengungkap fungsi serta jalur serangan yang mungkin tidak terlihat dalam kondisi lain. Selain itu, pendekatan ini membantu menghindari risiko praktis seperti terkurasnya baterai kendaraan, terhentinya fungsi-fungsi kendaraan, atau membuat kendaraan menjadi tidak dapat beroperasi selama proses pengujian berlangsung.

Teknik Untuk Pengintaian Aktif (*Active Reconnaissance*)

Pengintaian aktif biasanya dimulai dari pemetaan permukaan eksternal menuju analisis yang lebih terarah. Penguji sering kali memulainya dengan menangkap lalu lintas dasar (*baseline traffic*), memetakan antarmuka nirkabel dan *backend*, serta mengidentifikasi ECU dan layanan apa saja yang dapat diakses. Tergantung pada lingkup penugasan, mereka dapat beralih ke pemindaian kerentanan berbasis versi, analisis kode sumber atau *firmware*, *fuzzing*, dan jika memang diperlukan pengintaian yang berfokus pada perangkat keras. Dalam

praktiknya, proses ini bersifat iteratif: temuan dari *fuzzing*, analisis *firmware*, atau pengujian perangkat keras sering kali mengharuskan penguji untuk kembali ke tahapan penemuan awal dengan hipotesis yang lebih baik.

Memahami bagaimana teknik-teknik ini diterapkan pada setiap lapisan arsitektur kendaraan merupakan landasan untuk menyusun rangkaian serangan yang efektif. Pada bab-bab selanjutnya, kita akan membahas lebih mendalam mengenai bagaimana teknik-teknik tersebut dapat dimanfaatkan untuk menemukan kelemahan keamanan yang kemudian dapat diubah menjadi eksploitasi nyata.

Memetakan Jaringan Nirkabel Dalam Ekosistem Otomotif

Sebagaimana dijelaskan dalam bab 1, kendaraan modern dilengkapi dengan berbagai antarmuka komunikasi nirkabel, seperti Wi-Fi, Bluetooth, dan jaringan seluler. Antarmuka ini memiliki beragam fungsi:

- **Sistem infotainmen**: Terhubung ke internet untuk keperluan navigasi, streaming, dan aplikasi pengguna.
- **Sistem telematika**: Berkomunikasi dengan layanan cloud untuk diagnostik, pembaruan perangkat lunak, dan pelacakan kendaraan.
- **Komunikasi V2X (*Vehicle-to-Everything*)**: Memungkinkan kendaraan berinteraksi dengan lingkungannya, termasuk dengan kendaraan lain maupun infrastruktur sekitar.

Jaringan-jaringan ini menghubungkan kendaraan ke infrastruktur eksternal dan layanan berbasis cloud, yang sekaligus menyediakan titik masuk potensial bagi penyerang.

Setelah jaringan nirkabel diidentifikasi (enumerasi), pemindaian aktif (*active scanning*) dapat dilakukan untuk berinteraksi dengan jaringan tersebut guna mengumpulkan informasi mengenai host yang aktif, layanan yang berjalan, serta protokol komunikasi yang digunakan:

- **Pemindaian Wi-Fi**: Dapat mengungkap jaringan di dalam kendaraan yang keamanannya lemah atau titik akses (*access point*) yang masih menggunakan pengaturan bawaan (*default*).
- **Interaksi dengan jaringan seluler**: Dapat mengekspos kelemahan dalam mekanisme komunikasi jarak jauh kendaraan, seperti kerentanan pada modem seluler atau serangan penurunan tingkat keamanan protokol (*protocol security downgrade attacks*).
- **Analisis protokol**: Memahami protokol yang digunakan dapat membantu mengidentifikasi kelemahan dalam standar komunikasi atau implementasinya.

Salah satu temuan umum selama pemindaian aktif adalah keberadaan jaringan Wi-Fi yang tidak diamankan dengan baik, baik di dalam kendaraan itu sendiri maupun di infrastruktur sekitarnya. Sebagai contoh, koneksi Wi-Fi pada kendaraan mungkin rentan karena enkripsi yang lemah atau penggunaan kredensial bawaan, yang memungkinkan penyerang untuk terhubung ke jaringan kendaraan tersebut dan mengakses sistem yang sensitif.

Seorang penyerang yang telah menemukan kerentanan Wi-Fi spesifik pada model kendaraan tertentu mungkin akan berkeliling mencari model tersebut hingga berada dalam jangkauan Wi-Fi untuk melancarkan serangannya. Hal ini pernah ditunjukkan oleh peretas

yang berhasil mengakses sistem infotainmen Tesla dengan menerbangkan drone yang membawa perangkat Wi-Fi (*Wi-Fi dongle*) di dekat model Tesla yang rentan tersebut.

Analisis API Aplikasi dan Back-End

Mengingat konektivitas back-end telah menjadi hal yang lazim, kita tidak boleh mengabaikan aktivitas pengintaian (*reconnaissance*) yang mencakup sistem *back-end*. Teknik penemuan utama meliputi intersepsi koneksi antara aplikasi seluler kendaraan dan server *back-end*-nya, yang sangat berguna untuk mengidentifikasi potensi kelemahan. Kelemahan ini dapat berupa endpoint API yang tidak aman, yang membocorkan informasi sensitif kendaraan atau bahkan memungkinkan fungsi kendali jarak jauh yang berbahaya.

Sebagai contoh, menganalisis lalu lintas antara komputer pusat kendaraan dan server back-end OEM dapat mengungkap mekanisme autentikasi yang lemah atau tidak ada sama sekali, serta data yang tidak dienkripsi yang rentan terhadap serangan *man-in-the-middle*. Dengan meniru aplikasi kendaraan yang terhubung ke *back-end*, penguji dapat memanipulasi variabel, menyimulasikan berbagai skenario, dan mengamati bagaimana aplikasi tersebut menangani pengiriman serta penerimaan data. Analisis ini membantu mengidentifikasi praktik penyimpanan data yang tidak aman, enkripsi yang tidak memadai, serta kerentanan lain yang dapat mengancam privasi pengguna dan keamanan kendaraan.

Enumerasi ECU dan Penemuan Diagnostik

Metode utama lainnya yang digunakan selama pengintaian aktif adalah penggunaan perangkat diagnostik khusus dari produsen kendaraan untuk mengumpulkan informasi terperinci mengenai ECU yang terpasang dan rutinitas diagnostik yang didukungnya. Dengan berinteraksi melalui sistem diagnostik kendaraan, penguji dapat melakukan hal-hal berikut:

- **Mengidentifikasi ECU yang tersedia**: Mendeteksi ECU apa saja yang ada pada jaringan di dalam kendaraan.
- **Menemukan rutinitas layanan yang tersedia**: Mengidentifikasi layanan dan subrutin diagnostik yang ditawarkan oleh setiap ECU.
- **Menentukan interaksi ECU**: Memahami bagaimana ECU berinteraksi dengan klien diagnostik, baik yang ada di dalam maupun di luar kendaraan.
- **Memetakan pengenal (*identifier*) CAN dan layanan yang didukung**: Memperoleh pengetahuan tentang pengenal CAN yang digunakan oleh klien diagnostik beserta layanan yang didukungnya.

Informasi ini menjadi sangat berharga saat menyimulasikan penyerang yang memiliki akses ke jaringan CAN. Sebagai contoh, pertimbangkan topologi jaringan di mana ECU Telematika dan antarmuka OBD sama-sama terhubung ke ECU *Gateway* pusat. ECU *Gateway* ini mungkin dirancang untuk memblokir rutinitas diagnostik tertentu agar tidak diteruskan ke jaringan lain jika berasal dari unit telematika. Mengetahui bagaimana ECU *Gateway* merutekan perintah diagnostik merupakan langkah awal dalam menyusun rantai serangan, di mana unit telematika yang telah disusupi dapat menyamar sebagai klien diagnostik eksternal untuk melampaui mekanisme penyaringan di *Gateway*.

Pemindaian Jaringan dan Enumerasi Port

Dengan asumsi bahwa penyerang telah berhasil mendapatkan akses ke dalam jaringan kendaraan, pemindaian terhadap layanan yang tidak aman atau protokol komunikasi yang dikonfigurasi secara tidak tepat merupakan teknik yang ampuh untuk menemukan kerentanan berbasis jaringan. Mengingat jaringan kendaraan kini perlahan mulai menyerupai jaringan perusahaan, berbagai teknik yang lazim digunakan pada jaringan perusahaan tersebut juga dapat diterapkan pada kendaraan, sebagaimana akan kita bahas selanjutnya.

Pemindaian port (*Port scanning*)

Pemindaian port melibatkan penggunaan perangkat seperti Nmap untuk memindai port yang terbuka pada sistem tertanam (*embedded systems*) kendaraan melalui berbagai antarmuka yang terekspos. Metode ini sangat efektif terhadap ECU (*Electronic Control Unit*) berkinerja tinggi yang menyediakan layanan jaringan, serupa dengan yang ditemukan pada komputer pada umumnya. Aktivitas ini dapat mengidentifikasi layanan yang berjalan pada antarmuka komunikasi internal maupun eksternal kendaraan. Dengan berinteraksi dengan layanan-layanan tersebut, penguji dapat melakukan tindakan berikut:

- **Memetakan struktur sistem**: Memahami bagaimana berbagai komponen berkomunikasi dan mengidentifikasi topologi jaringan.
- **Mengidentifikasi titik lemah potensial**: Mendeteksi perangkat lunak yang belum diperbarui (*unpatched*), saluran komunikasi yang terekspos, atau layanan yang tidak diamankan.
- **Menilai kerentanan layanan**: Menentukan apakah layanan yang berjalan sudah usang atau rentan terhadap eksploitasi yang sudah diketahui.

Analisis Artefak

Selain kegiatan pengintaian (*reconnaissance*) pasif maupun aktif, sebagian besar pekerjaan tim keamanan ofensif terletak pada analisis artefak yang ditemukan. Bergantung pada tingkat informasi yang tersedia, tim dapat menggunakan beberapa teknik untuk mengubah sumber daya yang telah dikumpulkan menjadi temuan potensial.

Analisis Kode Statis dan Analisis Aliran Informasi

Saat melakukan *white box* pentesting dengan akses ke kode sumber (*source code*), pemeriksaan kode statis seperti pemeriksaan aturan pengodean CERT C/C++ menjadi metode yang ampuh untuk menemukan kelemahan keamanan. Analisis statis memeriksa kode program tanpa menjalankannya, dengan fokus pada pemahaman berbagai kemungkinan jalur eksekusi. Jika kode sumber tidak tersedia, *Static Application Security Testing* (SAST) berbasis biner dapat digunakan untuk memeriksa berkas yang telah dikompilasi (seperti binary) melalui proses disassembly dan analisis struktur kode. Kedua jenis pengujian berbasis SAST ini memiliki keterbatasan sebagai berikut:

- **Tingkat *false-positive* yang tinggi**: Mengidentifikasi sesuatu yang sebenarnya bukan kerentanan sebagai sebuah masalah, karena adanya pelanggaran aturan pengodean yang tidak memiliki jalur eksploitasi praktis.

- **Kegagalan dalam mendeteksi kerentanan spesifik saat runtime**: Menyebabkan false-negative karena ketidakmampuan untuk meniru perebutan sumber daya (*resource contention*) atau kondisi *race condition*.
- **Memerlukan verifikasi manual**: Hasil pengujian tidak dapat divalidasi secara otomatis, sehingga memerlukan tinjauan dari pakar untuk mengonfirmasi masalah tersebut.

SAST yang Disempurnakan Dengan AI

AI generatif kini semakin banyak diterapkan dalam SAST (*Static Application Security Testing*) untuk meningkatkan kegunaan hasil analisis statis, bukan untuk menggantikan mesin analisis dasarnya. Alat SAST tradisional tetap berharga karena mampu mengurai kode, menegakkan aturan penulisan kode yang aman, serta menghasilkan temuan yang dapat direproduksi. Namun, alat-alat tersebut sering kali kesulitan menentukan apakah suatu masalah yang dilaporkan benar-benar dapat dieksploitasi, apakah logika sanitasi sudah memadai, atau apakah temuan tersebut relevan dalam konteks sistem yang lebih luas. SAST yang disempurnakan dengan AI mengatasi kesenjangan ini dengan memadukan keluaran analisis statis dengan kemampuan penalaran kontekstual dan pemahaman kode yang dimiliki oleh model bahasa besar (*Large Language Models*/LLM).

Dalam alur kerja SAST yang disempurnakan dengan AI, pemindai konvensional pertama-tama mengidentifikasi potensi kelemahan, seperti operasi memori yang tidak aman, kurangnya validasi input, penggunaan API yang tidak aman, kredensial yang tertanam dalam kode (*hardcoded*), atau aliran data yang tercemar dari sumber yang tidak tepercaya menuju titik akhir (*sink*) yang sensitif. Selanjutnya, LLM meninjau temuan tersebut dalam konteksnya termasuk kode di sekitarnya, jalur pemanggilan (*call paths*), jejak aliran data, dokumentasi API, aturan penulisan kode, serta deskripsi kerentanan. Hal ini memungkinkan penyajian penjelasan yang lebih mendalam untuk menilai apakah suatu temuan kemungkinan besar merupakan *true positive* (temuan yang benar-benar valid), sekaligus memberikan rekomendasi langkah perbaikan yang spesifik untuk basis kode tersebut.

Penelitian terkini menunjukkan bahwa pendekatan hibrida ini sangat berguna untuk mengurangi beban kerja analis. Sebagai contoh, sistem berbasis LLM seperti IRIS menggabungkan analisis statis dengan penalaran kontekstual berbasis LLM untuk mendeteksi kerentanan di seluruh repositori. Pendekatan lain menggunakan LLM untuk melakukan triase terhadap peringatan SAST guna mengurangi *false positive* (temuan yang tidak valid). Berbagai perangkat komersial pun mulai mengarah ke pendekatan ini. GitHub Copilot Autofix, misalnya, memanfaatkan peringatan pemindaian kode dari CodeQL disertai dengan konteks kode dan penjelasan yang dihasilkan oleh LLM untuk menyarankan perbaikan bagi temuan keamanan. Meskipun bermanfaat bagi tim pertahanan, perangkat ini juga sangat berguna bagi tim keamanan ofensif yang melakukan *white-box penetration testing*. Alat-alat tersebut membantu mempertajam fokus pada masalah nyata yang layak ditindaklanjuti, sehingga tim dapat mengalokasikan waktu untuk menyusun rantai serangan (*attack chains*) yang lebih berpeluang membuahkan hasil.

Terlepas dari berbagai manfaat tersebut, SAST yang disempurnakan dengan AI sebaiknya dipandang sebagai kemampuan pendukung, bukan sebagai pengambil keputusan

keamanan yang bersifat otoritatif dalam menentukan apakah suatu temuan benar-benar merupakan kerentanan yang valid. LLM dapat mengarang penjelasan, salah memahami alur kendali, melewatkan kondisi kerentanan yang tidak kentara, atau mengusulkan perbaikan yang justru mengubah perilaku sistem yang seharusnya. Oleh karena itu, temuan dan perbaikan yang dihasilkan harus divalidasi melalui tinjauan pakar, pengujian regresi, pemeriksaan keamanan kode, serta pengujian dinamis atau *fuzzing* jika diperlukan. Jika langkah-langkah ini tidak dilakukan, dapat muncul temuan positif palsu (*false positive*) yang akan menurunkan kredibilitas tim pengujian penetrasi ketika temuan tersebut terbukti sebagai celah yang tidak dapat dieksploitasi.

Rekayasa Balik *Firmware*

Ketika kode sumber tidak tersedia seperti dalam pengujian penetrasi *black box* rekayasa balik (*reverse engineering*) *firmware* menjadi salah satu metode utama untuk menemukan kerentanan pada kode maupun logika sistem. Rekayasa balik *firmware* merupakan proses yang sangat intensif, membutuhkan banyak waktu serta pengetahuan khusus untuk mengungkap kerentanan dalam sistem tertanam (*embedded systems*). Proses ini diawali dengan akuisisi *firmware*; sering kali *firmware* harus diekstraksi langsung dari memori perangkat karena sifatnya yang bersifat eksklusif (*proprietary*) dan adanya upaya dari produsen untuk melindungi kekayaan intelektual mereka.

Teknik yang digunakan untuk tujuan ini mencakup pemrograman *flash*, pemantauan bus, atau modifikasi sirkuit *Joint Test Action Group* (JTAG) guna mengatasi fitur keamanan perangkat keras. Karena komputer berkinerja tinggi (*High Performance Computers* atau HPC) menjalankan instruksi dari *Dynamic Random-Access Memory* (DRAM), penggunaan interposer DRAM menjadi teknik yang umum untuk mengekstraksi *firmware* setelah dimuat ke dalam memori tahap di mana *firmware* tersebut kemungkinan besar sudah didekripsi dari penyimpanan *flash* eksternal.

Setelah akuisisi, langkah berikutnya adalah pembongkaran dan ekstraksi *firmware*, yang mencakup penguraian berkas *firmware* untuk mengakses isinya. Tahap ini sering kali menjadi rumit akibat penggunaan format eksklusif atau enkripsi, sehingga memerlukan perangkat khusus seperti Binwalk atau FRAK untuk melakukan analisis. Tingkat keberhasilan pada tahap ini bisa bervariasi, mengingat perangkat canggih sekalipun terkadang hanya mampu membongkar sebagian isi dari jenis *firmware* tertentu. Memahami struktur dan fungsionalitas *firmware* sangatlah penting untuk menemukan potensi bug atau pintu belakang (*backdoor*). Para ahli rekayasa balik (*reverse engineer*) menghadapi tantangan unik yang berbeda dari tantangan pada format standar seperti *Executable and Linkable Format* (ELF), yang memiliki header deskriptif untuk membantu identifikasi struktur program. Menangani *firmware* mentah memerlukan metodologi sistematis guna mengidentifikasi titik masuk (*entry point*), segmen kode, dan struktur data tanpa adanya petunjuk format yang lazim ditemukan.
Berikut adalah beberapa kiat untuk melakukan rekayasa balik (*reverse engineering*) *firmware* secara efektif:

- **Tentukan arsitektur sejak awal**: Lakukan pemeriksaan fisik jika memungkinkan, lalu verifikasi dengan alat otomatis untuk memastikan akurasinya.

- **Manfaatkan alat pemetaan memori (*memory mapping*)**: Alat seperti BinBloom dapat membantu menyimpulkan tata letak memori dengan mengelompokkan nilai alamat, sehingga menyoroti area kode, bagian data, dan fungsi diagnostik yang mungkin ada.
- **Gunakan skrip untuk otomatisasi**: Mengotomatiskan identifikasi segmen kode melalui skrip dapat mengurangi kesalahan manusia dan mempercepat proses rekayasa balik.
- **Verifikasi penunjuk fungsi (*function pointer*) melalui analisis**: Jika metode konvensional tidak berhasil, analisis segmen kode menggunakan skrip dapat mengungkap alamat fungsi serta titik masuk (*entry point*) program.

Teknik yang sangat berkaitan dengan hal ini adalah *re-hosting firmware*, yaitu menjalankan *firmware* sistem tertanam baik secara keseluruhan maupun Sebagian di luar lingkungan ECU aslinya guna mendukung pengujian yang dapat diskalakan. Emulasi penuh menjadi pilihan menarik jika arsitektur prosesor, peta memori, dan periferal yang diperlukan dapat dimodelkan dengan cukup akurat untuk menjalankan *firmware* tanpa perangkat keras aslinya. Pendekatan ini menawarkan kontrol yang kuat, kemampuan pengulangan (*repeatability*), instrumentasi, dan otomatisasi.

Namun, *firmware* otomotif sering kali sangat bergantung pada periferal MCU dan SoC, seperti pengatur waktu (*timer*), interupsi, watchdog, serta lalu lintas jaringan dari ECU lain. Jika ketergantungan ini tidak dapat diemulasi sepenuhnya, penguji dapat menggunakan emulasi parsial, pemodelan periferal, stubbing, atau metode *hardware-in-the-loop*. Dalam konfigurasi ini, akses ke periferal yang tidak didukung dapat disimulasikan melalui perangkat lunak, diganti dengan model yang lebih sederhana, atau diteruskan ke perangkat keras nyata, sehingga *firmware* tetap dapat berjalan dalam kondisi yang cukup realistis.

Setelah *firmware* berhasil di-*re-host*, penguji dapat menerapkan berbagai teknik seperti *fuzzing*, pengujian berbasis cakupan (*coverage-guided testing*), eksekusi simbolik (*symbolic execution*), dan instrumentasi saat waktu berjalan (*runtime instrumentation*). Keberhasilan praktis dari *re-hosting* sangat bergantung pada seberapa akurat lingkungan pengujian dalam mereplikasi asumsi *firmware* terkait memori, periferal, pengaturan waktu, interupsi, dan status jaringan kendaraan.

Pemindaian Kerentanan Berbasis Versi

Jika berkas biner perangkat lunak tersedia atau diperoleh melalui salah satu metode yang disebutkan sebelumnya analisis komposisi perangkat lunak (*software composition analysis*) memungkinkan penguji untuk memindai perangkat lunak kendaraan guna mendeteksi kerentanan yang sudah diketahui. Teknik ini membantu mengidentifikasi versi perangkat lunak yang rentan dieksploitasi, kesalahan konfigurasi, atau layanan usang yang berpotensi menjadi titik masuk bagi penyerang.

Sebagai contoh, pemindaian biner (memeriksa berkas biner *firmware* untuk mencari kerentanan yang sudah diketahui) dapat mengungkap kelemahan yang tidak terlihat melalui pemindaian jaringan saja. Jika kita dapat menanyakan versi distribusi perangkat lunak tertentu pada ECU, maka kita dapat menemukan CVE tanpa perlu mengekstrak perangkat lunak ECU tersebut. Ini adalah teknik yang umum diterapkan, misalnya, saat menguji ECU berbasis Linux.

Analisis Dinamis dan Fuzzing

Analisis dinamis mengevaluasi perilaku program secara waktu nyata (*real-time*) saat perangkat lunak dijalankan, sehingga membantu mengidentifikasi kerentanan yang mungkin terlewatkan oleh analisis statis. Salah satu metode utama dalam analisis dinamis adalah *fuzzing*, yaitu pengujian perangkat lunak dengan cara menghasilkan berbagai masukan yang bervariasi, lalu mengamati bagaimana sistem menangani data yang tidak terduga, tidak sesuai format, atau berada dalam kondisi ekstrem (*edge-case*). Para penguji penetrasi (*penetration tester*) sering mengandalkan *fuzzing* untuk mengungkap kerentanan perangkat lunak serta kelemahan protokol yang sulit ditemukan hanya melalui pemeriksaan manual.

Cakupan *fuzzing* yang dapat dilakukan sangat bergantung pada tingkat akses penguji serta informasi yang disediakan oleh klien. Dalam pengujian *white-box*, penguji mungkin memiliki akses ke kode sumber, antarmuka internal, spesifikasi format berkas, dokumentasi protokol, simbol debug, atau perangkat pendukung (*harness*) yang memungkinkan pengujian langsung terhadap komponen perangkat lunak internal. Dalam pengujian *grey-box*, penguji mungkin menerima dokumentasi tertentu, berkas biner representatif, berkas awal (*seed files*), atau rincian antarmuka parsial. Dalam pengujian *black-box*, *fuzzing* biasanya terbatas pada API yang dapat diakses dari luar, protokol komunikasi, layanan diagnostik, pengurai berkas (*file parsers*), dan saluran masukan yang dapat diamati.

Dalam semua kasus, kualitas hasil *fuzzing* bergantung pada kualitas masukan yang digunakan. Melakukan *fuzzing* pada seluruh ruang kemungkinan masukan biasanya tidak efisien; oleh karena itu, penguji menggunakan teknik yang memfokuskan pengujian pada masukan yang paling mungkin memicu perilaku yang tidak terduga. Fuzzer berbasis mutasi (*mutation-based fuzzer*) menghasilkan kasus uji dengan mengambil masukan awal yang valid, lalu mengubahnya secara sistematis seperti mengubah nilai byte, menukar segmen, memotong kolom, atau menyisipkan data acak. Karena dimulai dari sampel yang diketahui valid, fuzzer berbasis mutasi dapat dengan cepat menghasilkan masukan yang rusak (*malformed*) namun tetap tampak realistis, tanpa memerlukan pemahaman menyeluruh mengenai protokol atau format berkas yang mendasarinya. Pendekatan ini sangat berguna ketika penguji memiliki sampel berkas biner, pesan yang tertangkap, berkas citra OTA (*over-the-air*), atau lalu lintas diagnostik, namun tidak memiliki spesifikasi yang terperinci.

Fuzzer berbasis generasi (*generation-based fuzzer*) membuat masukan uji dari awal menggunakan model protokol, format berkas, atau struktur data target. Fuzzer ini secara sistematis menyusun kolom, pesan, urutan, dan kasus-kasus ekstrem (*edge cases*) yang mengikuti format yang diharapkan, sembari memvariasikan nilai, panjang, urutan, dan validitasnya. Pendekatan ini dapat memberikan cakupan yang lebih mendalam dan terstruktur, namun sangat bergantung pada pengetahuan yang akurat mengenai format atau protokol target.

Sebagai contoh, *fuzzing* berbasis generasi mungkin tepat untuk perangkat lunak yang memproses masukan dari pengguna seperti berkas pembaruan OTA hanya jika penguji memiliki akses ke spesifikasi format berkas yang relevan, rincian penandatanganan atau pengemasan, aturan penguraian (*parsing*), atau dokumentasi internal lainnya. Pendekatan ini

lebih umum ditemukan dalam pengujian *white-box* atau *grey-box*. Jika informasi tersebut tidak tersedia, file *fuzzing* berbasis mutasi tetap dapat dilakukan dengan menggunakan berkas biner citra baik yang disediakan maupun yang dikumpulkan sebagai masukan awal (*seed inputs*).

Pertanyaan yang sering muncul adalah bagaimana *fuzzing* dapat bermanfaat jika satu-satunya hasil yang terlihat hanyalah *crash* (kerusakan sistem). *Crash, reboot* (mulai ulang sistem), hang (sistem membeku), kehabisan memori, reset oleh watchdog, serta perubahan status yang tidak terduga sering kali menjadi indikator bahwa *fuzzer* telah mencapai jalur kode yang rentan atau tidak ditangani dengan baik. Gejala-gejala ini dapat mengarah pada kerentanan yang bisa dieksploitasi, seperti *buffer overflow*, kesalahan penanganan bilangan bulat (*integer handling errors*), bug pada parser, *race condition*, atau masalah kehabisan sumber daya. Proses debugging tambahan, instrumentasi, *triase crash*, dan analisis akar masalah (*root cause analysis*) biasanya diperlukan untuk menentukan apakah perilaku tersebut dapat dieksploitasi.

Fuzzing juga sangat berguna untuk protokol komunikasi yang umum digunakan dalam sistem tertanam (*embedded systems*), termasuk CAN, UART, SPI, dan Automotive Ethernet. Dengan mengirimkan urutan pesan yang tidak terduga, frame yang tidak sesuai format, pola waktu yang tidak lazim, atau nilai kolom yang tidak valid, penguji dapat mengamati bagaimana ECU dan perangkat lunak pendukung menangani input yang berada di luar asumsi operasional normal. Satu metode *fuzzing* lain yang patut diperhatikan adalah *fuzzing* berbasis AI (*AI-enhanced fuzzing*), yang menggunakan teknik pembelajaran mesin (*machine learning*) atau penalaran otomatis untuk memandu pembuatan input, memprioritaskan kasus uji yang potensial, atau meningkatkan cakupan pengujian (*coverage*) berdasarkan perilaku program yang teramati.

Fuzzing Berbasis AI

Fuzzing berbasis AI dapat diterapkan dalam dua cara yang saling berkaitan. Pada tingkat protokol, fuzzer yang dibantu oleh AI memanfaatkan pengetahuan sebelumnya mengenai kerentanan sistem tertanam, protokol komunikasi, dan format perintah untuk memprioritaskan input secara lebih efektif dibandingkan dengan mutasi acak murni. Hal ini membuat *fuzzing* menjadi lebih terarah dan peka terhadap konteks, terutama saat menguji parser protokol dan antarmuka yang memiliki status (*stateful interfaces*).

Jika kode sumber tersedia, *fuzzing* berbasis AI akan menyerupai *white-box fuzzing*. Dalam konteks ini, model bahasa besar (*Large Language Model* atau LLM) dapat menganalisis kode, mengidentifikasi titik masuk (*entry point*) yang potensial, serta membantu dalam pembuatan *fuzzing harness*, input awal (*seed inputs*), atau logika pengujian yang mengarahkan eksekusi menuju jalur-jalur yang relevan dengan aspek keamanan. Alih-alih menggantikan mesin *fuzzing*, komponen AI berfungsi untuk memperbaiki pengaturan dan memberikan panduan, sehingga *fuzzing* yang berbasis cakupan (*coverage-guided fuzzing*) dapat menjangkau perilaku program yang lebih dalam dengan upaya manual yang lebih sedikit. Tren ini semakin terlihat pada perangkat-perangkat industri; sebagai contoh, *code intelligence* telah mendeskripsikan pendekatan berbasis AI di mana sebuah agen menganalisis kode, membuat pengujian *fuzzing*, menjalankannya, serta melaporkan kerentanan yang ditemukan secara

otonom. Untuk sistem tertanam (*embedded*) dan sistem berbasis protokol, pendekatan berlapis terhadap *fuzzing* yang diperkuat dengan kecerdasan buatan (AI) membantu memfokuskan pengujian pada kategori kelemahan tertentu:

- **Pengujian struktur perintah**: Fuzzer menghasilkan urutan perintah yang realistis dan spesifik untuk protokol tertentu guna mengungkap cacat dalam proses penguraian (*parsing*) dan penanganan pesan.
- **Pengujian batasan parameter**: Fuzzer memeriksa nilai parameter minimum, maksimum, yang tidak sesuai format, maupun yang tidak terduga untuk menemukan kesalahan overflow, underflow, dan validasi.
- **Pengujian waktu dan urutan**: Fuzzer memvariasikan urutan, jeda waktu, dan konkurensi perintah untuk mengungkap *race condition*, kesalahan pada *state-machine*, serta cacat sinkronisasi.

Pendekatan berlapis ini meningkatkan efisiensi dengan menggabungkan pemahaman terhadap protokol dan analisis yang lebih mendalam mengenai titik-titik kegagalan yang mungkin terjadi. Pendekatan ini akan jauh lebih efektif jika dipadukan dengan teknik *white-box fuzzing* yang memiliki akses ke kode sumber (*source-aware*).

Teknik Pengintaian Perangkat Keras

Sejauh ini, kita telah memfokuskan teknik penemuan pada aspek perangkat lunak dan jaringan. Namun, perangkat keras juga menjadi target yang menarik untuk kegiatan penemuan karena perannya yang mendasar sebagai landasan platform keamanan. Pengintaian yang berfokus pada perangkat keras biasanya dimulai dengan mengidentifikasi paket perangkat, keluarga prosesor atau SoC (*System-on-Chip*), perangkat memori, pin yang terbuka, antarmuka debug, serta komponen-komponen yang berkaitan dengan keamanan. Jika dokumentasi tersedia, penguji dapat memanfaatkannya untuk memahami mode boot, peta fuse, perilaku JTAG atau SWD, penyimpanan kunci aman, serta kontrol akses untuk debugging.

Jika dokumentasi tidak tersedia, penguji dapat mengandalkan pemeriksaan fisik, enumerasi pinout, deteksi antarmuka, dan analisis sinyal untuk menyimpulkan bagaimana perangkat tersebut dikonfigurasi. Saat menerapkan teknik penemuan pada perangkat keras, penyerang biasanya menggunakan tiga kategori serangan utama, yang akan kita bahas selanjutnya.

Serangan Non-Invasif

Serangan non-invasif dilakukan dengan mengamati perangkat tanpa memodifikasi silikon secara fisik atau mengganggu operasinya secara sengaja. Tujuannya sering kali untuk menyimpulkan informasi rahasia melalui kebocoran yang tidak disengaja. Contohnya mencakup teknik *side-channel* seperti *Simple Power Analysis* (SPA), *Differential Power Analysis* (DPA), analisis waktu (*timing analysis*), analisis elektromagnetik, kebocoran berbasis cache, dan pengamatan optik. Metode-metode ini biasanya digunakan untuk mengevaluasi apakah operasi kriptografis membocorkan informasi yang cukup untuk memulihkan kunci atau menyimpulkan status yang terlindungi.

Serangan Injeksi Kesalahan (*Fault-Injection*) Semi-Invasif atau Invasif

Serangan injeksi kesalahan semi-invasif atau invasif dilakukan dengan sengaja mengganggu perangkat untuk memicu kesalahan, melewati pemeriksaan, mengubah alur kontrol, atau memintas mekanisme keamanan. Contohnya meliputi *voltage glitching, clock glitching*, injeksi kesalahan elektromagnetik (*Electromagnetic Fault Injection* atau EMFI), injeksi kesalahan menggunakan laser, tekanan termal, serta serangan gangguan memori seperti Rowhammer. Teknik-teknik ini dapat digunakan untuk menguji apakah suatu ECU atau SoC dapat dipaksa untuk melewati pemeriksaan *secure boot*, memintas kunci debug, rutinitas autentikasi, atau keputusan kontrol akses.

Serangan Invasif Penuh dan Destruktif

Serangan invasif penuh dan destruktif memerlukan pembukaan kemasan (*de-capping*), modifikasi, atau pemeriksaan langsung pada cip. Tujuannya adalah untuk menginspeksi struktur internal, membaca jalur data internal, atau mengekstrak informasi rahasia dari penyimpanan yang terlindungi. Contoh-contohnya meliputi microprobing, rekayasa balik silikon (*silicon reverse engineering*), modifikasi menggunakan berkas ion terfokus (*focused ion beam* atau FIB), ablasi laser, serta pemeriksaan langsung terhadap struktur fuse atau memori yang hanya dapat diprogram sekali (*one-time-programmable* atau OTP). Metode-metode ini memerlukan biaya tinggi dan keahlian khusus, namun dapat dibenarkan saat mengevaluasi aset bernilai tinggi, seperti kunci privat akar (*root private keys*), kunci utama (*master keys*), atau bit konfigurasi keamanan.

Hasil dari ekstraksi aset perangkat keras dan penemuan kelemahan perangkat keras berguna untuk tahapan pengujian penetrasi selanjutnya. Sebagai contoh, penemuan bahwa fuse bit dapat menempatkan cip ke dalam kondisi yang tidak aman, bahwa kunci debug (*debug lock*) dapat dilewati melalui injeksi kesalahan (*fault injection*), atau bahwa operasi kriptografis membocorkan informasi yang bergantung pada kunci, dapat menjadi dasar untuk eksploitasi di kemudian hari. Dalam konteks ini, ekstraksi aset perangkat keras termasuk dalam kategori pengintaian aktif (*active reconnaissance*) karena metode ini mengidentifikasi aset yang dilindungi, asumsi keamanan, dan jalur serangan yang tidak terlihat melalui pemindaian jaringan, analisis *firmware*, atau pengujian aplikasi saja.

Menghadapi Jalan Buntu Selama Fase Penemuan

Saat melakukan fase penemuan, sering kali kita menelusuri jalur investigasi yang tampak menjanjikan namun pada akhirnya tidak membuahkan hasil sebuah skenario yang sering digambarkan sebagai "masuk ke dalam lubang kelinci" (*venturing down a rabbit hole*). Mengenali kapan Anda mendekati jalan buntu sangat penting untuk mengelola waktu dan sumber daya secara efektif. Karena penguji penetrasi bekerja dengan anggaran terbatas dan jadwal yang ketat, sangatlah krusial untuk segera menyesuaikan strategi guna menghindari upaya yang tidak produktif.

Membuat keputusan yang tepat mengenai langkah selanjutnya juga sangat penting agar tidak membuang waktu pada petunjuk yang tidak produktif. Penilaian ini bergantung pada pengalaman serta batasan-batasan dalam penugasan tersebut. Bahkan jika Anda berada di jalur yang salah, pelajaran yang didapat tetap bisa berguna untuk upaya di masa depan. Oleh

karena itu, mengetahui kapan harus beralih ke jalur investigasi lain sangatlah penting untuk menghindari jalan buntu dan menjaga momentum menuju tujuan Anda. Meskipun ketekunan adalah sifat yang berharga, penting untuk menimbang potensi hasil dari suatu pendekatan dibandingkan dengan waktu dan sumber daya yang dihabiskan. Dengan demikian, menetapkan tujuan dan kerangka waktu yang jelas untuk setiap jalur investigasi dapat mencegah investasi berlebih pada area yang tidak produktif.

Terkadang, selama investigasi, Anda mungkin menemukan hal-hal tak terduga yang meskipun tidak berkaitan langsung dengan tujuan awal layak untuk ditelusuri lebih lanjut. Sebagai contoh, saat menganalisis struktur berkas citra (*image file*) untuk menemukan lokasi penyimpanan nilai hash citra tersebut, Anda mungkin menemukan log dalam format ASCII yang menunjukkan bahwa target menggunakan protokol yang seharusnya dinonaktifkan dalam versi produksi. Temuan ini cukup signifikan dan layak untuk diselidiki lebih lanjut guna menentukan apakah protokol tersebut dapat dieksploitasi di kemudian hari.

Setiap langkah, baik yang berhasil maupun tidak, akan berkontribusi pada keahlian Anda secara keseluruhan. Bahkan ketika suatu jalur tertentu tidak membuahkan hasil yang diinginkan, wawasan yang diperoleh dapat menjadi bahan pertimbangan untuk upaya-upaya di masa mendatang. Sebagai contoh, memahami mekanisme keamanan yang mencegah ekstraksi *firmware* melalui satu metode tertentu dapat membantu Anda mengenali pertahanan serupa pada perangkat lain atau memberikan gagasan mengenai pendekatan serangan yang baru.

Pentingnya Pengambilan Keputusan Strategis

Pengujian penetrasi (*penetration testing*) yang efektif bergantung pada pengambilan keputusan strategis mengenai di mana upaya harus difokuskan. Hal ini mencakup aspek-aspek berikut:

- **Menilai rasio biaya-manfaat**: Mempertimbangkan potensi hasil dari suatu pendekatan dibandingkan dengan waktu dan sumber daya yang dihabiskan.
- **Menetapkan tonggak pencapaian (*milestone*) dan batasan waktu**: Menentukan tujuan dan kerangka waktu yang jelas untuk menjaga fokus serta efisiensi.
- **Mendokumentasikan temuan**: Mencatat metode, observasi, dan hasil secara mendetail, yang sangat berharga untuk menyempurnakan strategi seiring berjalannya waktu.

Pola pikir strategis ini tidak hanya meningkatkan efektivitas penugasan Anda saat ini, tetapi juga berkontribusi pada pengembangan diri Anda sebagai penguji penetrasi, sehingga Anda dapat menghadapi tantangan di masa depan dengan kepercayaan diri dan keterampilan yang lebih baik. Sebagai bagian dari proses pengambilan keputusan strategis, sering kali bermanfaat untuk melibatkan klien dalam diskusi mengenai perkembangan pengujian guna membantu menentukan prioritas dan area fokus, terutama saat menghadapi keterbatasan waktu.

2.3 RANTAI SERANGAN DAN METODE EKSPLOITASI PADA KENDARAAN

Langkah alami setelah fase penemuan (*discovery phase*) yang sukses adalah penyusunan rantai serangan (*attack chain*) yang memanfaatkan kerentanan yang ditemukan

pada fase sebelumnya. Perlu diingat bahwa fase penemuan dan fase serangan mungkin memerlukan beberapa kali iterasi sebelum tujuan serangan tercapai. Pada setiap iterasi, kerentanan tambahan akan ditemukan, yang memungkinkan serangan berlanjut lebih jauh sebelum diperlukan penemuan lebih lanjut untuk menentukan cara mengatasi hambatan berikutnya. Dalam bagian ini, kita akan berfokus pada langkah-langkah yang dilakukan selama fase serangan, dengan tetap mengingat bahwa transisi kembali ke fase penemuan adalah hal yang wajar.

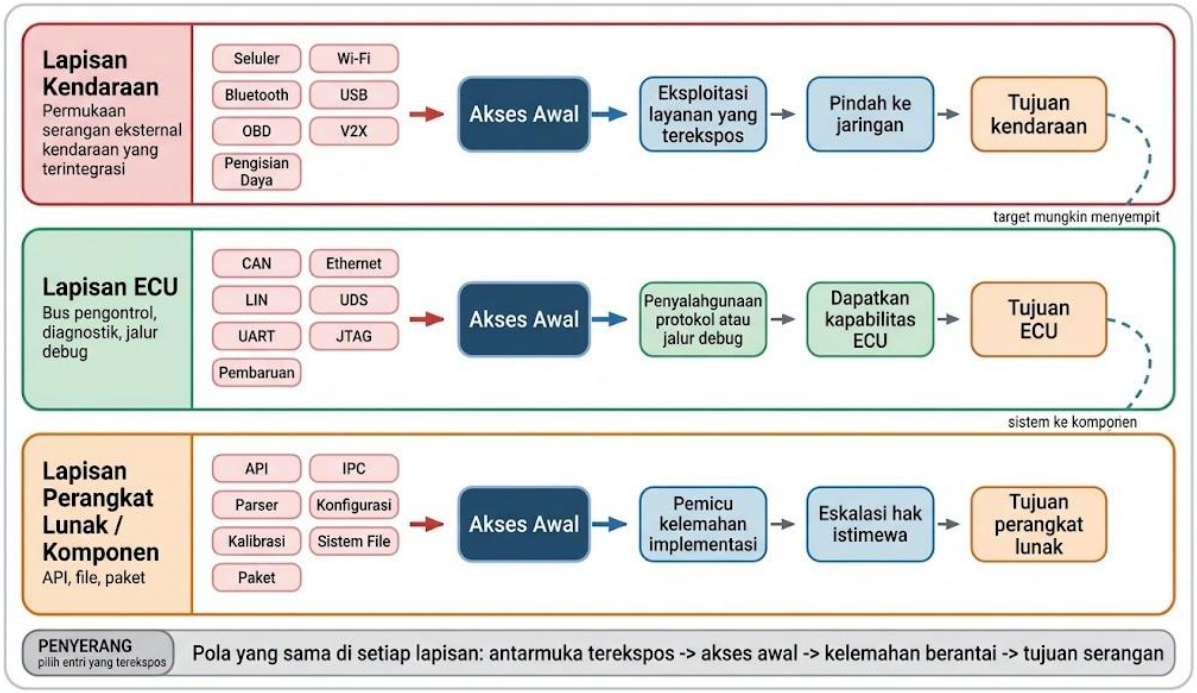
Dasar-Dasar Rantai Serangan

Rantai serangan (*attack chains*) merujuk pada langkah-langkah berurutan yang diambil oleh penyerang untuk mengeksploitasi kerentanan dalam satu atau lebih sistem guna mencapai tujuan serangan. Jarang sekali satu kelemahan saja cukup untuk mengompromikan suatu kendaraan. Sebaliknya, sering kali diperlukan serangkaian kerentanan yang saling terkait untuk mencapai keberhasilan dalam melakukan peretasan. Proses multi-langkah ini, yang dikenal sebagai rantai serangan, merupakan kunci untuk memahami bagaimana sistem kendaraan yang kompleks dapat dikompromikan, serta sangat penting bagi para profesional keamanan, baik di bidang ofensif maupun defensif.

Kemampuan untuk menyusun rangkaian serangan yang efektif merupakan hasil dari perpaduan pengalaman, penelitian, dan eksperimen selama bertahun-tahun di bidang keamanan ofensif. Tujuannya adalah menggabungkan berbagai kelemahan sedemikian rupa hingga berujung pada akses tidak sah, peningkatan hak akses (*privilege escalation*), serta manipulasi sistem-sistem krusial. Dalam dunia otomotif, hal ini bisa berkisar dari sekadar mendapatkan akses tidak sah ke perangkat komputasi berkinerja tinggi (HPC) seperti sistem hiburan dalam kendaraan (*infotainment*) hingga tindakan yang lebih berbahaya, seperti mengubah perilaku mesin atau menonaktifkan fungsi keselamatan vital, misalnya sistem pengereman.

Langkah krusial dalam membentuk rantai serangan adalah memahami arsitektur E/E (elektrik/elektronik) kendaraan, yang telah kami bahas pada bagian 1. Saat menyerang kendaraan yang terintegrasi penuh, langkah awal yang dilakukan adalah memilih titik akses awal. Titik ini biasanya berupa salah satu antarmuka yang dapat diakses dari luar, seperti Wi-Fi, Bluetooth, OBD, USB, dan sebagainya. Dalam upaya menembus satu unit kontrol elektronik (ECU), pendekatan yang sama tetap berlaku; perbedaannya terletak pada titik akses awal yang kini mengacu pada antarmuka yang terbuka pada ECU tersebut, misalnya CAN, Ethernet, JTAG, dan lain-lain. Pendekatan ini berlaku sama, baik saat menargetkan komponen perangkat lunak maupun perangkat keras, terlepas dari posisi komponen tersebut dalam hierarki kendaraan yang terintegrasi penuh.

Hal ini diilustrasikan dalam Gambar 2.3, yang menunjukkan bagaimana penyerang memilih antarmuka terbuka pada lapisan yang menjadi target mereka, kemudian merangkaikan kelemahan-kelemahan yang ada untuk mencapai tujuan serangan mereka.



Gambar 2.3: Membangun akses awal pada berbagai tingkatan hierarki arsitektur E/E

Memahami cara kerja rantai serangan (*attack chain*) sangatlah penting bagi insinyur keamanan defensif. Dengan mengidentifikasi tahapan-tahapan dalam rantai serangan—termasuk akses awal, pergerakan lateral (*lateral movement*), eskalasi hak akses, dan eksekusi—langkah-langkah pertahanan dapat diterapkan untuk menggagalkan serangan tersebut. Hal ini dicapai melalui strategi pertahanan berlapis (*defense-in-depth*), yang memastikan bahwa meskipun penyerang berhasil menembus satu lapisan keamanan, terdapat kendali tambahan yang mencegah mereka mencapai tujuan akhirnya.

Selain itu, dengan memahami bagaimana rantai serangan (*attack chains*) dapat dirancang dan digunakan, insinyur keamanan defensif dapat memperoleh wawasan yang lebih mendalam mengenai cara memodelkan ancaman serta menyusun jalur serangan (*attack paths*) selama proses TARA (*Threat Analysis and Risk Assessment*). Untuk membantu kita membangun rantai serangan yang efektif, mari kita tinjau beberapa kerangka kerja yang telah mapan untuk penyusunan jalur serangan.

Kerangka Kerja Untuk Membangun Rantai Serangan

Memanfaatkan kerangka kerja serangan yang sudah ada dapat membantu kita mengatur langkah-langkah dalam sebuah rantai serangan yang sukses melalui pendekatan yang terstruktur. Untuk itu, kita akan mengeksplorasi dua kerangka kerja utama dalam bidang ini.

Enumerasi Pola Serangan Umum (*Common Attack Pattern Enumeration*)

Common Attack Pattern Enumeration and Classification (CAPEC) dari MITRE merupakan katalog komprehensif mengenai pola-pola serangan yang menyediakan kerangka kerja terstruktur untuk memahami berbagai mekanisme yang digunakan pihak lawan (*adversaries*) dalam mengeksploitasi kerentanan. CAPEC mengorganisasikan pola-pola

serangan ke dalam kategori hierarkis mulai dari pola konseptual tingkat tinggi hingga teknik serangan teknis yang mendetail sehingga memungkinkan para profesional keamanan untuk menganalisis ancaman secara sistematis.

Mekanisme serangan dalam CAPEC mencakup spektrum yang luas, antara lain *spoofing* (penyerang menyamarkan identitas untuk mendapatkan akses tanpa izin), *flooding* (mengirimkan permintaan berlebihan hingga sistem kewalahan), dan *code injection* (mengeksploitasi kelemahan validasi input untuk menjalankan perintah arbitrer). Mekanisme lainnya meliputi *resource manipulation* (modifikasi aset atau proses tanpa izin) serta probing (pihak lawan mengumpulkan informasi untuk mengidentifikasi kerentanan). Dengan mengatalogkan mekanisme-mekanisme ini, CAPEC memfasilitasi pemodelan ancaman, penilaian kerentanan, dan pengembangan strategi pertahanan untuk berbagai lingkungan yang berbeda.

Memahami mekanisme serangan yang didefinisikan oleh CAPEC merupakan cara yang efektif untuk mengetahui bagaimana langkah-langkah serangan tertentu dapat dilakukan guna mencapai tujuan serangan. Memang, tidak semua pola serangan dalam CAPEC dapat diterapkan pada sistem otomotif; namun, terdapat cukup banyak pola yang relevan untuk konteks tersebut. Ambil contoh, CAPEC-682: Eksploitasi *Firmware* atau Kode ROM dengan Kerentanan yang Tidak Dapat Ditambal:

> *Pihak lawan dapat mengeksploitasi firmware atau kode ROM yang rentan dan tidak dapat diperbaiki (di-patch). Perangkat mungkin tidak dapat diperbaiki karena produsen—baik secara sengaja maupun tidak—merancangnya tanpa mekanisme pembaruan perangkat lunak. Bahkan jika pembaruan secara teknis dimungkinkan, suatu perangkat bisa menjadi tidak dapat diperbaiki jika produsen mengakhiri dukungan dan berhenti merilis pembaruan.*

Pola serangan (*attack pattern*) diawali dengan mendeskripsikan metode serangan, kemudian menjelaskan alur eksekusi yang umum terjadi. Pola ini juga menyajikan informasi pendukung, seperti prasyarat, tingkat keahlian yang diperlukan, konsekuensi, potensi mitigasi, contoh kasus, serta *Common Weakness Enumeration* (CWE) yang terkait.

Secara khusus untuk CAPEC-682, contoh yang diberikan menggambarkan penyerang menemukan perangkat yang menjalankan *firmware* yang tidak dapat diperbarui (*unpatchable*) dan memiliki kerentanan berupa *authentication bypass* (pengelabuan autentikasi). Penyerang memanfaatkan celah tersebut untuk memasang kode berbahaya pada perangkat. Setelah berhasil disusupi, perangkat-perangkat ini akan memindai perangkat rentan lainnya dan merekrutnya ke dalam botnet yang lebih besar, yang nantinya dapat digunakan untuk melancarkan serangan *Distributed Denial of Service* (DDoS).

Dalam konteks otomotif, pola serangan ini sangat relevan jika kode BootROM pada sebuah *System-on-Chip* (SoC) mengandung kerentanan yang memungkinkan kode berbahaya untuk dijalankan saat proses booting, atau memungkinkan mekanisme *secure boot* untuk dilewati. Untuk mengurangi kemungkinan terjadinya serangan ini, tim pengembang harus menerapkan pengendalian baik dari sisi proses maupun teknis.

Dari sisi proses, kode BootROM harus dikembangkan melalui siklus hidup pengembangan yang aman (*secure development lifecycle*), dengan praktik desain yang menjamin keamanan tinggi, tinjauan kode yang aman, serta pengujian berbasis serangan (*adversarial testing*), seperti *fuzzing* pada input berkas biner. Dari sisi teknis, perilaku BootROM dapat dirancang agar tetap memiliki kemampuan perbaikan terbatas melalui mekanisme seperti fuses atau memori *one-time-programmable* (OTP), yang memungkinkan perubahan pada keputusan tertentu di dalam BootROM jika ditemukan cacat setelah cip tersebut memasuki fase operasional.

Kerangka Kerja MITRE ATT&CK

Sumber daya lain yang berguna untuk membantu penyusunan rantai serangan (*attack chain*) yang efektif adalah kerangka kerja MITRE ATT&CK. Kerangka kerja ini merupakan basis pengetahuan terstruktur yang mengategorikan serta mendeskripsikan taktik, teknik, dan prosedur (*Tactics, Techniques, and Procedures* – TTP) yang diamati dalam serangan siber di dunia nyata. ATT&CK menyediakan peta komprehensif mengenai langkah-langkah yang diambil penyerang setelah berhasil mengakses suatu sistem, mencakup berbagai tahapan mulai dari akses awal (*initial access*) hingga eksfiltrasi data dan dampak serangan.

Kerangka kerja ini disusun berdasarkan taktik yang mewakili tujuan penyerang pada setiap tahapan dan teknik yang menjabarkan metode spesifik untuk mencapai tujuan tersebut. Dengan mendata perilaku-perilaku ini, ATT&CK menjadi perangkat yang berharga untuk memahami perilaku pihak penyerang dalam rangka menyusun rantai serangan yang efektif. Saat merancang rantai serangan (*attack chain*), Anda tidak harus menggunakan setiap taktik dan teknik yang didefinisikan dalam kerangka kerja ATT&CK. Namun, dengan memahami tahapan rantai serangan yang sedang berlangsung, Anda dapat menemukan teknik-teknik yang relevan untuk tahap tersebut.

Sebagai contoh, ketika Anda berupaya mendapatkan akses awal (*initial access*) ke dalam sistem target, terdapat sepuluh teknik yang dapat membantu Anda mencapai tujuan tersebut. Dalam konteks sistem otomotif, salah satu teknik yang cukup relevan adalah T0862, yang dikenal sebagai kompromi rantai pasok (*supply chain compromise*). Berikut adalah kutipan dari deskripsi teknik tersebut yang sangat relevan dengan sistem otomotif:

> *Pihak lawan dapat melakukan kompromi rantai pasok untuk mendapatkan akses ke lingkungan sistem kendali melalui produk, perangkat lunak, dan alur kerja yang telah terinfeksi. Kompromi rantai pasok merupakan manipulasi terhadap produk—seperti perangkat atau perangkat lunak—ataupun mekanisme penyampaiannya, sebelum produk tersebut diterima oleh konsumen akhir. Kompromi yang dilakukan pihak lawan terhadap produk dan mekanisme ini bertujuan untuk merusak data atau sistem, setelah produk yang terinfeksi masuk ke dalam lingkungan target.*
> *Perangkat palsu dapat masuk ke dalam rantai pasok global, sehingga menimbulkan risiko keamanan dan siber bagi pemilik serta operator aset. Perangkat-perangkat ini mungkin tidak memenuhi persyaratan keselamatan, teknis, maupun manufaktur yang ditetapkan oleh badan pengatur, namun bisa saja dilengkapi dengan label yang mengindikasikan kepatuhan terhadap standar industri. Akibat tidak dipenuhinya standar tersebut serta kualitas yang secara*

keseluruhan lebih rendah, produk palsu dapat menimbulkan risiko keselamatan dan operasional yang serius.

Teknik serangan ini sangat relevan dengan rantai pasok otomotif, yang memiliki karakteristik sangat terdistribusi sehingga membuka banyak celah bagi terjadinya kompromi pada rantai pasok tersebut. Salah satu contohnya adalah penyisipan sensor palsu atau perangkat jaringan berbahaya yang dapat berperilaku menyimpang dalam kondisi tertentu, sehingga memicu insiden yang membahayakan. Sebagai seorang profesional di bidang keamanan ofensif, Anda sebaiknya berupaya memanfaatkan teknik-teknik yang dapat mendukung pencapaian tujuan serangan Anda secara keseluruhan.

Taktik Dalam Konteks Otomotif

Di sepanjang buku ini, setiap kali kami menyajikan studi kasus mengenai serangan nyata, kami akan menjabarkannya menggunakan taksonomi taktik dari kerangka kerja MITRE ATT&CK, karena kerangka tersebut menyediakan struktur umum untuk mengurai rantai serangan. Berikut ini, kami akan menyoroti taktik-taktik yang paling relevan dalam konteks otomotif:

1. **Akses awal (*Initial access*)**: Teknik untuk memperoleh akses tidak sah ke dalam sistem tertanam (*embedded systems*), sering kali dengan mengeksploitasi kerentanan *firmware* yang belum diperbaiki, kredensial bawaan yang lemah, atau port komunikasi yang terbuka.
2. **Eksekusi (*Execution*)**: Metode untuk menjalankan kode berbahaya pada perangkat, seperti injeksi perintah jarak jauh atau eksploitasi mekanisme pembaruan yang tidak aman.
3. **Persistensi (*Persistence*)**: Strategi untuk mempertahankan akses pada sistem tertanam meskipun perangkat dimulai ulang (*reboot*) atau *firmware* diperbarui, termasuk dengan memodifikasi skrip saat sistem dinyalakan (*startup scripts*) atau menanamkan *firmware* berbahaya.
4. **Peningkatan hak akses (*Privilege escalation*)**: Pendekatan untuk mendapatkan izin dan hak akses yang lebih tinggi, misalnya dengan mengeksploitasi kontrol akses yang lemah dalam sistem operasi tamu (*guest operating system*).
5. **Penghindaran pertahanan (*Defense evasion*)**: Teknik untuk menghindari deteksi oleh mekanisme keamanan sistem tertanam, seperti menonaktifkan pencatatan log atau memodifikasi keluaran telemetri.
6. **Akses kredensial (*Credential access*)**: Metode untuk mendapatkan kredensial perangkat, termasuk mengekstraknya dari memori, mencegat protokol autentikasi, atau memulihkan kunci yang tersimpan.
7. **Penemuan (*Discovery*)**: Teknik untuk mengumpulkan informasi mengenai lingkungan sistem tertanam, seperti mengidentifikasi periferal yang terhubung, konfigurasi jaringan, dan layanan yang dapat diakses.

8. **Pergerakan lateral (*Lateral movement*)**: Pendekatan untuk berpindah di dalam jaringan perangkat tertanam, sering kali dengan memanfaatkan penyaringan (*filtering*) yang lemah pada gerbang pusat (*central gateway*) atau *switch Ethernet*.
9. **Pengumpulan (*Collection*)**: Teknik untuk mengumpulkan data dari perangkat, termasuk data sensor, log yang tersimpan, atau konfigurasi khusus perangkat.
10. **Perintah dan kendali (*Command and control*/C2)**: Metode untuk membangun komunikasi dengan sistem tertanam dari sistem jarak jauh, sering kali melalui protokol jaringan yang dimodifikasi atau layanan yang tidak aman.
11. **Eksfiltrasi (*Exfiltration*)**: Teknik untuk mengeluarkan data yang telah dikumpulkan dari perangkat ke lokasi eksternal, yang umumnya melibatkan saluran terenkripsi atau pemalsuan protokol (*protocol spoofing*).
12. **Dampak (*Impact*)**: Tindakan yang mengganggu fungsionalitas perangkat atau keselamatan pengguna, seperti manipulasi data, manipulasi sensor, atau penonaktifan komponen-komponen krusial.

Memahami bagaimana penyerang bergerak di seluruh arsitektur kelistrikan/elektronik (E/E) kendaraan untuk mencapai tujuan serangan mereka merupakan konsep mendasar lain yang layak untuk ditelusuri. Saat mempelajari tentang rantai serangan (*attack chains*), Anda akan sering menjumpai istilah *lateral movement* (pergerakan lateral) dan *pivoting* (pengalihan akses), yang akan kita bahas selanjutnya.

***Lateral Movement* dan *Pivoting* Dalam Jaringan Kendaraan**

Teknik *lateral movement* dan *pivoting* sangat krusial untuk menyusun rantai serangan bertahap yang mencakup berbagai sistem kendaraan. Setelah mendapatkan akses awal, penyerang sering kali menggunakan metode ini untuk menjelajahi dan menyusupi sistem lain di dalam jaringan kendaraan, meningkatkan hak akses mereka, serta menyasar aset-aset krusial seperti komputer pusat kendaraan, sistem telematika, dan platform *infotainment*.

Cara sederhana untuk membedakan *lateral movement* dengan *pivoting* adalah dengan menganggap *lateral movement* sebagai perluasan jangkauan penyerang, sementara *pivoting* merupakan salah satu cara yang digunakan penyerang untuk mewujudkan perluasan tersebut. Sebagai contoh, jika seorang penyerang berhasil menyusup ke dalam ECU sistem hiburan (*infotainment*) dan kemudian menggunakan akses tersebut untuk menjangkau layanan lain di dalam kendaraan, fungsi diagnostik, atau ECU lainnya, maka penyerang tersebut melakukan *lateral movement* (pergerakan lateral) karena mereka bergerak melampaui titik penyusupan awal. Jika penyerang menggunakan ECU sistem hiburan yang telah disusupi tersebut sebagai jembatan atau perantara untuk mengirim lalu lintas ke segmen jaringan kendaraan yang sebelumnya tidak dapat diakses langsung dari luar kendaraan, maka ECU sistem hiburan itu berfungsi sebagai *pivot point* (titik tumpu). Dengan demikian, *lateral movement* menggambarkan perkembangan serangan yang lebih luas dari satu sistem ke sistem lainnya, sementara *pivoting* menggambarkan penggunaan komponen yang telah disusupi sebagai batu loncatan yang memungkinkan penyerang menjangkau sistem-sistem tambahan tersebut.

Rantai Serangan Pada Tingkat Kendaraan vs. Tingkat ECU

Logika rantai serangan yang sama berlaku baik jika targetnya adalah kendaraan secara keseluruhan maupun satu ECU tertentu. Perbedaannya terletak pada cakupan pergerakan penyerang dan aset yang diincar. Pada tingkat kendaraan, penyerang biasanya memulai melalui antarmuka yang dapat diakses dari luar, seperti Bluetooth, Wi-Fi, USB, konektivitas seluler, atau port OBD. Setelah mendapatkan akses awal, penyerang menjalankan kode atau menyalahgunakan antarmuka yang rentan tersebut untuk membangun pijakan. Dari sana, penyerang mungkin berupaya meningkatkan hak akses (*privilege escalation*), membangun persistensi, dan bergerak secara lateral ke ECU lain yang memiliki fungsi lebih berharga.

Dalam serangan yang melibatkan banyak ECU, *lateral movement* menjadi sangat penting karena titik penyusupan awal penyerang sering kali bukanlah target akhir. Sebagai contoh, penyusupan pada ECU sistem hiburan mungkin hanyalah langkah awal untuk mencapai ECU yang berkaitan dengan aspek keselamatan (*safety-relevant ECU*). Jika segmentasi jaringan atau *gateway* pusat membatasi akses langsung, ECU yang telah disusupi tersebut mungkin perlu berfungsi sebagai *pivot point* untuk masuk ke domain lain. Dengan cara ini, rantai serangan pada tingkat kendaraan paling tepat dipahami sebagai perkembangan yang melintasi batasan kepercayaan (*trust boundaries*), segmen jaringan, dan tingkat hak akses, hingga penyerang mencapai fungsi atau aset yang dituju.

Pada tingkat ECU, urutan yang sama muncul dalam bentuk yang lebih terlokalisasi. Bayangkan sebuah platform komputasi berkinerja tinggi yang menjalankan beberapa mesin virtual atau proses yang terisolasi. Seorang penyerang mungkin terlebih dahulu mendapatkan eksekusi kode di dalam satu lingkungan tamu (*guest environment*), kemudian membangun persistensi dengan memodifikasi sistem berkas, lalu meningkatkan hak akses (*privilege escalation*) ke tingkat kernel tamu atau lapisan hipervisor. Jika batasan isolasi dapat dilampaui, penyerang dapat bergerak secara lateral dari satu mesin virtual atau batasan proses ke yang lain hingga mencapai fungsi target. Dengan demikian, pada tingkat ECU, pergerakan lateral dan peningkatan hak akses dapat terjadi di dalam arsitektur internal satu perangkat, alih-alih melintasi beberapa ECU fisik yang berbeda.

Jika dilihat secara keseluruhan, kedua perspektif ini mempertegas satu poin penting: rantai serangan tidak bergantung pada skala. Baik saat penyerang bergerak dari satu ECU ke ECU lain melalui jaringan kendaraan, maupun dari satu konteks eksekusi ke konteks lainnya di dalam satu ECU, logika dasarnya tetap sama: memperoleh akses, memperluas kendali, mengatasi batasan isolasi, dan terus melangkah menuju tujuan serangan.

Metode Serangan

Setelah penyerang mengidentifikasi kerentanan, antarmuka, atau kontrol keamanan yang lemah, langkah selanjutnya adalah memilih metode yang dapat memajukan rantai serangan (*attack chain*). Metode-metode ini bukanlah bagian terpisah dari rantai serangan, melainkan teknik konkret yang digunakan untuk mencapai eksekusi kode, peningkatan hak akses (*privilege escalation*), persistensi, pergerakan lateral (*lateral movement*), atau dampak terhadap keselamatan.

Beberapa metode serangan menargetkan integritas perangkat tegar (*firmware*) dan perangkat lunak. Ini mencakup manipulasi citra perangkat lunak, penyalahgunaan mekanisme pembaruan yang tidak aman, injeksi kode, serta eksploitasi kelemahan korupsi memori. Metode lainnya berfokus pada status sistem dan hak akses, seperti mengeksploitasi *race condition*, kerentanan *time-of-check to time-of-use* (TOCTOU), kontrol akses yang lemah, atau konfigurasi yang tidak aman untuk mendapatkan hak akses yang lebih tinggi atau melewati pemeriksaan keamanan.

Metode serangan lain beroperasi pada lapisan jaringan dan protokol. Ini termasuk serangan *man-in-the-middle*, penolakan layanan (*denial-of-service*), serangan *replay*, *spoofing*, serta enumerasi port atau layanan. Metode semacam ini sangat krusial karena banyak unit kontrol elektronik (ECU) bergantung pada alur pesan tepercaya, asumsi waktu (*timing assumptions*), dan pola komunikasi internal yang longgar. Penyerang juga dapat menargetkan mekanisme keamanan pendukung alih-alih logika aplikasi secara langsung. Kelemahan kriptografis seperti manajemen kunci yang buruk, vektor inisialisasi yang tetap, atau tingkat keacakan yang tidak memadai dapat memungkinkan pemalsuan pesan, dekripsi, atau penyamaran identitas.

Demikian pula, serangan berbasis lingkungan dan sensor, termasuk manipulasi sinyal dan injeksi kesalahan (*fault injection*), dapat mengubah cara sistem mempersepsikan dunia fisik, yang berpotensi menciptakan kondisi yang tidak aman atau dapat dieksploitasi. Terakhir, beberapa metode serangan berasal dari luar kendaraan itu sendiri. Serangan rantai pasok (*supply-chain attacks*), komponen palsu, serta perangkat atau subsistem yang dimodifikasi secara berbahaya dapat menghadirkan kerentanan bahkan sebelum kendaraan tersebut digunakan. Bagi tim keamanan ofensif, menguasai metode-metode ini merupakan keharusan untuk dapat menyusun rantai serangan efektif yang menggabungkan beberapa teknik tersebut. Sebuah serangan nyata mungkin dimulai dengan kelemahan protokol, menggunakan eksploitasi perangkat lunak untuk eksekusi kode, mengandalkan peningkatan hak akses untuk memperdalam kendali, lalu menyalahgunakan hubungan kepercayaan dalam jaringan untuk melakukan pergerakan lateral. Oleh karena itu, metode serangan paling tepat dipahami sebagai perangkat (*toolkit*) yang mendukung setiap tahapan dalam rantai serangan yang lebih luas. Mulai bab 4 hingga 9, kita akan mendalami berbagai metode serangan tersebut serta kelemahan keamanan yang dieksploitasinya, sembari menelaah aspek keamanan pada berbagai lapisan arsitektur E/E.

2.4 PELAPORAN DAN TINDAK LANJUT KEAMANAN OTOMOTIF

Komunikasi yang efektif dan pelaporan yang cermat bukan sekadar formalitas prosedural, melainkan elemen fundamental bagi keberhasilan pengujian penetrasi (*penetration test*) di sektor otomotif. Komunikasi yang jelas menyelaraskan para pemangku kepentingan, memungkinkan respons cepat terhadap temuan kritis, serta menjaga keamanan dan integritas proses pengujian. Laporan yang disusun dengan baik menyediakan informasi yang dapat ditindaklanjuti, yang diperlukan organisasi untuk memprioritaskan risiko keamanan dan menerapkan mitigasi yang tepat, sehingga meningkatkan perlindungan terhadap sistem

kendaraan yang terhubung (*connected vehicle systems*). Bagian ini membahas elemen-elemen kunci komunikasi selama pengujian penetrasi, termasuk cara menyusun laporan tertulis dan menjaga alur komunikasi yang jelas dengan seluruh pemangku kepentingan sepanjang proses pengujian.

Pentingnya Komunikasi Selama Pengujian Penetrasi

Dengan terus memberikan informasi kepada para pemangku kepentingan, kita memastikan bahwa semua pihak memiliki pemahaman yang sama mengenai tujuan pengujian dan masalah apa pun yang mungkin muncul. Komunikasi yang sigap sangatlah krusial, terutama ketika ditemukan kerentanan yang signifikan, sehingga mitigasi dapat segera dilakukan untuk melindungi sistem kendaraan. Dalam pengujian penetrasi otomotif, penting untuk berinteraksi dengan beberapa kontak utama di pihak klien:

- **Kontak utama (*Primary contact*)**: Biasanya merupakan pemimpin atau manajer proyek yang bertanggung jawab mengawasi keseluruhan pelaksanaan pengujian. Individu ini berfokus pada dampak bisnis dan tata kelola, serta memastikan bahwa pengujian selaras dengan tujuan organisasi seperti kepatuhan terhadap standar keamanan siber dan pemenuhan komitmen kepada pelanggan.
- **Kontak teknis (*Technical contact*)**: Menangani aspek teknis dari pengujian, memastikan bahwa pertanyaan teknis mengenai target diarahkan kepada pihak yang tepat di dalam organisasi guna menghindari penundaan yang tidak perlu. Individu ini biasanya bertugas mengumpulkan lembar data (*data sheets*), kode sumber (*source code*), dokumen desain, binari perangkat lunak, dan informasi pengaturan target yang diperlukan oleh penguji penetrasi untuk melaksanakan tugas mereka. Orang ini harus memahami ketentuan pengujian agar tidak membagikan informasi di luar batasan yang telah ditetapkan.

Pemicu Komunikasi

Selama proses pengujian berlangsung, dapat muncul peristiwa tertentu yang mengharuskan komunikasi segera dengan kontak utama atau kontak teknis. Salah satu contohnya adalah ditemukannya kerentanan kritis pada produk yang sudah dipasarkan atau produk yang hampir memasuki tahap produksi. Penguji penetrasi harus mengevaluasi peristiwa tersebut untuk menentukan apakah pengujian perlu dihentikan sementara guna memberi tahu para pemangku kepentingan, atau apakah sebaiknya menunggu hingga seluruh tujuan pengujian tercapai.

Misalkan Anda ditugaskan untuk mengevaluasi keamanan kode *bootROM* pada cip otomotif guna mengidentifikasi kerentanan yang dapat menggagalkan proses *secure boot*. Saat melakukan pengujian, Anda menemukan *backdoor* yang dapat menonaktifkan *secure boot* melalui prosedur tertentu yang dapat diakses. Sebagai bagian dari kesepakatan kerja, Anda telah diberi tahu mengenai tahapan-tahapan proyek (*milestones*) sehingga Anda memahami posisi pengujian penetrasi (*pentest*) dalam jadwal proyek tersebut. Cip tersebut sudah mendekati tahap *tape-out* (siap untuk diproduksi), dan berdasarkan pengetahuan Anda tentang jadwal proyek, Anda menyadari bahwa menunda pengungkapan kerentanan ini akan membuat vendor cip tidak memiliki cukup waktu untuk melakukan perbaikan sebelum

perubahan lebih lanjut tidak lagi dimungkinkan. Dalam situasi ini, Anda harus memprioritaskan pengungkapan temuan tersebut agar risiko dapat segera ditangani.

Selain temuan kritis seperti itu, pembaruan rutin misalnya dalam bentuk ringkasan mingguan dapat memberikan wawasan mengenai tugas yang telah diselesaikan, rencana mendatang, serta kendala apa pun yang dihadapi. Menjaga agar semua pihak tetap terinformasi dan terlibat dalam proses pengujian akan sangat bermanfaat, terutama saat menghadapi hambatan, seperti kerusakan perangkat atau ditemukannya anomali yang tidak terduga.

Menyusun dan Menyajikan Temuan

Setelah fase pengujian selesai, semua temuan harus disusun ke dalam laporan yang komprehensif dan terstruktur. Laporan ini harus memberikan wawasan yang dapat ditindaklanjuti untuk memandu upaya perbaikan serta meningkatkan postur keamanan organisasi. Berikut adalah komponen-komponen dalam laporan tersebut:

- **Ringkasan eksekutif (*Executive summary*)**: Gambaran tingkat tinggi yang ditujukan bagi manajemen, merangkum temuan utama dan rekomendasi. Bagian ini harus menjelaskan dengan jelas dampak bisnis dari kerentanan yang ditemukan, serta menekankan signifikansinya dalam konteks tujuan dan selera risiko (*risk appetite*) organisasi.
- **Metodologi**: Penjelasan terperinci mengenai standar dan kerangka kerja yang diikuti selama pengujian. Sebagai contoh, Anda dapat merujuk pada kelas ancaman yang disyaratkan oleh UNECE R155, atau mencantumkan standar FIPS yang digunakan dalam memverifikasi modul kriptografi.
- **Temuan**: Bagian inti dari laporan yang mencantumkan kerentanan yang ditemukan, beserta tingkat keparahannya, potensi dampaknya, dan rekomendasi untuk perbaikan. Setiap temuan harus mencakup hal-hal berikut:
    - **Peringkat risiko (*Risk rating*)**: Menetapkan tingkat risiko berdasarkan faktor-faktor seperti kemudahan eksploitasi dan potensi dampak, dengan mengikuti kerangka kerja yang telah ditetapkan, seperti tingkat risiko menurut ISO/SAE 21434 atau skor CVSS.
    - **Tingkat ancaman**: Menilai ancaman keseluruhan yang ditimbulkan oleh kerentanan tersebut, dengan mempertimbangkan kemungkinan eksploitasi.
    - **Skenario kerusakan**: Uraikan kemungkinan dampak buruk jika kerentanan tersebut dieksploitasi, guna membantu para pemangku kepentingan memahami implikasi di dunia nyata.
    - **Detail eksploitasi**: Berikan rincian teknis mengenai bagaimana kerentanan dapat dieksploitasi, termasuk contoh bukti konsep (*proof-of-concept*) atau jalur serangan yang digunakan selama pengujian.

Pengamanan dan Penyimpanan Laporan

Mengingat sifat sensitif dari informasi yang terkandung dalam laporan pengujian penetrasi (*penetration testing*), menjamin keamanannya adalah hal yang sangat penting. Pertama, metode penyampaian laporan harus disetujui oleh pihak penerima, misalnya melalui

portal perusahaan yang aman. Hal ini bertujuan untuk mencegah intersepsi laporan selama proses pengiriman serta kebocoran kerentanan yang ditemukan kepada publik.

Kedua, laporan tersebut harus ditandai sebagai rahasia atau sangat rahasia dengan menggunakan penanda (*watermark*) yang sesuai, guna menghindari penyebaran yang tidak disengaja kepada pihak yang tidak berwenang. Praktik yang baik adalah dengan menyertakan watermark berupa nama dan alamat email penerima pada laporan tersebut. Langkah ini menegaskan kepada penerima bahwa mereka bertanggung jawab untuk tidak membagikan laporan tersebut kepada pihak lain; jika terjadi kebocoran, maka sumber kebocoran dapat dilacak kembali kepada mereka. Saat disimpan oleh organisasi penguji penetrasi, laporan harus dilindungi dengan kontrol akses yang ketat dan metode enkripsi. Idealnya, laporan tersebut dihapus setelah jangka waktu tertentu untuk mencegah kebocoran yang tidak disengaja di masa mendatang, misalnya dalam insiden keamanan siber.

Pencatatan terperinci mengenai siapa yang mengakses laporan dan kapan akses tersebut dilakukan akan mendukung akuntabilitas, sehingga setiap upaya akses yang tidak sah dapat dilacak dan diselidiki dengan segera. Sebagai penerima laporan, tanggung jawab Anda setelah memahami hasilnya adalah melakukan triase terhadap kerentanan yang dilaporkan serta mengonfirmasi penilaian tingkat keparahannya. Jika Anda mengembangkan produk otomotif dengan mengacu pada sistem manajemen keamanan siber, Anda diharapkan mengikuti proses *Product Security Incident Response Team* (PSIRT) untuk menerima dan mengelola laporan kerentanan tersebut dengan tepat. Mengikuti proses PSIRT memungkinkan pelacakan kerentanan mulai dari identifikasi hingga penyelesaian akhir sehingga upaya perbaikan dapat dikelola secara efektif.

Aktivitas Pasca-Perbaikan: Memastikan Keamanan yang Berkelanjutan

Identifikasi dan perbaikan kerentanan sangat krusial untuk menjamin keselamatan dan keamanan kendaraan. Penguji penetrasi diharapkan tidak hanya memberikan informasi mengenai kerentanan, tetapi juga memberikan rekomendasi untuk langkah-langkah perbaikan. Mereka mungkin mengusulkan perbaikan kode, penambahan kontrol keamanan, atau perubahan desain arsitektur. Dalam semua kasus, rekomendasi-rekomendasi ini harus dievaluasi oleh pihak penerima untuk menentukan prioritas temuan mana yang perlu diperbaiki terlebih dahulu, serta bagaimana memvalidasi bahwa perbaikan tersebut efektif dalam mengurangi risiko eksploitasi. Kegiatan utama pasca-remediasi meliputi hal-hal berikut:

- Pengujian ulang (*revalidation testing*): Melakukan pengujian tindak lanjut untuk memastikan bahwa perbaikan yang diterapkan telah mengatasi kerentanan yang ada. Hal ini dapat mencakup upaya untuk melakukan eksploitasi kembali guna memastikan bahwa metode serangan tersebut tidak lagi efektif. Dalam tahap ini, mungkin diperlukan dukungan dari pihak penguji penetrasi (*penetration tester*) untuk menciptakan kembali kondisi serangan. Sebagai contoh, jika serangan tersebut memerlukan peralatan khusus seperti perangkat untuk fault injection atau probe untuk analisis *side-channel* maka mengandalkan pihak penguji penetrasi untuk mengevaluasi efektivitas perbaikan mungkin menjadi satu-satunya opsi yang layak agar masalah tersebut dapat dianggap terselesaikan.

- Analisis akar penyebab (*root cause analysis*): Setelah suatu kerentanan ditemukan, analisis akar penyebab harus dilakukan untuk mengidentifikasi kekurangan dalam proses maupun kelemahan teknis yang menyebabkan kerentanan tersebut muncul sejak awal. Hal ini bisa berupa kekurangan dalam praktik penulisan kode yang aman (*secure coding*) atau kelemahan dalam proses pemodelan ancaman (*threat modeling*). Apa pun penyebabnya, penting bagi organisasi untuk memahami akar masalah tersebut dan menyusun rencana untuk menanganinya pada produk-produk mendatang. Sering kali, langkah ini berujung pada pembaruan materi pelatihan untuk memastikan bahwa pelajaran yang dipetik dapat disebarluaskan ke seluruh organisasi, bukan hanya kepada tim yang mengalami masalah tersebut.
- Pembaruan pengujian keamanan: Salah satu pelajaran penting dari proses remediasi kerentanan adalah bahwa jika suatu masalah terjadi sekali, ada kemungkinan kesalahan yang sama akan terulang di masa depan. Selain analisis akar penyebab dan kegiatan remediasi standar, ada baiknya untuk mengevaluasi apakah prosedur pengujian keamanan dapat diperbarui guna mendeteksi masalah serupa di masa mendatang, sebelum masalah tersebut mencapai produk akhir. Dengan menetapkan kasus uji (*test case*) baru yang mampu menangkap bug tersebut, organisasi dapat memastikan bahwa jika bug yang sama muncul kembali di kemudian hari, hal itu tidak akan luput dari deteksi.

2.5 TAHAPAN DAN STRATEGI PENGUJIAN PENETRASI KEAMANAN OTOMOTIF

Sepanjang bab ini, kita telah melihat bahwa pengujian penetrasi dalam industri otomotif bukan sekadar pemenuhan persyaratan regulasi atau formalitas prosedural semata. Sebaliknya, pengujian ini merupakan proses krusial yang menjadi bagian integral dalam menjamin keamanan dan keselamatan kendaraan modern. Kita telah melihat bagaimana setiap fase dalam proses pengujian penetrasi mendukung tujuan utama, yaitu mengidentifikasi dan memitigasi kerentanan sebelum pihak lawan (penyerang) dapat memanfaatkannya.

Dengan mengawali proses melalui perencanaan yang matang yang mempertimbangkan ruang lingkup, persyaratan hukum, serta manajemen risiko secara cermat para penguji dapat memprioritaskan aspek-aspek sistem yang tepat untuk dievaluasi. Terlebih lagi, menetapkan tujuan dan parameter yang jelas tidak hanya memandu proses pengujian, tetapi juga membangun pemahaman bersama di antara semua pemangku kepentingan guna menghindari ketidakselarasan selama proses tersebut berlangsung.

Selama fase penemuan (*discovery*), kami mengkaji bagaimana penggunaan teknik pengintaian pasif maupun aktif memungkinkan penguji untuk memetakan permukaan serangan (*attack surface*) kendaraan secara komprehensif, guna memaksimalkan peluang ditemukannya kelemahan keamanan. Dengan mengumpulkan informasi secara cermat, penguji dapat mengidentifikasi titik masuk potensial dan celah kelemahan yang mungkin luput dari perhatian. Selanjutnya, kami beralih ke fase serangan (*attack*) yang memanfaatkan informasi dari fase penemuan, sehingga penguji dapat mengeksploitasi kerentanan yang telah

diidentifikasi secara terkendali dan berulang. Melalui simulasi skenario serangan dunia nyata, fase ini memberikan wawasan berharga mengenai bagaimana pihak lawan dapat menyusup ke dalam sistem, sekaligus menekankan pentingnya deteksi proaktif dan perbaikan kerentanan. Bukti keberhasilan eksploitasi membantu organisasi memahami tingkat kelayakan temuan tersebut serta dampak keputusan keamanan mereka dalam mencegah masalah serupa di masa depan.

Terakhir, kami membahas fase pelaporan dan tindakan tindak lanjut. Laporan yang komprehensif tidak hanya menyoroti kerentanan yang ditemukan, tetapi juga menyajikan langkah-langkah perbaikan yang dapat ditindaklanjuti. Dengan memberikan panduan praktis serta memprioritaskan risiko berdasarkan tingkat keparahan dan dampak potensialnya, laporan ini membantu produsen kendaraan (OEM) dan pemasok dalam memperkuat pertahanan mereka secara proaktif.

Kami juga menyoroti pentingnya mengaitkan kegiatan pengujian penetrasi (*penetration testing*) dengan proses PSIRT (*Product Security Incident Response Team*) untuk memantau kerentanan hingga tuntas diperbaiki. Selain itu, melakukan analisis akar masalah (*root cause analysis*) yang tepat sangatlah penting untuk memperkuat langkah-langkah proses maupun kontrol teknis, guna mengurangi kemungkinan munculnya kembali masalah yang sama di masa mendatang.

Dalam bab ini, kami sengaja tidak membahas secara mendalam mengenai perangkat (*tools*) yang digunakan pada setiap fase pengujian penetrasi, karena fokus utama kami adalah memperkenalkan konsep dan teknik mendasar dalam tahapan pengujian penetrasi tersebut. Pada bab berikutnya, kami akan mengalihkan fokus pada berbagai perangkat yang tersedia untuk melaksanakan metode penemuan dan serangan yang telah diperkenalkan dalam bab ini.

BAB 3
MEMBANGUN KUMPULAN PERANGKAT (TOOL ARSENAL)

Dalam dua bab pertama, kami telah menyajikan cetak biru untuk melaksanakan kegiatan keamanan ofensif. Pada bab ini, fokus kita adalah pada perangkat-perangkat yang memungkinkan peluncuran serangan terhadap berbagai lapisan arsitektur kendaraan. Anda dapat menganggap bab ini sebagai peta perangkat keamanan ofensif pilihan, yang dapat dijadikan referensi saat Anda membaca bab 4 hingga 9.

Untuk membantu Anda membangun kumpulan perangkat keamanan ofensif, kami akan meninjau berbagai perangkat yang mencakup setiap aspek arsitektur kendaraan mulai dari cloud, lapisan jaringan nirkabel, jaringan di dalam kendaraan (*in-vehicle network*), ECU berbasis MCU dan SoC, hingga komponen sensor serta AI/ML. Mengingat AI telah mengubah lanskap keamanan ofensif, di mana pun memungkinkan, kami akan menunjukkan perangkat yang didukung atau digerakkan oleh AI, yang dapat melengkapi atau menggantikan perangkat konvensional.

Di akhir bab ini, Anda diharapkan memiliki pemahaman yang cukup baik mengenai perangkat yang diperlukan untuk menjalankan kampanye pengujian keamanan ofensif, serta cara memulai peretasan kendaraan secara etis. Perlu dicatat bahwa penekanan di sini adalah pada perangkat dan teknik yang digunakan, bukan secara spesifik pada penyusunan rantai serangan (*attack chains*) yang memanfaatkan perangkat tersebut. Sepanjang bab ini, perangkat-perangkat yang dibahas akan dikategorikan berdasarkan dua fase utama:

1. ***Discovery* (pengintaian dan analisis)**: membuat target dapat diamati, melakukan enumerasi antarmuka, memetakan layanan, mengekstrak *firmware*, dan menyusun hipotesis yang akurat.
2. ***Exploitation and validation* (eksploitasi dan validasi)**: manipulasi terkendali untuk mengonfirmasi kelemahan, mengukur dampak, serta menghasilkan bukti pengujian dan artefak yang dapat direproduksi.

Pada bagian selanjutnya dari bab ini, kami akan mengelompokkan perangkat-perangkat tersebut berdasarkan lapisan arsitektur kendaraan berikut:

- Perangkat untuk *backend* kendaraan
- Perangkat untuk antarmuka nirkabel
- Perangkat untuk jaringan di dalam kendaraan (*in-vehicle network*)
- Perangkat ECU berbasis MCU dengan performa tinggi
- Perangkat untuk sensor kendaraan
- Bagaimana AI mengubah pengujian penetrasi (*penetration testing*)

3.1 PERANGKAT UNTUK SISTEM BACKEND KENDARAAN

Kendaraan terhubung (*connected vehicles*), sebagaimana akan dibahas pada bab 4, bergantung pada *server backend* dan *platform cloud* (seperti AWS IoT Core, Azure IoT Hub)

untuk telematika, pengumpulan data, fitur kendali jarak jauh, dan manajemen armada. Penguji penetrasi yang menargetkan *backend* kendaraan harus mengevaluasi API cloud, jalur komunikasi antara kendaraan dan cloud, serta konfigurasi infrastruktur cloud. Perangkat di kategori ini membantu mencegat dan menganalisis komunikasi *backend* (tahap penemuan/ *discovery*), mengaudit konfigurasi cloud, dan mengeksploitasi kerentanan di sisi cloud. Pada bagian ini, kita akan berfokus pada perangkat yang dikelompokkan berdasarkan kategori penilaian untuk sistem cloud dan *backend*.

Penemuan Permukaan Serangan Eksternal dan Validasi Eksposur

Pemindai berskala internet sering digunakan dalam penilaian *backend* otomotif ketika organisasi perlu memastikan aset apa saja yang terekspos secara tidak sengaja. Meskipun bukan perangkat khusus otomotif, alat-alat ini lazim digunakan untuk mendeteksi broker MQTT, endpoint API, atau layanan publik dengan konfigurasi yang salah, yang dapat menjadi risiko bagi ekosistem kendaraan:

- **Shodan dan Censys** berfungsi sebagai mesin pencari untuk perangkat yang terhubung ke internet. Penguji penetrasi menggunakan alat ini untuk menemukan endpoint telematika kendaraan atau layanan cloud yang terekspos. Sebagai contoh, broker MQTT atau endpoint API dengan konfigurasi yang keliru dapat ditemukan melalui Shodan dengan mencari banner unik. Alat-alat ini membantu dalam memetakan permukaan serangan selama fase pengintaian (*reconnaissance*).
- **Amass** melakukan enumerasi terhadap permukaan serangan infrastruktur yang menghadap ke internet melalui enumerasi DNS, analisis log transparansi sertifikat, dan pencarian ASN, serta memetakan subdomain dan rentang IP untuk organisasi target. Penguji penetrasi dapat menjalankan Amass dalam mode pasif terhadap domain *backend* kendaraan untuk menemukan endpoint API cloud, server pembaruan OTA (*Over-the-Air*), dan gerbang telematika yang tidak terlihat melalui penelusuran web biasa. Hasilnya berupa grafik hubungan antar-host yang dapat memandu pemindaian lebih lanjut menggunakan Shodan dan Censys.
- **SecurityTrails** menyediakan basis data riwayat DNS dan OSINT untuk mengambil catatan DNS historis, daftar subdomain, data WHOIS, serta asosiasi alamat IP untuk domain target. Seorang penguji dapat menggunakan SecurityTrails untuk domain *backend* kendaraan OEM guna menemukan lingkungan staging, layanan internal yang terekspos melalui DNS, serta infrastruktur yang ada sebelum penerapan sistem produksi saat ini sehingga memperluas cakupan permukaan serangan melampaui apa yang terlihat dari catatan DNS saat ini.
- **masscan** adalah pemindai port TCP *stateless* (tanpa status) yang dirancang untuk pemindaian berkecepatan tinggi pada rentang alamat IP yang luas. Penguji keamanan dapat menggunakan masscan untuk mengidentifikasi semua port TCP yang terbuka dengan cepat di seluruh rentang IP yang ditetapkan untuk infrastruktur cloud OEM kendaraan atau jaringan privat APN, sehingga menghasilkan peta layanan yang lengkap sebelum melakukan enumerasi terarah menggunakan Nmap.

- **GitLeaks** memindai repositori Git dan riwayat commit untuk mencari string dengan entropi tinggi serta pola yang menyerupai kunci API, kata sandi, dan sertifikat di seluruh *branch* (cabang) dan tag. Seorang pentester dapat menjalankan GitLeaks pada repositori kode sumber OEM untuk menemukan rahasia yang pernah di-commit namun tidak dihapus termasuk kunci API aktif untuk layanan telematika kendaraan yang masih dapat dieksploitasi meskipun telah ditimpa (*overwritten*) pada commit terbaru.
- **truffleHog** menelusuri riwayat commit Git dan konten berkas untuk mencari string dengan entropi tinggi atau pola yang sesuai dengan kredensial yang diketahui, mencakup branch, tag, serta perubahan yang disimpan sementara (*stashed changes*). Penguji dapat menggunakan truffleHog bersama dengan GitLeaks untuk mendeteksi kredensial yang sempat ditambahkan lalu ditimpa pada commit berikutnya, namun tetap dapat diakses di dalam git object store sebuah kelalaian yang sering terjadi dalam alur kerja pengembangan OEM.
- **GitGuardian** memantau repositori publik maupun privat secara real-time untuk mendeteksi rahasia yang terekspos, serta memberikan peringatan kepada tim keamanan jika ditemukan kredensial di dalam kode sumber. Dalam konteks pengujian penetrasi (*penetration test*), platform ini dapat menunjukkan kepada pihak OEM bahwa kredensial mereka sedang atau pernah terekspos ke publik, yang memperkuat temuan bahwa praktik pengelolaan rahasia (*secret management*) memerlukan perbaikan.

Aplikasi Pendamping Kendaraan dan API Backend

Backend kendaraan sering kali diakses melalui aplikasi seluler. Perangkat yang digunakan serupa dengan yang dipakai dalam pengujian API seluler/web standar:

- **Jadx** melakukan dekompilasi file APK Android menjadi kode sumber Java, termasuk manifes aplikasi, file sumber daya, dan semua definisi kelas. Seorang penguji keamanan dapat menggunakan Jadx untuk melakukan rekayasa balik (*reverse engineering*) pada aplikasi pendamping kendaraan dari produsen (OEM), guna menemukan URL dasar API, logika autentikasi, detail implementasi certificate pinning, serta kredensial yang tertanam secara permanen (*hardcoded*) di dalam kode. Saat ini, Jadx merupakan standar dekompiler Android yang menggantikan JD-GUI untuk sebagian besar pekerjaan yang melibatkan file APK.
- **MobSF (*Mobile Security Framework*)** melakukan analisis statis dan dinamis secara otomatis terhadap file biner aplikasi Android maupun iOS, serta dijalankan sebagai aplikasi web lokal. Seorang penguji penetrasi (*pentester*) dapat memasukkan file APK atau IPA dari aplikasi pendamping kendaraan OEM ke dalam MobSF untuk mengekstrak endpoint API, token autentikasi, kunci kriptografi, dan alamat server *backend* yang tertanam di dalam kode aplikasi. Hal ini memberikan target langsung untuk pengujian *backend* tahap selanjutnya, tanpa memerlukan akses ke kode sumber aplikasi tersebut.
- **Magisk** adalah kerangka kerja root Android yang bersifat *systemless* (tanpa memodifikasi partisi sistem). Kerangka kerja ini memungkinkan akses root pada

perangkat tanpa mengubah partisi sistem, sehingga mampu melewati berbagai pemeriksaan deteksi root yang diterapkan oleh aplikasi pendamping dari produsen perangkat (OEM). Setelah perangkat di-root menggunakan Magisk, penguji dapat memanfaatkan alat seperti Frida untuk menyisipkan kode ke dalam proses aplikasi saat sedang berjalan (*runtime*), memintas certificate pinning, serta memeriksa panggilan API dan alur autentikasi yang tidak dapat diamati tanpa akses root.

- **OpenSSL s_client** adalah klien *Transport Layer Security* (TLS) berbasis baris perintah yang berfungsi untuk memulai jabat tangan (*handshake*) dengan server jarak jauh. Alat ini akan melaporkan parameter sesi yang telah disepakati secara lengkap, termasuk informasi mengenai apakah server mengajukan permintaan sertifikat kepada klien. Seorang penguji penetrasi (*pentester*) dapat menggunakan alat ini terhadap API *backend* kendaraan atau titik akhir (*endpoint*) pembaruan OTA untuk menentukan apakah server menerapkan mutual TLS (mTLS). Penguji dapat mencoba melakukan koneksi tanpa menyertakan sertifikat klien untuk melihat apakah *backend* tetap menerima sesi tersebut dan memproses perintah tanpa memvalidasi identitas pihak yang melakukan panggilan.

Pencegatan, Pemeriksaan, dan Rekonstruksi Alur Kerja API

Selama fase penemuan (*discovery*), penguji sering kali menangkap lalu lintas antara komponen kendaraan yang terekspos secara eksternal (seperti aplikasi seluler) dan layanan *backend*. *Intercepting proxy* (proksi pencegat) merupakan salah satu perangkat pertama yang digunakan karena memungkinkan penguji untuk memahami serta memanipulasi kontrak permintaan/tanggapan (*request/response*) antara aplikasi pendamping, head unit, dan layanan cloud:

- **Fiddler** dapat mencegat pesan kontrol dari aplikasi ke cloud, sehingga penguji dapat memeriksa token autentikasi dan enkripsi. Perangkat ini memfasilitasi pengulangan (*replay*) atau modifikasi permintaan untuk menguji validasi input (eksploitasi API cloud).
- **Burp Suite** menyediakan kemampuan serupa dengan antarmuka yang dapat diperluas untuk *fuzzing* layanan web. Dokumentasi Burp Suite menjelaskan bahwa Burp AI merupakan fitur berbasis kecerdasan buatan yang terintegrasi ke dalam Burp Suite untuk menemukan kerentanan secara lebih efisien, memahami teknologi web yang kompleks, serta menyederhanakan pengaturan autentikasi; dokumentasi tersebut juga menegaskan bahwa fitur AI hanya akan berjalan jika diaktifkan secara eksplisit.
- **mitmproxy** menawarkan alur kerja serupa dalam bentuk perangkat sumber terbuka (*open-source*), yang berfungsi sebagai intercepting proxy interaktif dengan dukungan SSL/TLS untuk HTTP/1, HTTP/2, dan WebSockets. Perangkat ini memiliki berbagai antarmuka (konsol, web, dan dump-style) yang menggunakan mesin proksi inti yang sama.
- **Postman** adalah klien API yang memungkinkan pembuatan dan pengiriman permintaan HTTP kustom dengan kendali penuh atas header dan body. Penguji menggunakan Postman untuk mengulang atau memodifikasi panggilan API telematika

yang telah ditangkap (misalnya, perintah untuk membuka pintu mobil) dengan berbagai parameter, guna melakukan upaya eksploitasi (seperti *replay attack* atau manipulasi parameter).

- **OWASP ZAP (*Zed Attack Proxy*)** adalah pemindai aplikasi web sumber terbuka yang mencegat, memeriksa, dan memodifikasi lalu lintas HTTP serta HTTPS antara klien dan server. Penguji keamanan dapat mengonfigurasi aplikasi pendamping kendaraan atau klien pembaruan OTA agar lalu lintasnya diarahkan melalui ZAP. Hal ini memungkinkan pemindaian pasif untuk mendeteksi header keamanan yang hilang, serta pemindaian aktif untuk menemukan celah injeksi, kontrol akses yang rusak (*broken access control*), dan titik akhir (*endpoint*) API yang tidak memerlukan autentikasi. ZAP merupakan alternatif sumber terbuka langsung untuk Burp Suite dalam penilaian aplikasi pendamping dan API web.
- **MQTT Explorer** adalah klien lintas platform yang terhubung ke broker MQTT dan menampilkan semua topik yang dipublikasikan dalam tampilan terstruktur seperti pohon (*tree-structured*), termasuk pesan yang disimpan (*retained messages*) dan muatan (*payload*) topik. Penguji keamanan dapat terhubung ke broker *backend* kendaraan seperti AWS IoT Core atau Eclipse Mosquitto untuk mengamati topik yang dipublikasikan oleh ECU, aplikasi seluler, dan layanan manajemen armada. Hal ini memungkinkan identifikasi topik tanpa autentikasi, muatan dalam bentuk teks terbuka (*cleartext*), serta saluran telemetri yang bersifat sensitif.
- **jwt_tool** adalah utilitas baris perintah untuk menguji *JSON Web Token* (JWT), termasuk pemeriksaan otomatis terhadap serangan tanpa algoritma (*non-algorithm attack*), kerentanan kebingungan algoritma (seperti RS256 yang diubah menjadi HS256), serta penggunaan kunci penandatanganan (*signing secret*) yang lemah. Seorang penguji penetrasi (*pentester*) dapat mencegat token autentikasi dari aplikasi pendamping kendaraan dan menjalankan jwt_tool untuk menentukan apakah *backend* menerima token yang ditandatangani dengan algoritma null atau tanda tangan palsu. Jika diterima, hal ini dapat memberikan akses tidak sah ke API kontrol kendaraan, seperti perintah untuk membuka kunci pintu dari jarak jauh atau menyalakan mesin.
- **sslyze** melakukan analisis konfigurasi TLS secara otomatis terhadap server jarak jauh. Alat ini menguji keberadaan cipher suite yang lemah, kerentanan penurunan versi protokol (*protocol downgrade*), validitas sertifikat, OCSP stapling, serta penggunaan versi TLS yang sudah usang. Penguji keamanan dapat menjalankan sslyze pada endpoint API *backend* kendaraan maupun server pembaruan OTA untuk menghasilkan laporan kesalahan konfigurasi TLS dalam format yang dapat dibaca oleh mesin, yang kemudian menjadi dasar bukti atas temuan terkait risiko intersepsi lalu lintas data.
- **tcpdump** menangkap paket jaringan dari antarmuka tertentu dan menyimpannya ke dalam file *Packet Capture* (PCAP) untuk dianalisis secara offline menggunakan sintaks filter yang kompatibel dengan libpcap. Pentester menggunakan tcpdump pada server *backend* atau *gateway* di lingkungan tanpa antarmuka grafis (*headless*) di mana Wireshark tidak tersedia untuk menangkap aliran data *Transmission Control Protocol*

(TCP) mentah, siaran *User Datagram Protocol* (UDP), serta jabat tangan (*handshake*) protokol, yang nantinya dapat didekode menggunakan Wireshark atau Scapy.

- **boofuzz** adalah alat fuzzer protokol jaringan berbasis Python yang mendukung *fuzzing* stateful (berbasis status) untuk protokol biner maupun berbasis teks kustom. Alat ini bekerja dengan mendefinisikan grafik permintaan (*request graph*) yang memodelkan urutan sesi yang valid. Seorang penguji keamanan dapat menggunakan boofuzz untuk melakukan *fuzzing* pada API REST *backend* kendaraan serta saluran perintah MQTT dengan cara menghasilkan urutan permintaan yang tidak sesuai format (malformed), guna mengidentifikasi kondisi crash dan respons kesalahan tak terduga yang mengindikasikan kerusakan memori (*memory corruption*) atau cacat logika pada penangan pesan di sisi *backend*.
- **Peach Fuzzer** adalah fuzzer berbasis model yang menggunakan berkas pit (berformat XML) untuk mendefinisikan tata bahasa protokol serta menghasilkan kasus uji baik yang valid maupun yang tidak sesuai format secara terstruktur dan dapat diulang. Penguji keamanan dapat memanfaatkan alat ini dengan terlebih dahulu menyusun berkas pit yang mendeskripsikan protokol API *backend* kendaraan atau protokol biner telematika khusus, kemudian menjalankan Peach untuk menguji logika penguraian (*parsing*) secara sistematis, dengan tujuan mendeteksi *buffer overflow*, kesalahan bilangan bulat (*integer errors*), serta kegagalan dalam penanganan input.

Penilaian Postur Cloud dan Simulasi Eksploitasi Cloud

Alat audit keamanan cloud sangat berharga karena kelemahan *backend* yang paling sering ditemukan dalam praktik bukanlah kerentanan *zero-day* yang baru, melainkan kesalahan konfigurasi dan pengelolaan identitas.

- **Scout Suite** adalah alat audit keamanan multi-cloud sumber terbuka (*open-source*) yang mengumpulkan data konfigurasi cloud dan menyoroti area risiko untuk memberikan gambaran yang jelas mengenai permukaan serangan (*attack surface*).
- **Prowler** adalah alat penilaian keamanan cloud sumber terbuka yang mengevaluasi lingkungan AWS, Azure, dan GCP berdasarkan tolok ukur CIS, PCI-DSS, ISO 27001, serta kerangka kerja kepatuhan khusus. Penguji keamanan dapat menggunakan Prowler pada akun cloud *backend* kendaraan (OEM) untuk mendeteksi kebijakan IAM yang salah konfigurasi, bucket S3 dengan izin akses yang terlalu longgar, volume penyimpanan yang tidak dienkripsi, serta celah dalam pencatatan (*logging*) semuanya dapat dieksploitasi oleh penyerang untuk mendapatkan akses persisten atau mencuri data telemetri.
- **Pacu** sangat berguna saat beralih dari tahap audit ke simulasi serangan terkendali. Alat ini merupakan kerangka kerja eksploitasi AWS sumber terbuka yang dirancang untuk pengujian ofensif terhadap lingkungan AWS dengan menggunakan serangkaian modul yang dapat diperluas.
- **Trivy** adalah pemindai kerentanan sumber terbuka untuk image kontainer, sistem berkas, dan konfigurasi *infrastructure-as-code* (IaC). Alat ini menghasilkan inventaris CVE yang dapat dibaca oleh mesin, disusun berdasarkan lapisan (layer) image. Penguji

keamanan dapat menggunakan Trivy untuk memindai image Docker yang digunakan dalam penerapan layanan mikro (*microservices*) *backend* kendaraan, guna mengidentifikasi lapisan base image yang rentan, instalasi paket yang salah konfigurasi, serta rahasia (*secrets*) yang terekspos di dalam sistem berkas image tersebut.

- **OWASP Dependency-Check** menganalisis dependensi proyek dan mengidentifikasi komponen yang memiliki CVE dengan membandingkan nama serta versi pustaka (*library*) terhadap *National Vulnerability Database*. Penguji keamanan dapat menjalankan *Dependency-Check* pada artefak build layanan *backend* kendaraan atau aplikasi pendukung untuk menemukan pustaka pihak ketiga yang rentan seperti stack TLS yang sudah usang atau kerangka kerja autentikasi yang tidak aman sehingga daftar perbaikan yang diprioritaskan dapat disusun sebelum sistem diterapkan (*deployed*).

Implementasi Referensi dan Perangkat Vendor

Penyedia layanan cloud sendiri menawarkan kerangka kerja penilaian keamanan, sebagaimana dibahas berikut ini:

- **AWS IoT Device Defender** adalah layanan pemantauan untuk AWS IoT yang dapat mengaudit konfigurasi IoT dan mendeteksi perilaku perangkat yang tidak wajar. Meskipun utamanya merupakan perangkat pertahanan, pemahaman mengenai pemeriksaan yang dilakukan oleh *Device Defender* (seperti audit kebijakan atau kedaluwarsa sertifikat) membantu penguji dalam mengetahui kesalahan konfigurasi apa saja yang perlu dicari (selama fase penemuan/*discovery*). Sebagai contoh, seorang penguji penetrasi (*pentester*) dapat memverifikasi apakah kebijakan AWS IoT telah menerapkan prinsip hak akses minimum (*least privilege*); jika ditemukan pelanggaran, maka hal tersebut menjadi temuan keamanan.
- **Microsoft Defender for IoT** sebelumnya dikenal sebagai *Azure Security Center for IoT* adalah layanan pemantauan keamanan berbasis cloud untuk lingkungan IoT dan teknologi operasional (OT). Layanan ini mengaudit konfigurasi Azure IoT Hub, mendeteksi perilaku perangkat yang tidak wajar, serta memetakan permukaan serangan (*attack surface*) dari kumpulan perangkat yang terhubung. Penguji keamanan dapat memanfaatkan fitur penemuan perangkat tanpa agen (*agentless*) dan hasil audit konfigurasi sebagai titik awal terstruktur untuk melakukan tinjauan manual terhadap postur keamanan cloud. Hal ini mencakup identifikasi celah pada kebijakan akses IoT Hub, pendaftaran perangkat tanpa autentikasi, serta kesalahan konfigurasi pada *device twin*, sebelum beralih ke pengujian eksploitasi aktif.
- **Eclipse Mosquitto** adalah message broker MQTT sumber terbuka (*open-source*) yang dapat diterapkan secara lokal oleh penguji keamanan untuk menyimulasikan broker *backend* kendaraan dalam lingkungan pengujian terkendali. Dengan menjalankan Mosquitto menggunakan konfigurasi ACL (*Access Control List*) standar atau yang sengaja dibuat lemah, penguji dapat memverifikasi apakah unit telematika kendaraan atau aplikasi pendamping terhubung ke broker tanpa autentikasi. Ini mengonfirmasi kerentanan terhadap serangan *rogue broker* (broker palsu), di mana penyerang dapat mencegat dan memodifikasi telemetri kendaraan atau pesan perintah.

Perangkat Dengan Dukungan AI

Dalam pekerjaan *backend*, AI sudah dapat diterapkan secara praktis karena API *backend* kaya akan teks, memiliki skema yang terstruktur, dan berbasis alur kerja (*workflow*). Burp AI merupakan contoh augmentasi AI yang mengubah alur kerja nyata: perangkat ini diintegrasikan ke dalam toolchain proksi dengan fokus pada peningkatan kecepatan dan pemahaman, bukan pada "eksploitasi otonom". Postman kini juga menyertakan kapabilitas berbasis AI (seperti "AI Request blocks" dalam alur kerja); namun, nilai keamanan praktisnya lebih terletak pada percepatan alur kerja dan dokumentasi, bukan pada kemampuan eksploitasi yang unik.

Postman tetap berfungsi terutama sebagai klien API dan lingkungan otomatisasi. Pada lapisan konfigurasi cloud, beberapa perangkat mengusung narasi "berbasis AI", namun nilai operasional yang sebenarnya tetap bersumber dari pemeriksaan dan enumerasi yang bersifat deterministik. Pendekatan yang dapat dipertanggungjawabkan dalam penggunaan AI di sini adalah untuk mempercepat proses *triase* (meringkas temuan, memetakan kesalahan konfigurasi ke jalur serangan yang nyata), dengan tetap memastikan bahwa bukti dasar yang digunakan berakar pada fakta API cloud serta log yang ada. Tabel berikut merangkum perangkat-perangkat representatif yang digunakan untuk intersepsi API, tinjauan postur cloud, validasi ofensif, penemuan eksposur, dan enumerasi bawaan penyedia layanan (*vendor-native*), beserta fase dan area target di mana masing-masing perangkat tersebut paling sering diterapkan:

Tabel 3.1: Survei mengenai perangkat pengujian keamanan cloud yang umum digunakan

Alat	Tujuan	Tahap Penggunaan	Target	Referensi
Burp Suite + Burp AI	Mencegat, memodifikasi, dan memutar ulang (*replay*) permintaan; akselerasi alur kerja dan analisis dengan bantuan AI	Penemuan/Eksploitasi	API telematika; lalu lintas *backend* seluler	Dokumentasi PortSwigger Burp AI.
mitmproxy	Mencegat/memeriksa/memodifikasi/memutar ulang lalu lintas web TLS menggunakan perangkat interaktif	Penemuan/Eksploitasi	API HTTP yang digunakan oleh aplikasi dan layanan kendaraan	Dokumentasi mitmproxy.
Scout Suite	Penilaian postur keamanan multi-cloud; pemetaan permukaan serangan (*attack surface*) melalui API cloud	Penemuan	Sumber daya cloud yang mendukung layanan kendaraan	Repositori Scout Suite.
Prowler	Pemeriksaan dan pelaporan keamanan cloud secara otomatis	Penemuan	Kondisi keamanan cloud di lingkungan AWS/Azure/GCP	Repositori Prowler.

Pacu	Modul simulasi serangan dan eksploitasi untuk akun AWS	Eksploitasi/Validasi	Kesalahan konfigurasi pada *backend* telematika yang di-hosting di AWS	Repositori/wiki Pacu.
Shodan/ Censys	Penemuan eksposur internet pada titik akhir (*endpoint*) publik	Penemuan	Layanan dan perantara (*broker*) yang terkait dengan kendaraan dan terekspos ke publik	Situs Shodan; dokumentasi pemindaian Censys.
AWS CLI / Azure CLI	Enumerasi yang realistis dan validasi tindakan menggunakan API vendor	Validasi	Sumber daya cloud, registri IoT, identitas, serta kebijakan	Dokumentasi AWS CLI; dokumentasi Azure CLI.

Setelah meninjau perangkat keamanan ofensif pada lapisan *backend*, kini kita akan beralih ke lapisan nirkabel kendaraan (*vehicle wireless layer*), di mana perangkat yang digunakan lebih berfokus pada visibilitas frekuensi radio (RF), perangkat keras khusus protokol, serta jaringan laboratorium yang terkendali.

3.2 PERANGKAT ANTARMUKA NIRKABEL

Sebagaimana akan kita lihat pada bab 5, kendaraan terhubung (*connected vehicles*) memiliki berbagai antarmuka nirkabel mulai dari Wi-Fi dan Bluetooth yang digunakan untuk sistem hiburan (*infotainment*) serta penyandingan dengan perangkat seluler, hingga koneksi seluler (4G/LTE/5G) untuk telematika dan komunikasi V2X. Penguji penetrasi (*penetration teste*r) memerlukan perangkat Frekuensi Radio (*Radio Frequency*/RF) khusus untuk menemukan dan mengeksploitasi kerentanan pada saluran-saluran tersebut. Bagian ini membahas perangkat untuk serangan dan analisis nirkabel yang mencakup berbagai teknologi radio yang lazim ditemukan di dalam maupun di sekitar kendaraan modern, termasuk Wi-Fi, Bluetooth dan BLE, koneksi telematika seluler, RFID/NFC, GNSS/GPS, serta tumpukan komunikasi V2X.

Pengujian nirkabel memiliki karakteristik unik dibandingkan dengan pengujian *backend*: jika kompromi pada *backend* lebih banyak berkaitan dengan aspek identitas dan kebenaran API, maka kompromi pada sistem nirkabel lebih dipengaruhi oleh realitas lapisan fisik (*physical layer*) dan tumpukan jaringan (*network stack*) seperti jangkauan, interferensi, pengaturan waktu (*timing*), dan status protokol.

Penemuan Wi-Fi dan Manipulasi Sisi Klien

Banyak kendaraan terhubung menyediakan konektivitas Wi-Fi; contohnya, hotspot Wi-Fi di dalam kendaraan untuk penumpang, atau Wi-Fi yang digunakan selama proses

pembaruan perangkat lunak maupun diagnostik. Perangkat pengujian penetrasi nirkabel standar dapat diterapkan di sini:

- **Aircrack-ng** adalah rangkaian perangkat sumber terbuka (*open-source*) untuk audit Wi-Fi 802.11. Dalam tahap penemuan (*discovery*), airodump-ng dapat digunakan untuk memindai SSID Wi-Fi kendaraan, kekuatan sinyal, dan klien yang terhubung. Jika jaringan Wi-Fi menggunakan keamanan yang lemah (misalnya, WPA2 dengan kata sandi yang lemah), penguji dapat menggunakan aireplay-ng untuk menangkap handshake dan aircrack-ng untuk mencoba membobol kata sandi tersebut. Dalam tahap eksploitasi, fitur injeksi paket pada Aircrack dapat digunakan untuk memutar ulang (*replay*) lalu lintas jaringan yang telah ditangkap atau melakukan deauthentication (memutuskan koneksi) terhadap klien, yang berpotensi mengganggu sistem kendaraan yang bergantung pada Wi-Fi. Perangkat ini dikelola secara aktif dan digunakan secara luas dalam pengujian keamanan nirkabel secara umum.
- **Kismet** adalah detektor dan sniffer jaringan nirkabel pasif yang berguna untuk menemukan jaringan ilegal (*rogue*) atau tersembunyi. Kismet dapat mendeteksi apakah sebuah kendaraan sedang memancarkan sinyal jaringan Wi-Fi tersembunyi atau apakah terdapat perangkat Wi-Fi yang tidak sah (misalnya, perangkat tambahan pihak ketiga). Alat ini mencatat SSID dan alamat MAC, serta dapat memberikan peringatan jika mendeteksi jaringan ad-hoc atau sinyal beacon yang tidak lazim. Penguji penetrasi (*penetration tester*) dapat menggunakan Kismet untuk memetakan lingkungan frekuensi radio (RF) di sekitar kendaraan guna mengidentifikasi semua perangkat yang memancarkan sinyal (contohnya, untuk menemukan antarmuka debugging nirkabel yang tidak ditujukan untuk penggunaan konsumen).
- **Pineapple (Hak5)** adalah perangkat keras khusus yang menjalankan *firmware* kustom, yang pada dasarnya berfungsi sebagai perangkat untuk membuat titik akses ilegal (*rogue access point*). Penguji dapat menggunakannya untuk melakukan serangan *man-in-the-middle* (MitM) terhadap kendaraan yang terhubung ke jaringan Wi-Fi tertentu. Sebagai contoh, jika sebuah mobil secara otomatis terhubung ke jaringan Wi-Fi rumah pemiliknya, Pineapple dapat meniru SSID tersebut untuk menangkap lalu lintas data atau melakukan phishing pada sistem infotainment kendaraan. Pineapple mengotomatiskan tahapan penemuan dan serangan (eksploitasi) terhadap klien Wi-Fi.
- **Flipper Zero** mendukung berbagai fitur, termasuk pembacaan sinyal Sub-GHz dan RFID 125 kHz. Deskripsi produk dari pihak produsen menyebutnya sebagai perangkat serbaguna (*multi-tool*) untuk "menjelajahi sistem kontrol akses dan protokol radio". Perangkat semacam ini sering digunakan dalam kegiatan pengintaian lapangan karena mempermudah persiapan, meskipun untuk validasi yang lebih serius, biasanya tetap diperlukan peralatan RF yang lebih khusus dan berstandar laboratorium.
- **hashcat** adalah perangkat pemulihan kata sandi yang dipercepat oleh GPU (*GPU-accelerated*). Alat ini mendukung berbagai metode serangan, seperti serangan kamus (*dictionary*), *brute-force*, berbasis aturan (*rule-based*), dan mask, untuk berbagai jenis hash. Seorang penguji keamanan dapat menggunakan hashcat untuk meretas

handshake WPA-PSK yang sebelumnya ditangkap menggunakan airodump-ng dari titik akses diagnostik Wi-Fi kendaraan guna menilai apakah kata sandi yang melindungi antarmuka diagnostik kendaraan tersebut dapat dipulihkan secara offline. hashcat merupakan langkah standar dalam proses peretasan setelah keberhasilan penangkapan handshake WPA, sekaligus melengkapi alur kerja airodump-ng yang telah dijelaskan sebelumnya dalam bagian ini.

Pengintaian dan Eksperimentasi Bluetooth/BLE

Bluetooth digunakan dalam kendaraan untuk sistem *hands-free*, penyandingan (*pairing*) ponsel pintar, dan kini semakin banyak digunakan untuk kunci digital (*Bluetooth Low Energy* atau BLE). Berbeda dengan Wi-Fi, Bluetooth memerlukan perangkat keras khusus untuk melakukan penyadapan (*sniffing*) dan injeksi data, dikarenakan adanya mekanisme *frequency hopping* (lompatan frekuensi) serta kompleksitas protokolnya. Berikut adalah beberapa perangkat utama yang akan kita bahas:

- **Ubertooth One** merupakan platform pengembangan nirkabel 2,4 GHz sumber terbuka (*open-source*) yang berfungsi sebagai alat penyadap (*sniffer*) Bluetooth. Ubertooth dapat memantau komunikasi Bluetooth Classic maupun BLE, menangkap paket data, bahkan menyadap koneksi tertentu dengan mengikuti lompatan frekuensi yang terjadi. Perangkat ini banyak digunakan oleh para peneliti untuk menganalisis lalu lintas Bluetooth dan mengidentifikasi kerentanan dalam proses penyandingan. Dalam tahap penemuan (*discovery*), penguji menggunakan Ubertooth untuk mencegat proses penyandingan antara ponsel dan head unit kendaraan secara pasif yang memungkinkan perolehan alamat perangkat atau bahkan peretasan metode penyandingan *Just Works* (menggunakan alat seperti Crackle untuk standar Bluetooth lama). Perangkat ini juga mendukung identifikasi saluran iklan (*advertisement channels*), pencegatan permintaan koneksi BLE, dan sebagainya, sehingga sangat berguna untuk mengeksplorasi *attack surface* (permukaan serangan) pada Bluetooth.
- **BtleJack** adalah perangkat khusus untuk penyadapan dan serangan pada BLE yang beroperasi menggunakan perangkat keras Ubertooth One. Alat ini mendukung pelacakan koneksi BLE, penangkapan paket secara pasif, serta pembajakan koneksi secara aktif. BtleJack memungkinkan penguji untuk membajak koneksi BLE dengan cara menyamar sebagai master BLE, atau dengan melakukan jamming (gangguan sinyal) untuk kemudian mengambil alih sesi. Hal ini dapat dimanfaatkan dalam tahap eksploitasi; misalnya, untuk menyuntikkan perintah berbahaya ke dalam komunikasi kunci digital berbasis BLE antara aplikasi ponsel dan kendaraan (jika protokol tersebut tidak memiliki enkripsi atau autentikasi yang memadai). BtleJack merupakan perangkat yang terus dikembangkan secara aktif dan berfokus pada keamanan BLE versi 4.x dan 5.x.
- **BlueZ** adalah stack protokol Bluetooth resmi untuk sistem operasi Linux. BlueZ menyediakan serangkaian utilitas baris perintah seperti hcitool, btmgmt, dan gatttool yang memungkinkan penguji untuk berinteraksi langsung dengan perangkat Bluetooth maupun BLE dari host Linux. Penguji keamanan dapat menggunakan alat-alat ini untuk

memindai perangkat Bluetooth di sekitar, menanyakan layanan dan profil yang didukung, serta melakukan enumerasi karakteristik GATT (*Generic Attribute Profile*) BLE pada antarmuka kendaraan. Hal ini mencakup penemuan endpoint telemetri, layanan kunci digital, serta sinyal iklan BLE untuk sistem akses tanpa kunci (*passive entry*), tanpa memerlukan perangkat lunak tambahan.

- **Bettercap** adalah kerangka kerja (*framework*) untuk serangan dan pemantauan jaringan yang mencakup Ethernet, Wi-Fi, dan *Bluetooth Low Energy* (BLE) dalam satu lingkungan interaktif. Penguji keamanan dapat menggunakan modul BLE pada Bettercap untuk melakukan enumerasi terhadap layanan dan karakteristik GATT pada antarmuka BLE kendaraan, membaca serta menulis nilai karakteristik, dan mencoba mengeksploitasi implementasi pemasangan (*pairing*) yang tidak aman. Dengan demikian, Bettercap mencakup fase pengintaian (*reconnaissance*) maupun pengujian aktif dalam penilaian BLE melalui satu perangkat saja.
- **Crackle**: Setelah penyerang berhasil menangkap urutan pemasangan (*pairing sequence*) menggunakan perangkat seperti Ubertooth, mereka masih harus menghadapi enkripsi yang melindungi tautan tersebut. Crackle adalah utilitas khusus yang dirancang untuk meretas kunci jangka panjang (*long-term keys*) pada *Bluetooth Low Energy* (BLE) dengan memanfaatkan kelemahan dalam model pemasangan warisan (*Legacy Pairing*), khususnya metode "*Just Works*" dan "*Passkey*". Dengan menurunkan kunci enkripsi dari jejak data yang tertangkap, penguji dapat mendekripsi lalu lintas data selanjutnya seperti perintah untuk "membuka kunci" atau "menyalakan mesin" sehingga aspek kerahasiaan protokol tersebut menjadi tidak berarti. Oleh karena itu, penilaian terhadap kunci digital berbasis BLE dianggap belum lengkap tanpa memverifikasi ketahanan model asosiasi tersebut terhadap upaya peretasan otomatis semacam itu.
- **InternalBlue**: Bagi peneliti yang bekerja pada lapisan arsitektur yang lebih dalam, InternalBlue menyediakan kerangka kerja untuk melakukan debugging pengontrol Bluetooth di tingkat *firmware*. Berbeda dengan perangkat sisi host yang berinteraksi dengan tumpukan (*stack*) Bluetooth melalui API standar, InternalBlue memungkinkan pemeriksaan dan manipulasi terhadap status internal pengontrol. Hal ini sangat berguna saat mencari kerentanan tingkat rendah di dalam *firmware chip Bluetooth* itu sendiri, yang berpotensi melewati kontrol keamanan di tingkat sistem operasi (OS). Meskipun lebih khusus dibandingkan perangkat sniffer standar, InternalBlue mewakili pendekatan "*white-box*" yang paling mendalam dalam pengujian ofensif Bluetooth.

Jaringan Laboratorium Seluler dan Pengujian Jaringan Palsu (*Rogue Network*)

Kendaraan yang terhubung (*connected vehicles*) biasanya dilengkapi dengan modem seluler (LTE atau 5G) untuk telematika guna mendukung layanan seperti pembukaan kunci jarak jauh dan pembaruan perangkat lunak melalui udara (OTA). Melakukan serangan langsung terhadap komunikasi seluler memerlukan perangkat canggih sering kali berupa *Software Defined Radio* (SDR) yang dikombinasikan dengan stack jaringan sumber terbuka (*open-source*) karena lalu lintas seluler yang terenkripsi tidak dapat sekadar "disadap" (*sniffed*) di udara tanpa

menggunakan emulator stasiun pangkalan (*base station*). Pengujian penetrasi pada antarmuka seluler berfokus pada dua aspek:

- Kemampuan kendaraan untuk mengautentikasi jaringan seluler (perlindungan terhadap stasiun pangkalan palsu).
- Layanan apa pun yang terekspos melalui jaringan seluler (layanan yang dapat diakses melalui alamat IP kendaraan).

Pada bagian ini, kita akan meninjau perangkat yang umum digunakan untuk menguji aspek-aspek tersebut:

- ***Software Defined Radio* (SDR)**: Perangkat keras seperti USRP (*Universal Software Radio Peripheral*) atau perangkat yang lebih terjangkau, HackRF One, sangat penting untuk pengujian seluler. SDR mampu memancarkan dan menerima sinyal radio apa pun. Dalam konteks kendaraan, penguji dapat menggunakan SDR untuk menyiapkan stasiun pangkalan palsu (*rogue base station*) guna keperluan penemuan (*discovery*) dan eksploitasi. Sebagai contoh, dengan menggunakan perangkat seperti srsRAN, penguji dapat menyebarkan menara seluler 4G LTE palsu yang mungkin dihubungkan oleh kendaraan jika kendaraan tersebut tidak melakukan autentikasi terhadap identitas jaringan. SDR juga memungkinkan penyadapan pesan siaran (*broadcast messages*) atau melakukan serangan penolakan layanan (*denial of service*) melalui gangguan sinyal (*jamming*). HackRF One (yang beroperasi pada rentang 1 MHz hingga 6 GHz) banyak digunakan untuk menguji teknologi nirkabel modern. Peretas biasanya menggunakan perangkat SDR seperti HackRF atau bahkan RTL-SDR untuk menyadap dan memanipulasi komunikasi nirkabel kendaraan.
- **srsRAN** (sebelumnya dikenal sebagai srsLTE) adalah perangkat lunak jaringan akses radio (*radio access network*) 4G/5G sumber terbuka. srsRAN memungkinkan penguji yang menggunakan SDR untuk meniru (*emulate*) eNodeB (stasiun pangkalan) LTE beserta jaringan intinya (*core network*). Dalam lingkungan yang terkendali (misalnya, di dalam ruang kedap sinyal radio/RF), penguji penetrasi dapat mengonfigurasi srsRAN agar berfungsi sebagai jaringan seluler dan membuat kendaraan terhubung ke jaringan tersebut. Hal ini memungkinkan pengujian apakah unit telematika memvalidasi kredensial jaringan seluler dengan benar. Penelitian menunjukkan bahwa kendaraan yang memprioritaskan konektivitas mungkin dapat terhubung ke sinyal LTE jahat yang kuat, sehingga memungkinkan potensi serangan MitM meskipun LTE memiliki otentikasi timbal balik (misalnya, dengan mengeksploitasi fallback ke 2G atau menggunakan serangan pengacauan dan penurunan kualitas). Dengan srsRAN, penguji juga dapat mengamati semua lalu lintas dari kendaraan di jaringan jahat dalam bentuk teks biasa (penemuan apa yang dikirimkan mobil ke cloud) dan mencoba eksploitasi seperti memanipulasi respons DNS atau mengirimkan pembaruan OTA berbahaya melalui server palsu.
- **OpenLTE/Open5GS** merupakan proyek sumber terbuka (*open-source*) alternatif untuk menyimulasikan jaringan seluler atau elemen-elemen jaringan inti (*core network*). Perangkat seperti OpenLTE dapat memantau kanal siaran (*broadcast channel*) LTE

tertentu secara pasif untuk memperoleh informasi mengenai jaringan (seperti ID sel atau daftar sel tetangga). Informasi ini dapat menjadi bagian dari tahap pengintaian (*reconnaissance*) untuk merencanakan serangan IMSI-catcher. Dalam praktiknya, perangkat ini berfungsi sebagai lapisan awal yang memberikan data untuk tahapan pengujian jaringan seluler selanjutnya.

- **OpenBTS** adalah *base transceiver station* (BTS) GSM berbasis perangkat lunak (*software-defined*) yang mengimplementasikan tumpukan protokol antarmuka udara (*air interface*) GSM Um menggunakan perangkat keras SDR (*Software-Defined Radio*) standar. Penguji keamanan dapat menggunakan OpenBTS di lingkungan laboratorium yang terlindung dari sinyal radio (*RF-shielded*) untuk membuat sel GSM palsu. Tujuannya adalah untuk menilai apakah unit telematika kendaraan atau modem seluler tertanam (*embedded*) akan terhubung ke sel tersebut tanpa memvalidasi identitas operator jaringan, sebuah kondisi yang memungkinkan serangan *Man-in-the-Middle* (MitM) pada komunikasi seluler kendaraan. OpenBTS melengkapi srsRAN dengan mencakup lapisan GSM/2G, yang relevan untuk menguji perilaku *fallback* (pengalihan ke jaringan yang lebih lama) pada unit telematika model lama.
- **Detektor IMSI Catcher / simulator Stingray**: Perangkat ini bukan alat "pengujian penetrasi" (*penetration testing*) dalam arti konvensional, melainkan digunakan untuk menguji ketahanan kendaraan terhadap menara seluler palsu. Sebagai contoh, dengan menggunakan perangkat SDR sederhana dan skrip Python, penguji dapat memancarkan sinyal 2G yang kuat (mengingat banyak modem masih memiliki fitur fallback ke 2G/GSM) untuk melihat apakah kendaraan akan terhubung ke sinyal tersebut. Selanjutnya, perangkat lain dapat digunakan untuk menangkap IMSI (pengidentifikasi unik) dari kartu SIM kendaraan, hal yang menjadi perhatian terkait privasi atau pelacakan. Suite perangkat lunak Osmocom (seperti OpenBSC, dll.) dapat digunakan untuk menyiapkan stasiun pangkalan (BTS) GSM semacam itu.
- **Wireshark** (dengan dissector 5G/LTE): Jika penguji berhasil menangkap lalu lintas seluler (misalnya menggunakan srsRAN atau emulator stasiun pangkalan seperti yang disebutkan sebelumnya), Wireshark dapat digunakan untuk mendekode protokol lapisan yang lebih tinggi, seperti NGAP, NAS, atau protokol diagnostik (misalnya, jika modul telematika memiliki port diagnostik terbuka yang dapat diakses melalui alamat IP seluler).

Alat Analisis RFID/NFC

Sistem akses kendaraan yang mengandalkan teknologi *Radio-Frequency Identification* (RFID) dan *Near-Field Communication* (NFC) termasuk immobilizer, sistem akses pasif (*passive entry*), dan kunci digital berbasis NFC memerlukan perangkat keras khusus yang mampu membaca, meniru (emulasi), dan memutar ulang (*replay*) sinyal transponder. Hal ini bertujuan untuk menilai apakah mekanisme autentikasi yang mendasarinya tahan terhadap serangan kloning maupun serangan relay.

- **Proxmark3** adalah platform riset RFID dan NFC sumber terbuka (*open-source*) yang mendukung aktivitas membaca, menulis, menyadap (*sniffing*), meniru, dan

mengkloning berbagai jenis transponder, baik pada pita frekuensi rendah (125 kHz) maupun frekuensi tinggi (13,56 MHz). Penguji dapat menggunakan perangkat ini untuk menangkap dan memutar ulang sinyal transponder immobilizer kendaraan, mengkloning kredensial akses pasif, serta menganalisis urutan *challenge-response* pada implementasi kunci digital berbasis NFC guna menentukan apakah skema autentikasinya rentan terhadap serangan relay atau replay.

- **Chameleon Mini** adalah perangkat keras sumber terbuka yang ringkas, dirancang untuk meniru dan menyadap kartu NFC serta RFID pada frekuensi 13,56 MHz. Perangkat ini dapat digunakan untuk meniru kredensial kunci digital kendaraan yang telah dikloning saat didekatkan ke pembaca akses pasif (*passive entry reader*). Dengan demikian, penguji dapat memeriksa apakah sistem tersebut menerapkan autentikasi kriptografis challenge-response atau justru menerima kredensial yang diputar ulang secara statis. Selain itu, perangkat ini juga dapat mencatat pertukaran data mentah antara kartu dan pembaca untuk keperluan analisis protokol secara offline.

Simulasi GNSS/GPS dan Pembuatan Sinyal Uji

Pengujian keamanan GNSS dalam sistem otomotif berfokus pada dua vektor utama: simulasi sinyal untuk mengevaluasi perilaku penerima (*receiver*) dalam kondisi terkendali, serta *spoofing* (pemalsuan sinyal) untuk menilai apakah sistem lokalisasi terintegrasi dan fitur keselamatan terkait mampu menolak input posisi, kecepatan, atau waktu yang tidak masuk akal.

- **GPS-SD™-SIM** menghasilkan aliran data sinyal baseband GPS yang dapat dikonversi menjadi sinyal RF menggunakan platform SDR (*Software-Defined Radio*) seperti HackRF One atau USRP. Hal ini memungkinkan penguji untuk membuat sinyal GPS sintetis tanpa harus memiliki simulator GNSS komersial. Penguji keamanan dapat menggunakannya di lingkungan terlindung (*shielded environment*) untuk menyuntikkan lintasan posisi dan waktu tertentu ke dalam penerima GNSS kendaraan, guna mengamati apakah sistem navigasi, fitur keamanan, atau layanan terkait menerima perubahan lokasi yang tidak masuk akal tanpa melakukan pemeriksaan validitas (*sanity checking*).
- **GNSS-SDR** adalah kerangka kerja penerima GNSS berbasis perangkat lunak (*software-defined*) sumber terbuka yang memproses sampel I/Q mentah dari front-end SDR untuk menghasilkan solusi posisi, kecepatan, dan waktu. Perangkat ini mendukung pemrosesan sinyal GPS, Galileo, GLONASS, dan BeiDou. Penguji keamanan dapat menggunakan GNSS-SDR untuk mendekode dan menganalisis struktur sinyal transmisi GNSS yang diterima di dekat kendaraan target, sehingga memperoleh wawasan mengenai parameter sinyal dan karakteristik waktu yang diperlukan untuk menyusun skenario *spoofing* yang kredibel.

Pembuatan Prototipe V2X dan Perangkat Stack Pesan

Pekerjaan yang bersifat ofensif dalam ranah V2X sering kali terkendala oleh kebutuhan akan perangkat keras khusus serta pertimbangan regulasi; namun, perangkat untuk protocol stack tetap krusial bagi keperluan pengujian dan analisis.

- **Vanetza** merupakan implementasi C++ sumber terbuka dari rangkaian protokol ETSI C-ITS, yang mencakup pesan pada lapisan *Facilities* (fasilitas), termasuk CAM dan DENM. Cohda Wireless SDK melengkapi jajaran perangkat keras MKx—serangkaian unit V2X yang dipasang pada kendaraan (*on-board*) maupun di tepi jalan (*roadside*) yang berfungsi sebagai *front-end* radio dalam konfigurasi pengujian, di mana Vanetza menyediakan protocol stack-nya.
- **scapy-radio**: Meskipun perangkat keras V2X khusus seperti Cohda Wireless MK5 merupakan standar industri untuk pengujian, perangkat tersebut sering kali terlalu mahal untuk tahap awal penelitian. scapy-radio berfungsi sebagai jembatan perangkat lunak yang krusial, memperluas kerangka kerja penyusunan paket Scapy untuk mendukung penyandian bingkai (*frame*) 802.11p yang diperlukan bagi DSRC (*Dedicated Short-Range Communications*). Jika dipadukan dengan perangkat SDR (*Software-Defined Radio*) serbaguna seperti HackRF, scapy-radio memungkinkan penyuntikan *Basic Safety Messages* (BSM) atau pengujian ketahanan fusi pesan tanpa memerlukan perangkat pengembangan V2X khusus. Hal ini memungkinkan para insinyur untuk beralih dari sekadar "peretasan bus" (*bus hacking*) standar menuju "pemalsuan lingkungan" (*environmental spoofing*) dengan memanfaatkan perangkat RF yang sudah mereka miliki.

Alat yang Diperkuat Dengan AI

Nirkabel merupakan salah satu lapisan yang paling sulit untuk diintegrasikan dengan AI agar tetap efektif dalam kondisi pengujian nyata, karena lingkungan RF cenderung penuh derau (*noisy*) dan penentuan kebenaran dasar (*ground truth*) memerlukan biaya yang tinggi. Meski demikian, AI sudah dapat diterapkan secara praktis dalam satu bidang: deteksi dan klasifikasi sinyal. DeepSig memperkenalkan OmniSIG sebagai solusi deteksi dan klasifikasi berbasis AI untuk data I/Q waktu nyata yang diterima oleh perangkat radio yang didukung, dengan penerapan pada sistem Linux di berbagai konfigurasi perangkat keras seperti ARM, x86, dan GPU.

Dalam konteks penggunaan LLM (seperti untuk pembuatan fuzz protokol bagi tumpukan protokol nirkabel yang bersifat *stateful*), sering kali yang ditemukan hanyalah prototipe hasil penelitian atau alur kerja yang terbatas pada lingkungan laboratorium. Tabel berikut merangkum berbagai perangkat penilaian nirkabel berdasarkan bidang kegiatannya, yang mencakup penemuan RF secara pasif, pengujian Wi-Fi dan Bluetooth, pekerjaan laboratorium berbasis SDR, emulasi jaringan seluler, analisis RFID/NFC, validasi GNSS, pengujian V2X, serta klasifikasi RF dengan bantuan AI:

Tabel 3.2: Alat keamanan jaringan nirkabel

Alat	**Tujuan**	**Fase Penggunaan**	**Target**	**™referensi**
Aircrack-ng	Perangkat untuk menangkap, menyuntikkan, dan	Penemuan/Eksploitasi	Titik akses (*hotspot*) pada kendaraan; saluran pembaruan melalui Wi-Fi	Dokumen Aircrack-ng.

	mengevaluasi lalu lintas Wi-Fi			
Kismet	Penemuan RF/nirkabel secara pasif pada berbagai protokol	Penemuan	Pemetaan lingkungan RF Wi-Fi/Bluetooth di sekitar kendaraan	Dokumen situs Kismet.
Flipper Zero	Perangkat serbaguna portabel untuk pengintaian sub-GHz/RFID	Penemuan	Pengintaian lapangan terkait kontrol akses dan protokol radio	Dokumentasi Flipper; deskripsi produk.
WiFi Pineapple	Platform khusus untuk rogue AP (titik akses tidak sah) dan audit nirkabel	Penemuan/ Eksploitasi	Hubungan kepercayaan (*trust relationships*) Wi-Fi di sisi klien	Dokumentasi Hak5 PineAP.
Ubertooth	Perangkat keras untuk sniffing BLE dan eksperimen Bluetooth	Penemuan	Analisis lalu lintas dan pemasangan (*pairing)* Bluetooth/BLE	Repositori Ubertooth; catatan status produk.
BtleJack	Alur kerja sniffing, jamming, dan pembajakan BLE menggunakan micro:bit	Eksploitasi/ Validasi	Manipulasi protokol/sesi BLE	Repositori/*firmware* BtleJack.
srsRAN Project + Open5GS	Jaringan laboratorium LTE/5G terkendali (RAN + core) untuk keperluan pengujian	Penemuan/ Validasi	Perilaku modem telematika dalam jaringan terkendali	Dokumentasi srsRAN; dokumentasi Open5GS
HackRF / USRP	Platform SDR untuk eksperimen RF yang terkendali	Penemuan/ Validasi	Laboratorium RF untuk perangkat LTE/5G, pita ISM, dan teknologi terkait V2X	Halaman HackRF; manual UHD/USRP
MahasiG	Deteksi dan klasifikasi sinyal RF berbasis AI dari data I/Q	Penemuan	Klasifikasi lingkungan RF dan identifikasi pemancar	Halaman DeepSig OmniSIG.
Proxmark3 (™FID™esearchGroup fork)	Interaksi, analisis, dan otomatisasi RFID/NFC	Penemuan/ Validasi	Penggunaan RFID/NFC dalam alur kerja pengendalian akses	Repositori RRG proxmark3; repositori resmi yang telah diarsipkan tercantum di sini.
GPS-SD™-SIM	Pembuatan baseband GPS untuk	Validasi	Pengujian penerima GNSS dan lokalisasi	Repositori GPS-SDR-SIM.

	konversi RF berbasis SDR		terpadu dalam pengaturan terkendali	
Vanetza/ Cohda SDK	Tumpukan protokol V2X dan ekosistem SDK perangkat	Penemuan/ Validasi	Pengujian tumpukan pesan ETSI ITS-G5 / C-ITS	Repositori Vanetza.

3.3 PERANGKAT JARINGAN DALAM KENDARAAN

Di dalam mobil, jaringan dalam kendaraan (*in-vehicle network*) merupakan area serangan yang luas, sebagaimana akan kita lihat pada bab C. Jaringan ini menjadi sarana bagi ECU (*Electronic Control Unit*) untuk berkomunikasi satu sama lain serta dengan berbagai sensor dan aktuator. Kendaraan yang terhubung (*connected vehicle*) menggunakan perpaduan protokol komunikasi klasik seperti CAN bus (*Controller Area Network*) dan LIN untuk komunikasi berbasis pesan serta Automotive Ethernet untuk komunikasi berbasis layanan guna mendukung transmisi data berkecepatan tinggi dan sensor pada sistem bantuan pengemudi canggih (*Advanced Driver Assistance Systems*/ADAS).

Penguji penetrasi (*penetration tester*) sering kali mengakses jaringan ini melalui port OBD-II atau sambungan kabel langsung. Bagian ini membahas dua kategori perangkat: perangkat berbasis CAN, CAN FD, dan SocketCAN untuk menguji protokol bus klasik, serta perangkat Automotive Ethernet untuk menguji komunikasi berbasis layanan dan IP. Kedua kategori tersebut mencakup antarmuka perangkat keras, perangkat lunak untuk menangkap dan menganalisis data, serta utilitas eksploitasi khusus protokol.

Alur Kerja Berbasis CAN/CAN FD dan SocketCAN

Pengujian jaringan CAN dan CAN FD pada platform berbasis Linux mengandalkan kerangka kerja SocketCAN, yang menyajikan antarmuka CAN sebagai perangkat jaringan standar dan didukung oleh sebagian besar perangkat keamanan otomotif sumber terbuka (*open-source*). Perangkat dalam kategori ini dibagi menjadi antarmuka perangkat keras, perangkat lunak, utilitas diagnostik, dan perangkat khusus untuk LIN.

Antarmuka CAN/LIN (Perangkat Keras): Akses fisik ke bus CAN atau LIN memerlukan antarmuka perangkat keras yang menghubungkan komputer ke kendaraan melalui port OBD-II atau sambungan kabel langsung. Adaptor berikut sering digunakan untuk tujuan ini:

- **CANtact**, sebuah adaptor USB-ke-CAN sumber terbuka yang dikenal sebagai "*The Open Source Car Tool*." Perangkat ini terdeteksi sebagai perangkat SocketCAN pada sistem Linux. CANtact memungkinkan pemantauan dan pengiriman frame CAN menggunakan perangkat lunak terbuka. Perangkat ini dikelola secara aktif dan memiliki harga yang terjangkau.
- **USB2CAN / CANable / PCAN-USB** adalah adaptor serupa yang menyediakan satu atau dua kanal CAN. Sebagai contoh, USB2CAN merupakan antarmuka murah yang terdaftar sebagai perangkat jaringan pada Linux. Perangkat-perangkat ini berfungsi sebagai alat penemuan (untuk memantau lalu lintas bus/sniffing) sekaligus alat eksploitasi (untuk mengirimkan frame yang telah dirancang khusus).

- **Modul antarmuka LIN**: Contohnya, LinUSBConverter (LIN-ke-USB dengan dukungan SLCAN) untuk analisis bus LIN.

Setelah antarmuka perangkat keras tersedia, perangkat lunak berikut digunakan untuk menangkap, mendekode, dan memanipulasi lalu lintas CAN.

- **can-utils** adalah rangkaian perangkat lunak berbasis baris perintah (*command-line*) dalam Linux SocketCAN. Utilitas utamanya meliputi candump (mencatat semua frame untuk dianalisis), cansniffer (menampilkan perubahan byte sinyal secara waktu nyata/real-time), cansend (mengirim frame), dan canplayer (memutar ulang jejak data yang telah dicatat). Perangkat-perangkat ini merupakan alat dasar untuk melakukan eksplorasi pada CAN. Untuk keperluan eksploitasi, cansend dapat digunakan untuk menyuntikkan paket secara manual guna menguji respons ECU (contohnya, mengirim pesan CAN untuk membuka kunci pintu). Perangkat-perangkat ini bersifat tingkat rendah (*low-level*) namun sangat fleksibel dan dikelola sebagai bagian dari sistem operasi Linux.
- **cantools** adalah pustaka Python dan perangkat baris perintah (*command-line tool*) untuk melakukan dekode serta enkode pesan CAN menggunakan berkas basis data DBC atau KCD. Perangkat ini mendukung ekstraksi sinyal, pembuatan pesan, dan penguraian (*parsing*) berkas log. Seorang penguji keamanan dapat menggunakan cantools untuk mengurai berkas DBC kendaraan dan mendekode berkas log candump menjadi nilai sinyal yang dapat dipahami manusia. Hal ini memungkinkan interpretasi cepat terhadap data mentah frame CAN tanpa perlu melakukan perhitungan bit-field secara manual, serta memfasilitasi pembuatan frame dengan enkoding yang tepat untuk keperluan pengujian injeksi.
- **SavvyCAN** adalah perangkat berbasis antarmuka grafis (GUI) lintas platform yang digunakan untuk reverse engineering dan penangkapan data pada bus CAN. SavvyCAN dapat terhubung ke antarmuka SocketCAN maupun antarmuka bawaan lainnya, serta menyediakan visualisasi lalu lintas CAN seperti pembuatan grafik nilai sinyal, penyaringan berdasarkan ID, dan interpretasi data menggunakan berkas DBC (basis data CAN) yang dimuat. Perangkat ini juga mendukung pengiriman frame dan pembuatan skrip dasar. Dalam tahap penemuan (*discovery*), SavvyCAN sangat mempercepat proses *reverse engineering* dengan memungkinkan pengguna untuk melihat pola (contohnya, memetakan sinyal untuk menentukan ID CAN mana yang berkaitan dengan kecepatan kendaraan). Untuk tahap eksploitasi, penguji dapat memanfaatkan fungsi transmisi atau fitur "serangan replay" (*replay attack*) pada SavvyCAN guna mengirimkan kembali rangkaian frame yang telah ditangkap sebelumnya (misalnya, untuk memicu kembali suatu tindakan, seperti membunyikan klakson).
- **Kayak** merupakan perangkat GUI CAN sumber terbuka (*open-source*) berbasis Java yang mendukung pencatatan (logging) dan pengiriman pesan CAN, serta pemindaian UDS (*Unified Diagnostic Services*). Perangkat ini berguna untuk menemukan ECU diagnostik dan mengirimkan permintaan diagnostik.

- **BUSMASTER™** adalah aplikasi sumber terbuka untuk simulasi, analisis, dan pengujian sistem bus CAN, LIN, dan FlexRay, yang dikembangkan di bawah inisiatif RBEI-ETAS. Penguji keamanan dapat menggunakan BUSMASTER untuk mencatat dan mengirimkan ulang (*replay*) lalu lintas CAN, menyimulasikan node ECU pada bus, serta menyuntikkan frame tertentu sembari memantau respons bus. Perangkat ini menjadi alternatif berbasis antarmuka grafis untuk sistem operasi Windows sebagai pengganti perangkat can-utils pada Linux terutama untuk lingkungan yang tidak mendukung SocketCAN.
- **CANalyzer** adalah perangkat komersial dari Vector yang banyak digunakan dalam industri otomotif. Perangkat ini menyediakan analisis profesional untuk protokol CAN/LIN dengan fitur-fitur lengkap, seperti analisis waktu (*timing analysis*) dan simulasi node. Para penguji di industri ini sering menggunakan perangkat dari Vector untuk menyimulasikan ECU atau menyusun skenario komunikasi yang kompleks. Sebagai contoh, Vector CANoe menghadirkan kemampuan untuk menyimulasikan seluruh jaringan kendaraan serta melakukan *fuzzing* menggunakan skrip CAPL, yang dapat menjadi sarana ampuh dalam merancang eksploitasi pada bus CAN.
- **CANToolz** merupakan kerangka kerja (*framework*) berbasis Python yang bersifat sumber terbuka (*open-source*), yang dirancang khusus untuk pengujian penetrasi (*pentesting*) jaringan kendaraan. CANToolz (dikenal juga sebagai YACHT) memungkinkan penggabungan berbagai modul dalam sebuah "alur kerja" (*pipeline*) untuk menjalankan tugas-tugas seperti *Man-in-the-Middle* (MitM) pada bus CAN, *fuzzing*, replay (pengiriman ulang pesan), hingga emulasi ECU. Perangkat ini mencakup modul untuk memindai layanan UDS (guna menemukan fungsi diagnostik) serta menyuntikkan kesalahan (*error injection*). Sebagai contoh, dengan menggunakan CANToolz, seorang penguji dapat menyiapkan proksi yang mampu memodifikasi pesan CAN tertentu secara langsung (*on-the-fly*). Perangkat ini dikelola secara aktif di GitHub dan dapat diintegrasikan dengan perangkat keras SocketCAN. Meskipun lebih kompleks dibandingkan can-utils, CANToolz sangat andal untuk melakukan pengujian *fuzzing* pada bus CAN secara sistematis.
- **Caring Caribou** adalah perangkat keamanan otomotif yang dirancang khusus untuk eksplorasi bus CAN. Perangkat ini mencakup modul khusus untuk penemuan layanan UDS, analisis seed-key, dan *fuzzing* pada tingkat UDS. Penguji keamanan dapat menggunakan Caring Caribou pada ECU melalui antarmuka SocketCAN untuk mendata layanan UDS dan jenis sesi yang didukung, mengidentifikasi fungsi diagnostik mana yang merespons tanpa autentikasi, serta membuat permintaan UDS yang tidak valid (*malformed*) guna menguji penanganan input dalam rutinitas diagnostik ECU.
- **Scapy** menyediakan lapisan protokol khusus otomotif meliputi CAN, ISO-TP, UDS, DoIP, dan SOME/IP melalui modul contrib-nya. Hal ini memberikan lingkungan berbasis skrip bagi penguji untuk menyusun dan mengurai lalu lintas protokol otomotif pada lapisan mana pun. Penguji keamanan dapat menggunakan Scapy untuk membangun pemindai kustom, memutar ulang sesi diagnostik yang telah direkam, serta menyusun urutan protokol yang tidak valid sesuatu yang mungkin tidak didukung oleh perangkat khusus

lainnya. Oleh karena itu, Scapy menjadi pilihan standar untuk eksperimen protokol dan pengembangan perangkat kustom dalam lingkungan pengujian jaringan kendaraan.

- **Pustaka python-can dan python-isoTp** memungkinkan penulisan skrip kustom untuk melakukan penemuan UDS. Sebagai contoh, seorang penguji dapat menulis skrip Python untuk menelusuri berbagai ID layanan UDS dan subfungsi yang mungkin ada, dengan memanfaatkan python-can untuk mengirim permintaan serta mencatat respons yang diterima.
- **UDSim** adalah simulator ECU sederhana untuk protokol UDS yang dapat dijalankan pada platform Arduino atau Linux. Penguji menggunakan UDSim untuk berlatih atau menyimulasikan ECU target saat menyusun sebuah eksploitasi.
- **Perangkat pemindai diagnostik (OEM)**: ini adalah perangkat kelas dealer (seperti Snap-On Solus atau pemindai Bosch) maupun perangkat resmi dari pabrikan kendaraan itu sendiri (seperti Tech2, VCDS, dll.). Meskipun tidak dirancang untuk peretasan, seorang pentester dapat memanfaatkannya dalam tahap penemuan untuk mengidentifikasi ID CAN dan fungsi-fungsi tertentu dengan mengamati data yang dikirimkan oleh perangkat resmi OEM tersebut. Risikonya, perangkat ini sendiri bisa menjadi target; para peneliti telah menunjukkan adanya kerentanan pada peralatan diagnostik tingkat dealer (seperti buatan Snap-On atau Bosch), yang berpotensi menyebarkan perangkat lunak berbahaya (*malware*) melalui port OBD-II. Dalam sebuah pengujian penetrasi, salah satu langkah yang dapat dilakukan sebagai bagian dari pemeriksaan pertahanan berlapis (*defense-in-depth*) adalah memverifikasi bahwa port OBD-II pada kendaraan tidak dapat dieksploitasi menggunakan perangkat tertentu.
- **Perangkat untuk LIN Bus**: Sering kali, *gateway* CAN akan meneruskan data LIN dalam bentuk frame CAN; namun, untuk komponen tertentu (seperti modul kursi yang menggunakan protokol LIN), pengujian langsung pada jalur LIN mungkin diperlukan. Perangkat untuk memantau lalu lintas LIN (LIN sniffer), seperti LinUSBConverter atau penganalisis logika (*logic analyzer*) yang dilengkapi dengan dekoder LIN, dapat digunakan. Salah satu pendekatan terbuka adalah menggunakan Arduino yang dipadukan dengan transceiver LIN dan pustaka (*library*) Arduino untuk LIN, guna menyuntikkan atau memantau frame LIN. Perangkat lunak seperti TSMaster (sebuah lingkungan pengembangan CAN/LIN komersial) juga mendukung analisis protokol LIN. Secara umum, setelah berhasil mengakses jalur LIN, penguji akan memeriksa apakah dimungkinkan untuk mengirimkan perintah yang tidak valid guna memicu perilaku yang tidak semestinya, atau melakukan eskalasi ke jaringan CAN melalui *gateway*.

Akses Ethernet Otomotif dan Perangkat Untuk Berbagai Protokol

Hambatan praktis dalam pengujian Ethernet otomotif sering kali terletak pada adaptasi lapisan fisik (*physical layer*). Technica Engineering mendeskripsikan konverter media 100BASE-T1 buatannya sebagai konversi lapisan fisik antara Ethernet otomotif dan NIC (*Network Interface Card*) RJ-45 standar. Intrepid mendeskripsikan RAD-Moon2 sebagai konverter media yang menjembatani Ethernet standar dengan Ethernet otomotif (100/1000BASE-T1). Setelah terhubung, perangkat jaringan yang umum digunakan dapat kembali dimanfaatkan, seperti

tcpdump, sslyze, dan mitmproxy yang telah disebutkan sebelumnya, serta perangkat lain yang dibahas dalam bagian ini:

- **Nmap** adalah pemindai jaringan sumber terbuka (*open-source*) yang mampu melakukan penemuan host (*host discovery*), pemindaian port, deteksi versi layanan, dan identifikasi sistem operasi (*OS fingerprinting*) menggunakan mesin penguji (*probe*) dan pengaturan waktu yang fleksibel. Penguji keamanan dapat menjalankan Nmap pada segmen Ethernet otomotif untuk mendata semua node yang aktif, mengidentifikasi port TCP dan UDP yang terbuka pada unit IVI (*In-Vehicle Infotainment*), ECU *gateway*, serta unit kontrol telematika, sekaligus mengidentifikasi versi layanan yang terekspos pada setiap host. Hal ini menghasilkan peta jaringan yang lengkap sebelum melakukan eksploitasi yang ditargetkan.
- **Wireshark** adalah penganalisis protokol jaringan yang menangkap paket dari antarmuka langsung atau file PCAP, lalu membedahnya berdasarkan berbagai protokol, termasuk SOME/IP, DoIP, dan TCP/IP standar. Penguji keamanan dapat menggunakan Wireshark pada perangkat pengujian (*test bench*) Ethernet otomotif untuk memeriksa lalu lintas penemuan layanan SOME/IP, mendekode sesi diagnostik DoIP antara perangkat penguji eksternal dan ECU *gateway*, serta mengidentifikasi payload dalam bentuk teks terbuka (*cleartext*) atau ketiadaan perlindungan integritas pada komunikasi antar-ECU celah yang dapat dieksploitasi oleh node mana pun yang memiliki akses ke segmen tersebut.
- **Scapy** menyediakan lapisan protokol khusus otomotif meliputi CAN, ISO-TP, UDS, DoIP, dan SOME/IP melalui modul kontribusi (*contrib modules*) seperti yang telah disebutkan di atas. Hal ini memberikan lingkungan berbasis skrip bagi penguji untuk menyusun dan mengurai lalu lintas protokol otomotif pada lapisan mana pun. Dengan demikian, seorang penguji keamanan dapat menggunakan Scapy pada perangkat uji (*bench*) *Automotive Ethernet* untuk menyusun permintaan diagnostik DoIP kustom, menyuntikkan pesan penemuan layanan (*service discovery*) SOME/IP yang tidak sesuai format, serta membangun urutan replay yang menargetkan ECU *gateway* dan pengontrol domain (*domain controller*). Hal ini mencakup interaksi protokol yang tidak didukung oleh perangkat khusus (yang dibuat untuk tujuan tertentu).
- **Metasploit** adalah kerangka kerja (*framework*) pengujian penetrasi yang menyediakan modul eksploitasi, pemindai tambahan (*auxiliary scanners*), dan kapabilitas pasca-eksploitasi untuk berbagai layanan jaringan serta kerentanan. Penguji keamanan biasanya menggunakan Metasploit pada segmen *Automotive Ethernet* untuk menjalankan pemindai tambahan terhadap layanan yang terbuka pada unit IVI (*In-Vehicle Infotainment*) dan *gateway* telematika, memvalidasi apakah CVE yang diketahui pada komponen Linux tertanam (*embedded Linux*) dapat dieksploitasi, serta menyimulasikan pergerakan lateral (*lateral movement*) setelah sistem berhasil disusupi. Tujuannya adalah untuk menilai apakah kontrol segmentasi jaringan di dalam kendaraan mampu membatasi jangkauan penyerang setelah mereka mendapatkan akses awal.

- **Ettercap** adalah perangkat intersepsi jaringan yang menggabungkan teknik ARP poisoning, penangkapan lalu lintas dengan metode man-in-the-middle, serta alat pengurai protokol (*protocol dissectors*) untuk menganalisis lalu lintas jaringan lokal. Penguji keamanan menggunakan Ettercap pada perangkat uji Automotive Ethernet untuk mencegat komunikasi antara unit IVI dan ECU *gateway* dengan cara meracuni (*poisoning*) cache ARP pada kedua node tersebut, lalu menyuntikkan paket yang telah dimodifikasi untuk menguji apakah perlindungan integritas pada lapisan transpor seperti TLS atau MACsec diterapkan pada segmen jaringan tersebut.
- **Yersinia** adalah perangkat untuk menyerang protokol jaringan yang mengimplementasikan modul eksploitasi aktif bagi protokol lapisan 2 (layer 2), termasuk STP, CDP, DHCP, HSRP, dan DTP. Penguji keamanan menjalankan Yersinia pada segmen Automotive Ethernet untuk mengirimkan notifikasi perubahan topologi STP yang telah dimanipulasi atau melakukan serangan kehabisan sumber daya DHCP (*DHCP exhaustion*). Hal ini bertujuan untuk menguji apakah switch Ethernet di dalam kendaraan menerapkan kontrol keamanan port yang dapat mencegah manipulasi protokol secara tidak sah dari node jaringan yang telah disusupi.
- **CANalyzer** juga dapat dimanfaatkan untuk pengujian penetrasi Automotive Ethernet jika dipadukan dengan perangkat VN5000, guna menganalisis berbagai protokol seperti SOME/IP, SOME/IP-SD, IP, TCP, UDP, AVB, RTP, PTP, dan banyak lagi.

Alat yang Diperkuat AI

Penguatan berbasis AI saat ini paling efektif diterapkan pada jaringan di dalam kendaraan (*in-vehicle networks*) jika protokol target bersifat terstruktur dan memiliki status (*stateful*), seperti SOME/IP, DoIP, atau UDS melalui IP. Sebaliknya, rekayasa balik (*reverse engineering*) pada protokol CAN mentah cenderung masih sangat bergantung pada intervensi manusia, karena memerlukan pengamatan terhadap perilaku fisik kendaraan setelah sinyal CAN dimanipulasi. CyTAL memperkenalkan ProtoCrawler sebagai alat yang menyediakan *fuzzing* cerdas untuk protokol otomotif termasuk SOME/IP, DoIP, UDS, dan stack terkait yang berfungsi untuk mengidentifikasi kelemahan implementasi protokol dalam arsitektur kendaraan modern.

Di sinilah kemampuan menyerupai AI dapat mengubah alur kerja: teknologi ini mampu mempercepat pembuatan pesan yang valid secara protokol serta transisi status, sehingga mempersingkat waktu yang dibutuhkan dari tahap "*saya bisa menyusup ke dalam protokol*" menjadi "*saya bisa menguji implementasi protokol tersebut secara sistematis*." Tabel berikut merangkum berbagai alat representatif untuk penilaian jaringan di dalam kendaraan, yang mencakup fungsi seperti penangkapan (*capture*) dan pemutaran ulang (*replay*) data CAN dan CAN FD, *fuzzing*, enumerasi diagnostik, akses ke Ethernet otomotif, inspeksi paket, pemindaian layanan IP, serta validasi yang memahami protokol terkait:

Tabel 3.3: Alat jaringan dalam kendaraan

Alat	**Tujuan**	**Fase Penggunaan**	**Target**	**Referensi**
can-utils	Alat ruang pengguna SocketCAN untuk pengambilan, pemutaran ulang, injeksi	Penemuan/Eksploitasi	Jaringan internal CAN/CAN FD	README linux-can can-utils.
SavvyCAN	Analisis GUI, visualisasi, rekayasa balik, pemutaran ulang	Penemuan/Eksploitasi	Inferensi dan pencatatan (*logging*) sinyal CAN/CAN FD	Repositori SavvyCAN
CANToolz	Pipeline CAN modular untuk MitM, *fuzzing*, pemindaian	Penemuan/Eksploitasi	Alur kerja pengujian keamanan CAN	README CANToolz; materi presentasi Black Hat (*konteks historis*).
Caring Caribou	Eksplorasi otomotif + penemuan UDS dan dukungan *fuzzing*	Penemuan/Validasi	UDS melalui CAN, alur kerja penemuan ECU (*ECU discovery*)	Repositori/dokumentasi Caring Caribou.
Scapy automotive layers	Pembuatan paket khusus untuk ISO-TP/UDS/DoIP/SOME-IP	Penemuan/Validasi	Lapisan transport dan layanan diagnostik otomotif	Dokumentasi otomotif Scapy; API kontribusi SOME/IP.
Wireshark	Inspeksi/analisis dengan dukungan SOME/IP dan DoIP	Penemuan	Sesi Automotive Ethernet, SOME/IP, dan DoIP	Referensi filter SOME/IP dan DoIP.
Technica MediaConverter/ Intrepid™ AD-Moon2	Akses lapisan fisik ke jaringan 100/1000BASE-T1	Fasilitator penemuan	Akses fisik Ethernet otomotif	Halaman produk Technica; halaman RAD-Moon2.
Nmap/ Metasploit	Enumerasi host/layanan dan kerangka kerja eksploitasi	Penemuan/Eksploitasi	Layanan yang diekspos pada jaringan IP di dalam kendaraan	Dokumentasi Nmap; tinjauan umum Metasploit.
ProtoCrawler	*Fuzzing* cerdas untuk protokol otomotif	Validasi	Tumpukan protokol SOME/IP, DoIP, dan UDS	Ulasan mengenai *fuzzing* protokol otomotif oleh CyTAL; halaman ProtoCrawler.

Setelah memahami bus dan tumpukan protokol internal, kita akan beralih ke titik akhir komputasi, dimulai dengan ECU berkinerja tinggi, di mana alat ofensif mulai menyerupai Linux tertanam dan eksploitasi titik akhir perusahaan.

3.4 ALAT ECU BERKINERJA TINGGI DAN BERBASIS MCU

ECU berkinerja tinggi mencakup sistem ADAS, unit infotainment, dan komputer pusat yang menjalankan Linux, Android, QNX, atau kombinasi dari sistem tersebut pada satu atau lebih SoC yang kuat. Seperti yang akan kita lihat di chauícu T, dalam banyak kasus, platform ini juga menampung *hypervisor* dan memiliki rantai boot aman yang kompleks. Konsekuensi keamanannya langsung terasa: ketika sebuah kendaraan mengkonsolidasikan fungsi ke dalam komputer pusat, satu kompromi dapat menyebar ke seluruh domain yang sebelumnya terpisah secara fisik.

Di sisi lain, ECU berbasis MCU masih dominan dalam fungsi seperti powertrain, subsistem pengereman, pengontrol bodi, dan banyak pengontrol periferal. Seperti yang akan kita lihat di chauícu k, lingkungan ini umumnya melibatkan AUTOSAR Classic, RTOS/*firmware* bare-metal, dan arsitektur seperti ARM Cortex-M, TriCore, PowerPC, dan keluarga MCU otomotif lainnya. Alat yang digunakan untuk pengujian penetrasi kedua jenis ECU sebagian besar tumpang tindih; oleh karena itu, kita akan menyajikan keduanya bersama-sama sambil menunjukkan kapan suatu alat dapat diterapkan pada satu jenis dan tidak pada jenis lainnya.

Penemuan Antarmuka Perangkat Keras dan Akses Debug

Sebelum analisis perangkat lunak atau *firmware* dapat dimulai, penguji sering kali perlu menentukan antarmuka fisik apa saja yang tersedia pada ECU dan apakah antarmuka tersebut menyediakan akses ke fungsi debug, pemrograman, atau boundary-scan. Aktivitas ini mencakup identifikasi titik uji (*test points*), pin konektor, antarmuka JTAG atau SWD, konsol UART, pin mode boot, serta titik akses perangkat keras lainnya yang mungkin memberikan akses istimewa ke perangkat tersebut.

Alat-alat berikut ini umum digunakan untuk menemukan, memvalidasi, atau berinteraksi dengan antarmuka tersebut selama penilaian ECU tertanam (*embedded ECU*):

- **JTAGulator** adalah perangkat keras sumber terbuka (*open-source*) yang membantu dalam mengidentifikasi antarmuka on-chip debug (OCD) melalui titik uji dan konektor.
- **OpenOCD** bertujuan untuk menyediakan kemampuan debugging, pemrograman dalam sistem (*in-system programming*), dan pengujian *boundary-scan* untuk perangkat target tertanam; alat ini sering menjadi penghubung antara *port debug* yang ditemukan dengan analisis perangkat lunak.
- **J-Link dari SEGGER™** merupakan ekosistem debug probe yang banyak digunakan dan mendukung berbagai perangkat lintas platform.
- **Bus Pirate** adalah alat serbaguna untuk bekerja dengan protokol serial seperti I2C, SPI, UART, dan lainnya, yang memungkinkan interaksi dengan cip tanpa perlu menulis kode khusus. Hal ini penting dalam pengerjaan ECU berbasis mikrokontroler (MCU) karena banyak informasi rahasia yang krusial bagi keamanan serta artefak konfigurasi boot disimpan pada periferal yang menggunakan protokol bus untuk berkomunikasi dengan MCU.
- **i2c-tools** (i2cdetect / i2cdump / i2cset) adalah utilitas baris perintah Linux untuk memeriksa dan berinteraksi dengan perangkat pada bus I2C: i2cdetect memindai bus untuk mencari alamat perangkat yang merespons, i2cdump membaca semua nilai

register dari perangkat tertentu, dan i2cset menuliskan nilai ke register tertentu. Penguji keamanan dapat menggunakan alat-alat ini pada ECU berbasis Linux atau unit IVI (*In-Vehicle Infotainment*) untuk mendata IC sensor dan pengontrol periferal yang terhubung, membaca register kalibrasi, serta menuliskan nilai uji guna memastikan apakah bus I2C tersebut tidak memiliki kontrol akses yang memadai yang dapat memungkinkan penyerang lokal untuk memanipulasi sensor yang terhubung dalam subsistem ADAS.

- **minicom** adalah program emulasi terminal untuk komunikasi port serial yang menyediakan antarmuka konsol berbasis teks ke sistem tertanam (*embedded systems*) melalui RS-232 atau adaptor USB-ke-serial. Seorang penguji keamanan menggunakan minicom untuk mengakses *bootloader* atau konsol serial Linux pada ECU kendaraan atau unit IVI, guna memeriksa apakah antarmuka serial dibiarkan terbuka pada perangkat produksi, apakah antarmuka tersebut memerlukan autentikasi, serta apakah tersedia akses root shell atau perintah *bootloader* yang dapat melewati verifikasi tanda tangan (*signature verification*) pada *firmware*.

Perangkat Untuk Penelitian *Fault Injection* (Injeksi Kesalahan) dan *Side-Channel* (Saluran Samping) (Validasi)

Ketika akses debug konvensional dikunci dan perlindungan *firmware* diaktifkan, penguji dapat beralih ke teknik validasi fisik yang memeriksa perilaku perangkat saat diukur atau diberi gangguan (*fault*). Perangkat *side-channel* mengamati kebocoran seperti emisi daya atau elektromagnetik untuk menyimpulkan informasi rahasia atau perilaku implementasi. Sementara itu, perangkat *fault injection* sengaja mengganggu eksekusi untuk menguji apakah pemeriksaan keamanan, logika *secure boot*, atau operasi kriptografis dapat dilewati (*bypassed*). Berikut adalah perangkat yang umum digunakan untuk memvalidasi apakah perlindungan berbasis perangkat keras tetap efektif dalam kondisi serangan fisik yang realistis:

- **ChipWhisperer** adalah perangkat sumber terbuka (*open-source*) yang berfokus pada analisis daya serta manipulasi tegangan atau clock (*glitching*) untuk memicu perilaku yang tidak diinginkan, seperti melewati pemeriksaan keamanan. Perangkat ini sering digunakan dalam penilaian keamanan MCU maupun HPC ketika akses debug dan *firmware* dibatasi.
- **ChipShouter** adalah platform *electromagnetic fault injection* yang mengirimkan pulsa bertegangan tinggi melalui probe genggam untuk memicu kesalahan sementara (*transient faults*) pada perangkat target tanpa memerlukan kontak elektrik langsung. Penguji keamanan (*pentester*) dapat menempatkan probe di atas MCU otomotif dan mengatur waktu pulsa agar bertepatan dengan instruksi yang krusial bagi keamanan misalnya saat verifikasi *secure boot* sehingga memaksa perangkat masuk ke status eksekusi yang tidak diinginkan.
- **™iscure Inspector** adalah platform komersial untuk analisis side-channel dan fault injection. Platform ini menggabungkan probe akuisisi perangkat keras dengan perangkat lunak untuk menangkap jejak daya (*power trace*) serta menerapkan teknik analisis statistik seperti *Differential Power Analysis* (DPA) dan *Correlation Power*

Analysis (CPA). Penguji keamanan dapat menggunakannya untuk mengekstrak kunci kriptografis dari implementasi pada ECU otomotif atau melakukan *fault injection* secara presisi terhadap rutinitas *secure boot*.

Akuisisi *Firmware* dan Rekayasa Balik (*Reverse Engineering*)

Setelah antarmuka perangkat keras atau artefak pembaruan diidentifikasi, penguji sering kali perlu memperoleh *firmware* dan mengubah berkas biner yang tidak transparan menjadi komponen perangkat lunak yang dapat dianalisis. Aktivitas ini dapat mencakup pembacaan memori *flash* eksternal, pembongkaran (*unpacking*) citra OTA, identifikasi sistem berkas, pembongkaran kode (*disassembling*) *bootloader*, emulasi *firmware*, atau instrumentasi proses yang sedang berjalan pada platform berbasis Linux, Android, atau QNX. Perangkat-perangkat berikut mendukung tahapan mulai dari perolehan *firmware* hingga analisis statis, emulasi, instrumentasi dinamis, dan penelitian kerentanan yang lebih mendalam:

- **flashrom** adalah utilitas untuk mendeteksi, membaca, menulis, memverifikasi, dan menghapus chip *flash*, serta mendukung berbagai perangkat pemrograman eksternal. Alat ini relevan digunakan ketika ECU menggunakan memori *flash* SPI eksternal, atau saat citra pembaruan (*update image*) dan artefak boot dapat diperoleh melalui pembacaan di luar perangkat (*off-board*).
- **Binwalk** adalah alat analisis *firmware* yang memindai citra biner untuk mencari tanda tangan file (file signature) yang tertanam, arsip terkompresi, dan format sistem file yang dikenal, lalu mengekstrak komponen-komponen tersebut secara otomatis. Penguji keamanan dapat menggunakannya untuk membongkar citra pembaruan OTA atau blob *firmware* ECU, guna memulihkan sistem file, file konfigurasi, dan komponen yang dapat dieksekusi (*executable*) untuk dianalisis lebih lanjut secara statis menggunakan Ghidra atau radare2.
- **QEMU** adalah emulator mesin dan virtualisator sumber terbuka (open-source) yang mendukung berbagai arsitektur prosesor, termasuk ARM, MIPS, dan x86. QEMU dapat digunakan untuk mengemulasi *firmware* ECU kendaraan atau unit IVI yang diekstrak dari perangkat keras target, sehingga memungkinkan analisis dinamis, *fuzzing*, dan penelitian kerentanan tanpa perlu akses fisik ke perangkat tersebut. Emulasi berbasis QEMU sangat berguna untuk penilaian tahap awal saat perangkat keras belum tersedia.
- **Ghidra** adalah kerangka kerja rekayasa balik (*reverse engineering*) sumber terbuka yang dikembangkan oleh NSA. Alat ini mendukung pembongkaran (*disassembly*) dan dekompilasi statis terhadap file biner yang dikompilasi untuk berbagai arsitektur prosesor, termasuk ARM, MIPS, dan TriCore. Penguji keamanan dapat menggunakannya untuk menganalisis *firmware* ECU yang telah diekstrak, memulihkan alur kontrol (*control flow*) dan batasan fungsi dalam file biner yang telah dihapus simbol debug-nya (*stripped binaries*), serta mengidentifikasi rutinitas penguraian (*parsing*) yang rentan atau pemeriksaan keamanan yang dinonaktifkan dalam komponen perangkat lunak otomotif.

- **IDA Pro** adalah perangkat lunak komersial untuk pembongkaran (*disassembly*) multi-prosesor dan debugging interaktif. Alat ini mendukung analisis file biner yang dapat dieksekusi untuk berbagai arsitektur, termasuk x86, ARM Cortex-M, dan TriCore. Penguji keamanan dapat menggunakan IDA Pro untuk menganalisis *firmware* ECU, *pustaka native* (*native libraries*) pada aplikasi pendamping, serta file biner *bootloader*, guna mengidentifikasi fungsi pustaka yang sudah dikenal, memulihkan nama variabel, dan memetakan alur kontrol dalam file biner produksi yang telah dihapus simbol debug-nya.
- **Frida** adalah perangkat instrumentasi dinamis yang menyuntikkan agen berbasis skrip ke dalam proses yang sedang berjalan. Perangkat ini memungkinkan *function hooking* (pengait fungsi), pemeriksaan memori, dan modifikasi saat runtime tanpa perlu mengompilasi ulang binary target, dengan dukungan khusus untuk Android, Linux, dan QNX. Seorang penguji keamanan dapat menggunakan Frida untuk melakukan hooking pada rutinitas certificate pinning dalam aplikasi pendamping kendaraan, mencegat panggilan fungsi kriptografi pada unit IVI (*In-Vehicle Infotainment*), atau memantau komunikasi antarproses (*inter-process communication*) pada lingkungan HPC (*High-Performance Computing*) berbasis QNX saat runtime.
- **radare2** adalah kerangka kerja reverse engineering sumber terbuka (*open-source*) yang menyediakan lingkungan berbasis baris perintah (*command-line*) untuk analisis binary, disassembly, debugging, dan pembuatan skrip di berbagai arsitektur. Penguji keamanan dapat menggunakannya untuk menganalisis *binary firmware* otomotif di lingkungan dengan sumber daya terbatas, menyusun alur kerja analisis kustom melalui skrip, serta melakukan validasi silang terhadap temuan dari Ghidra pada arsitektur yang memiliki dukungan plugin komunitas terbatas.

Fuzzing dan Pengujian Stres (*Stress Testing*) Kernel/Sistem

Setelah *firmware*, biner, driver, atau antarmuka sistem diidentifikasi, penguji dapat menggunakan teknik *fuzzing* dan pengujian stres untuk mengeksekusi jalur kode yang sulit dijangkau secara manual. Alat-alat berikut sering digunakan untuk mencapai tujuan tersebut:

- **AFL++** merupakan turunan (*fork*) dari AFL yang menawarkan kecepatan lebih tinggi, opsi mutasi yang lebih beragam, serta pilihan instrumentasi yang lebih luas. Alat ini menjadi pilihan populer untuk *fuzzing* berbasis cakupan (*coverage-guided fuzzing*) pada komponen userland, selama mekanisme *harnessing* (penghubung untuk pengujian) dapat dibuat.
- **Syzkaller** adalah kernel fuzzer berbasis cakupan yang beroperasi secara mandiri (tanpa pengawasan). Awalnya dikembangkan untuk kernel Linux, alat ini kini mendukung berbagai kernel lain, sehingga relevan untuk kernel Linux otomotif (serta driver khusus perangkat) yang digunakan dalam HPC (*High-Performance Computers*).

Alat yang Diperkuat Dengan AI

Penerapan AI paling matang dalam pekerjaan yang melibatkan ECU berkinerja tinggi (*high-performance ECU*), di mana AI membantu mengatasi dua hambatan utama yang memakan banyak biaya, yaitu proses triase/pemahaman biner serta pembuatan *fuzz harness*:

- **BinaryAI** adalah platform analisis keamanan yang menggunakan algoritma AI untuk menganalisis biner. Platform ini menyediakan kemampuan seperti penguraian (*parsing*) dan analisis komposisi perangkat lunak (*software composition analysis*), serta menawarkan plugin untuk alur kerja berbasis Ghidra atau IDA. Dalam praktiknya, alat ini dapat mempercepat jawaban atas pertanyaan seperti "apa isi *firmware* ini?" dan "apakah komponen ini digunakan di berbagai varian produk?", yang secara langsung memengaruhi potensi eksploitasi dan cakupan perbaikan (*patch*).
- **VulHunt** adalah kerangka kerja analisis kerentanan biner yang dirancang untuk menghadirkan analisis semantik pada tingkat kode ke dalam biner. Alat ini dapat diintegrasikan dengan model bahasa besar (*Large Language Models* atau LLM) untuk keperluan triase dan pencarian kerentanan. Hal ini sangat relevan dalam industri otomotif karena citra (*image*) *firmware* pada HPC sering kali memiliki ketergantungan (*dependency*) yang kompleks dan kerap menyertakan komponen dari pihak ketiga. Analisis biner semantik tentu sangat berguna untuk mempersingkat waktu mulai dari ekstraksi *firmware* hingga penemuan kerentanan.
- **oss-fuzz-gen** menghasilkan target *fuzzing* untuk proyek nyata dengan memanfaatkan berbagai LLM, yang kemudian diuji kinerjanya melalui OSS-Fuzz. Dokumentasi OSS-Fuzz juga menyebutkan bahwa pembuatan target berbasis LLM dapat mengurangi upaya integrasi manual.
- **CI Fuzz Spark** menggunakan alur kerja otomatis untuk membuat dan menjalankan pengujian *fuzzing*. Dalam praktiknya, ini merupakan salah satu penerapan AI yang paling dapat dipertanggungjawabkan: pembuatan dan validasi *fuzz harness* (perangkat penguji *fuzzing*) memakan banyak waktu, sehingga otomatisasi yang mampu menghasilkan harness siap pakai memiliki nilai guna langsung.

Tabel berikut merangkum berbagai perangkat representatif yang digunakan dalam penilaian sistem berbasis HPC maupun ECU berbasis MCU. Perangkat-perangkat ini mendukung alur kerja yang mencakup penemuan antarmuka perangkat keras dan perolehan *firmware*, hingga rekayasa balik (*reverse engineering*), instrumentasi saat waktu jalan (*runtime instrumentation*), *fuzzing*, pengujian stres kernel, triase biner, serta validasi fisik melalui teknik side-channel atau injeksi kesalahan (*fault-injection*):

Tabel 3.4: Alat keamanan ofensif untuk HPC dan MCU

Alat	Tujuan	Tahap Penggunaan	Target	Referensi
JTAGulator	Menemukan antarmuka debug bergaya JTAG/UART	Penemuan	ECU HPC dengan titik uji (*test point*) yang terbuka	Halaman JTAGulator; slide
BukaOCD	Melakukan debugging dan pemrograman dalam sistem (*in-system programming*) melalui JTAG/SWD	Sarana Penemuan/Eksploitasi	SoC tertanam (*embedded SoC*) dan adaptor debug yang didukung	Panduan pengguna OpenOCD.

SEGGE™ J-Link	Ekosistem perangkat probe debug dan peralatan pendukung	Sarana Penemuan/Eksploitasi	ECU berbasis ARM dan akses debug	Dokumentasi SEGGER J-Link
Binwalk v3	Identifikasi struktur *firmware* dan dukungan ekstraksinya	Penemuan	Blob *firmware* ECU dan citra pembaruan (*update image*)	Repositori Binwalk v3.
Ghidra	Analisis statis dan dekompilasi untuk kode yang telah dikompilasi	Penemuan	File biner ECU pada berbagai platform umum	Repositori/rilis Ghidra.
Jumat	Instrumentasi saat waktu berjalan (*runtime*) dan validasi perilaku	Validasi	Layanan userland Android/Linux/QNX	Dokumentasi Frida.
AFL++	*Fuzzing* berbasis cakupan (*coverage-guided*) untuk komponen userland	Validasi	Parser dan layanan yang menghadap ke jaringan	Repositori/situs AFL++.
syzkaller	*Fuzzing* kernel berbasis cakupan	Validasi	Kernel dan driver Linux	Repositori/dokumentasi syzkaller.
VulHunt	Analisis kerentanan biner secara semantik; triase dengan bantuan LLM	Penemuan/ Validasi	File biner dan pustaka ECU yang diekstraksi	Postingan/situs Binarly VulHunt.
OSS-Fuzz-gen / CI Fuzz AI Test Agent	Pembuatan fuzz harness dan otomatisasi *fuzzing* dengan bantuan LLM	Validasi	Komponen perangkat lunak ECU yang cocok untuk *fuzzing*	Repositori OSS-Fuzz-gen; halaman LLM OSS-Fuzz; dokumentasi CI.
*flash*rom	Membaca/menulis/verifikasi cip *flash*; mendukung perangkat pemrogram eksternal	Penemuan	Penyimpanan *firmware* dan memori *flash* SPI eksternal	Halaman manual/README *flash*rom.
Bus Pirate	Antarmuka serial multi-protokol (I2C/SPI/UART, dll.)	Penemuan	Bus periferal pada PCB ECU	Dokumentasi Bus Pirate.
Perangkat lunak Saleae Logic	Dekode protokol untuk bus sistem tertanam (*embedded buses*) (UART/I2C/SPI, dll.)	Penemuan	Penangkapan sinyal dan dekoding bus pada PCB	Dokumentasi protokol Saleae.
Pengendali Chip	Side-channel dan injeksi galat (daya/glitching)	Validasi	MCU yang terlindungi; penelitian mengenai pintasan (*bypass*) boot/autentikasi	Memulai penggunaan ChipWhisperer.
AI Biner	Pemahaman biner dan analisis komposisi dengan bantuan AI	Penemuan/Triage	Berkas biner ECU yang telah diekstraksi	Dokumentasi BinaryAI; repositori plugin.

Setelah membahas ECU, mulai dari unit komputasi pusat (*central compute*) hingga mikrokontroler (MCU), kini kita beralih ke lapisan sensor. Pada lapisan ini, aspek keamanan ofensif mencakup ranah RF/fisika serta ketahanan sistem pembelajaran mesin (ML)/persepsi.

3.5 PERANGKAT KEAMANAN OFENSIF PADA LAPISAN SENSOR KENDARAAN

Kelompok perangkat terakhir yang kita bahas dalam bab ini adalah alat yang berkaitan dengan sensor, yang sering ditemukan dalam pekerjaan terkait kendaraan terkoneksi (*connected vehicle*). Sistem sensor kendaraan menghadirkan permukaan serangan yang unik; ancamannya bukan berupa layanan yang dapat diakses melalui jaringan, melainkan lingkungan sinyal fisik. Oleh karena itu, diperlukan alat yang mampu menghasilkan, menangkap, atau memanipulasi input berbasis RF, optik, akustik, maupun lapisan AI untuk menilai integritas sensor dan ketahanan alur persepsi (*perception pipeline*).

Perbedaan penting di sini adalah bahwa "alat sensor" beroperasi di batas antara manipulasi pada lapisan perangkat lunak dan dampaknya di dunia fisik contohnya, penggunaan *adversarial patch* (tambalan yang dirancang khusus untuk mengecoh AI) pada rambu jalan yang menyebabkan sistem persepsi berbasis kamera melakukan klasifikasi yang salah. Dengan demikian, kedua dimensi serangan tersebut harus dipertimbangkan saat mengevaluasi temuan. Seperti yang akan kita lihat pada Bab 9, banyak sensor canggih mengadopsi arsitektur yang serupa dengan ECU berbasis MCU; untuk perangkat semacam itu, alat pengujian penetrasi yang telah dibahas di bagian sebelumnya tetap dapat digunakan. Bagian ini mencakup alat untuk menangkap dan mendekode sinyal RF dari sistem sensor kendaraan, melakukan *replay* (pengulangan) serta injeksi transmisi pada pita frekuensi di bawah 1 GHz (sub-GHz), dan menerapkan teknik berbasis AI untuk menganalisis lapisan sensor.

Penangkapan Sinyal RF, Analisis Protokol, dan Injeksi

Sistem sensor kendaraan seperti pemantau tekanan ban, penerima sinyal *keyless entry*, dan modul penyalaan mesin jarak jauh mengandalkan protokol RF pada pita frekuensi di bawah 1 GHz (sub-GHz). Dalam aplikasi otomotif, pita frekuensi yang umum digunakan adalah 315 MHz dan 433 MHz, yang sering kali bersifat eksklusif (*proprietary*) dan memiliki dokumentasi yang sangat terbatas. Sinyal-sinyal ini kerap dikirimkan tanpa perlindungan kriptografi atau hanya menggunakan implementasi *rolling-code* yang lemah, sehingga serangan jenis replay dan injeksi menjadi sangat mungkin untuk dilakukan.

Perangkat dalam bagian ini mencakup fase penemuan (*discovery*) maupun eksploitasi dalam pengujian RF: penangkapan sinyal secara pasif dan dekode protokol dapat menentukan apa yang sedang ditransmisikan serta bagaimana struktur sinyal tersebut, sementara platform yang memiliki kemampuan transmisi memungkinkan penguji keamanan untuk memutar ulang (*replay*) atau menyuntikkan (*inject*) bingkai data (*frame)* yang telah dirancang khusus guna menguji validasi penerima. Platform *software-defined radio* (SDR) kini memungkinkan kedua peran tersebut dilakukan tanpa memerlukan peralatan laboratorium khusus. Berikut adalah perangkat utama untuk penangkapan sinyal dan dekode protokol:

- **rtl_433** berfungsi untuk mendekode transmisi radio dari perangkat yang beroperasi pada pita frekuensi ISM (dan frekuensi lainnya). Perangkat ini relevan dalam konteks otomotif, khususnya untuk sensor dan perangkat telemetri yang menggunakan pita frekuensi umum (seperti sistem yang menyerupai TPMS). Kemampuan untuk melihat "apa yang sedang ditransmisikan" merupakan langkah awal sebelum hipotesis keamanan apa pun dapat diuji.
- **™TL-SD™ V3** adalah penerima SDR berbasis USB yang terjangkau, mencakup rentang frekuensi sekitar 500 kHz hingga 1,75 GHz, dan berfungsi sebagai front-end RF untuk dekode sinyal berbasis perangkat lunak. Jika dipadukan dengan rtl_433, penguji keamanan dapat menangkap siaran sensor TPMS dari kendaraan target secara pasif, lalu secara otomatis mendekode pengenal sensor, tekanan ban, suhu, dan status baterai semuanya tanpa memerlukan akses ke jaringan internal kendaraan. Hal ini memungkinkan analisis pemalsuan (*spoofing*) sensor serta penelitian mengenai pelacakan kendaraan.
- **Universal ™adio Hacker (U™H)** merupakan perangkat lunak untuk merekam, menganalisis, melakukan demodulasi, dan mendekode sinyal frekuensi radio yang ditangkap oleh penerima SDR. Dalam konteks sensor, seorang penguji keamanan (*pentester*) dapat menggunakan URH untuk menganalisis transmisi TPMS dan sinyal RKE yang ditangkap dari kendaraan target. Alat ini mampu mengungkap parameter modulasi, struktur paket, dan penyandian bit (*bit encoding*) dari protokol pita ISM yang bersifat eksklusif, bahkan tanpa memerlukan pengetahuan awal mengenai spesifikasinya. URH melengkapi rtl_433 dengan menyediakan lingkungan analisis protokol interaktif untuk sinyal-sinyal yang tidak dapat didekode secara otomatis oleh rtl_433.
- **GNU ™adio** adalah kerangka kerja (*framework*) pemrosesan sinyal sumber terbuka (*open-source*) yang mengimplementasikan protokol radio melalui perangkat lunak, dengan menggunakan model aliran data berbasis blok. GNU Radio berfungsi sebagai lingkungan eksekusi (*runtime*) untuk modul dekoder khusus kendaraan, seperti gr-tpms, dan memungkinkan pembuatan rangkaian penerima (receiver chain) kustom untuk protokol sensor pada pita frekuensi ISM yang tidak didukung oleh rtl_433 atau URH. Seorang penguji keamanan dapat menyusun flowgraph GNU Radio untuk memproses sampel I/Q yang ditangkap melalui perangkat seperti HackRF atau RTL-SDR, kemudian mengubahnya menjadi data sensor yang telah didekode.
- **gr-tpms** adalah modul out-of-tree untuk GNU Radio yang berfungsi sebagai penerima dan dekoder untuk transmisi RF TPMS (sistem pemantau tekanan ban) yang beroperasi pada pita frekuensi ISM 315 MHz dan 433 MHz. Penguji keamanan dapat menggunakan flowgraph gr-tpms dengan perangkat keras seperti RTL-SDR V3 atau HackRF One untuk menangkap siaran sensor TPMS dari kendaraan target. Melalui proses ini, penguji dapat memperoleh informasi seperti pengenal sensor, pembacaan tekanan ban, suhu, dan status baterai dari paket yang didekode, tanpa memerlukan akses ke jaringan internal kendaraan.

- **HackRF One** adalah perangkat transceiver SDR pita lebar (*wideband*) yang mencakup rentang frekuensi 1 MHz hingga 6 GHz. Perangkat ini mendukung operasi penerimaan maupun pengiriman sinyal, sehingga pengujian TPMS tidak terbatas pada penangkapan sinyal secara pasif saja. Penguji keamanan dapat menggunakannya bersama rtl_433 untuk menerima sinyal TPMS, lalu mengirimkan bingkai (*frame*) TPMS yang telah dimanipulasi pada frekuensi ISM yang tepat. Hal ini bertujuan untuk menguji apakah penerima pada kendaraan target akan menerima paket sensor yang dikirim ulang (*replay*) atau dipalsukan (*spoofed*) tanpa adanya validasi kebaruan data (*freshness validation*).
- **Yard Stick One** adalah perangkat transceiver SDR untuk rentang frekuensi di bawah 1 GHz (sub-GHz) yang beroperasi pada pita ISM 300 MHz hingga 928 MHz. Dengan Yard Stick One, penguji keamanan tidak hanya dapat menerima transmisi TPMS dan RKE (*Remote Keyless Entry*), tetapi juga dapat mengirimkan bingkai data hasil tangkapan sebelumnya (*replay frames*) pada frekuensi sub-GHz yang sesuai. Langkah ini memungkinkan penguji untuk memastikan apakah sistem penerima pada kendaraan target menerapkan mekanisme perlindungan terhadap serangan pengulangan (*replay attack*).

Perangkat yang Diperkuat Dengan AI

Penguatan AI pada lapisan sensor (*sensor-layer AI augmentation*) secara mendasar berbeda dengan penguatan AI pada sisi *backend* atau ECU. Dalam konteks ini, AI sering kali hadir di dalam sistem target (pada tahap persepsi dan fusi data), sekaligus dalam perangkat yang digunakan oleh penguji. Oleh karena itu, perangkat ofensif yang paling praktis adalah utilitas evaluasi *adversarial machine learning* (ML):

- **ART (*Adversarial Robustness Toolbox*)** adalah pustaka Python untuk keamanan ML yang dikelola oleh yayasan LF AI & Data. Pustaka ini secara eksplisit mendukung evaluasi terhadap berbagai ancaman seperti *evasion* (pengelabuan), *poisoning* (peracunan data), *extraction* (ekstraksi model), dan *inference* (inferensi). Penguji keamanan dapat menggunakan ART pada model persepsi otomotif misalnya, jaringan untuk deteksi objek atau klasifikasi lajur guna menghasilkan input adversarial. Input ini bertujuan untuk menguji apakah model dapat dikelabui secara andal dalam kondisi yang mencerminkan data sensor di dunia nyata.
- **Counterfit** merupakan lapisan otomatisasi generik untuk menilai keamanan sistem pembelajaran mesin serta mengintegrasikan berbagai kerangka kerja adversarial. Perangkat ini sangat berguna ketika lingkup penilaian mencakup model ML dalam alur persepsi, sistem ADAS yang disimulasikan, atau endpoint penyajian model (*model-serving endpoints*) yang digunakan dalam lingkungan pengembangan dan validasi.
- **Foolbox** adalah pustaka Python untuk menyusun contoh adversarial terhadap model pembelajaran mesin. Pustaka ini menggunakan API konsisten yang mengabstraksi berbagai metode serangan dan kerangka kerja model, termasuk PyTorch, TensorFlow, dan JAX. Dalam konteks otomotif, seorang pentester dapat menggunakan Foolbox untuk membuat input adversarial yang menargetkan komponen ADAS berbasis visi

seperti detektor pejalan kaki berbasis kamera atau pengklasifikasi rambu lalu lintas untuk menguji apakah perubahan kecil pada gambar input dapat menyebabkan kesalahan klasifikasi, tanpa memerlukan akses ke data pelatihan model tersebut. Foolbox melengkapi ART dengan menyediakan rangkaian algoritma serangan yang lebih luas di berbagai kerangka kerja.

- **Garak** adalah pemindai kerentanan LLM (*Large Language Model*) sumber terbuka yang memeriksa model bahasa untuk mendeteksi berbagai kegagalan, seperti prompt injection, jailbreaking, halusinasi, dan ekstraksi data, melalui kerangka kerja pengujian terstruktur. Penguji keamanan dapat menggunakan Garak untuk menilai implementasi asisten AI di dalam kendaraan maupun layanan LLM yang terhubung ke cloud, guna melihat apakah sistem tersebut rentan terhadap input adversarial yang dapat menyebabkan model melanggar batasan keamanan atau mengungkapkan konten prompt sistem yang bersifat sensitif.

Tabel berikut merangkum perangkat-perangkat yang umum digunakan untuk menilai permukaan serangan yang berkaitan dengan sensor kendaraan. Perangkat-perangkat ini mencakup visibilitas RF untuk sinyal sensor dan telemetri, eksperimentasi SDR yang terkendali, serta validasi pembelajaran mesin (ML) terhadap ancaman (*adversarial ML*) untuk komponen persepsi, fusi, dan elemen pembelajaran mesin lainnya yang digunakan dalam sistem kendaraan modern:

Tabel 3.5: Perangkat sensor kendaraan

Alat	**Tujuan**	**Tahap Penggunaan**	**Target**	**Referensi**
Hack™F / US™P	Platform SDR untuk transmisi/penerimaan RF bagi laboratorium TPMS/RF	Penemuan/Validasi	TPMS dan eksperimen RF lainnya yang terkait dengan sensor	Halaman HackRF; manual UHD/USRP
rtl_433	Dekode transmisi perangkat pada pita ISM	Penemuan	Visibilitas sensor/telemetri pada pita ISM	Repositori rtl_433
A™T	Perangkat evaluasi *machine learning* (ML) untuk skenario serangan (seperti evasion, poisoning, dll.)	Validasi	Model ML untuk persepsi/fusi dan komponen ML	Repositori/rilis ART
Barang Palsu	Lapisan otomatisasi untuk pengujian keamanan ML di berbagai kerangka kerja (*framework*)	Validasi	Sistem ML serta alur kerja evaluasinya	Repositori *Counter*fit; catatan rilis Microsoft

Sebelum mengakhiri pembahasan mengenai perangkat pengujian penetrasi (*penetration testing*), penting untuk menyoroti peran kecerdasan buatan (AI) dalam perangkat keamanan

ofensif. AI mengubah lanskap mengenai apa saja yang dapat dilakukan, seberapa mendalam suatu pengujian dapat dilaksanakan, serta seberapa cepat tujuan pengujian dapat dicapai.

3.6 PENERAPAN AI DALAM PENGUJIAN KEAMANAN OFENSIF KENDARAAN

Dampak praktis AI yang paling signifikan saat ini terkonsentrasi pada bidang-bidang yang didominasi oleh teks, struktur, dan skala besar: alur kerja *backend*/API, triase biner pada *Electronic Control Unit* (ECU) yang sarat perangkat lunak, serta pembuatan *fuzz harness* untuk basis kode yang luas. Burp AI merupakan contoh yang tepat di lapisan *backend*; alat ini terintegrasi ke dalam alur kerja proxy yang umum digunakan dan dirancang khusus untuk mempercepat pemahaman serta mengurangi kesalahan manusia, tanpa menerapkan "otonomi yang selalu aktif" (*always-on autonomy*).

Pada lapisan perangkat lunak ECU, VulHunt dan literatur terkait menekankan analisis semantik terhadap biner serta membahas integrasi *Large Language Model* (LLM) untuk keperluan triase dan perburuan kerentanan pendekatan yang sangat relevan dengan realitas *firmware* yang memiliki banyak dependensi. Dalam konteks *fuzzing*, OSS-Fuzz-gen dan Spark AI secara langsung menjawab tantangan hambatan dalam pembuatan harness: OSS-Fuzz-gen memanfaatkan LLM untuk menghasilkan target *fuzzing* dan melakukan tolok ukur (*benchmarking*), sementara dokumentasi CI Fuzz Spark menjelaskan pembuatan pengujian *fuzzing* secara otomatis. Hal ini menghadirkan konsekuensi nyata bagi pihak pertahanan: jika komponen perangkat lunak kendaraan suatu organisasi pada prinsipnya dapat di-fuzz, penggunaan harness berbasis AI akan mempersingkat waktu yang dibutuhkan untuk memulai *fuzzing* maupun menemukan kerentanan. Kondisi ini mengubah dinamika ekonomi keamanan ofensif, bahkan tanpa perlu adanya exploit primitive yang baru.

AI juga membawa perubahan dalam pengujian nirkabel (*wireless*), meskipun dalam cakupan yang lebih terbatas. OmniSIG dari DeepSig merepresentasikan kategori "AI untuk pengukuran frekuensi radio (RF)" seperti deteksi dan klasifikasi sinyal dari aliran data I/Q di mana AI dapat mempercepat proses triase dalam lingkungan spektrum yang padat. Namun, semakin dalam alur kerja melibatkan aspek waktu yang ketat, batasan lapisan fisik (*PHY constraints*), serta perilaku spesifik perangkat, maka peran AI lebih berfungsi sebagai lapisan analisis pendukung daripada sebagai penggerak utama dalam eksploitasi.

Terakhir, dalam konteks sensor dan sistem otonom, AI mengubah esensi dari pertanyaan mengenai pengujian ofensif itu sendiri. ART dan *Counter*fit secara eksplisit menargetkan ancaman terhadap pembelajaran mesin (ML), seperti *evasion* (pengelabuan) dan *extraction* (ekstraksi), serta mengotomatiskan alur kerja evaluasi yang bersifat adversarial. Ini merupakan pergeseran yang signifikan: pengujian ofensif kini berkembang dari sekadar bertanya "bisakah saya memengaruhi sensor?" menjadi "bisakah saya mengevaluasi secara sistematis apakah sistem persepsi/fusi berbasis ML akan gagal di bawah gangguan yang realistis?". Pendekatan ini lebih dekat dengan analisis keamanan sistem (*safety analysis*) dibandingkan dengan keamanan aplikasi (*appsec*) konvensional.

Aspek di mana teknologi AI masih belum matang juga sama pentingnya. Ekstraksi fisik pada mikrokontroler (MCU), pembajakan kunci debug (*debug lock bypass*), dan injeksi

kesalahan (fault injection) tingkat laboratorium masih sangat bergantung pada instrumentasi, akses fisik, dan keahlian khusus untuk perangkat tertentu. Dokumentasi ChipWhisperer sendiri menekankan pada teknik glitching tegangan/jam (*clock*) serta analisis saluran samping (*side-channel*) jenis tugas yang tetap sangat bersifat fisik. Dapat disimpulkan bahwa "gelombang AI" akan membawa perubahan terbesar pada lapisan yang sudah menyerupai perangkat lunak, sementara perubahan terkecil akan terjadi pada lapisan yang dibatasi oleh hukum fisika.

3.7 PENGUJIAN KEAMANAN OFENSIF KENDARAAN TERHUBUNG

Dalam bab ini, kami meninjau ekosistem perangkat keamanan ofensif yang mendukung pengujian penetrasi di seluruh tumpukan teknologi kendaraan terhubung (*connected vehicle*). Alih-alih menyajikan perangkat secara terpisah, kami mengorganisasikannya berdasarkan lapisan arsitektur dan memetakannya ke dalam dua fase operasional yang lazim dalam sebagian besar penugasan di dunia nyata: penemuan (*discovery*) serta eksploitasi/validasi. Perspektif ini penting karena efektivitas pengujian penetrasi jarang bergantung pada satu perangkat canggih saja; sebaliknya, efektivitas tersebut muncul dari orkestrasi sistematis berbagai perangkat yang secara bertahap mengubah sistem yang semula tidak transparan menjadi sistem yang dapat diamati, diukur, dan pada akhirnya, diuji.

Di seluruh bagian sistem *backend* kendaraan, antarmuka nirkabel, jaringan di dalam kendaraan, platform komputasi ECU, hingga subsistem sensor, terdapat pola yang konsisten. Perangkat untuk fase penemuan berfokus pada visibilitas: melakukan enumerasi infrastruktur yang terbuka, memetakan protokol, mengekstraksi artefak *firmware*, menangkap lalu lintas data, serta merekonstruksi alur kerja. Perangkat untuk fase eksploitasi dan validasi kemudian memanfaatkan visibilitas tersebut untuk menguji hipotesis mengenai kelemahan sistem baik dengan menyusun input tertentu, memanipulasi status protokol, melakukan *fuzzing* pada implementasi, maupun memicu perilaku abnormal dalam kondisi yang terkendali. Dengan demikian, alur kerja seorang penguji penetrasi menyerupai proses ilmiah: penguji melakukan observasi, merumuskan hipotesis, menguji hipotesis tersebut, lalu memvalidasi hasilnya. Perangkat-perangkat yang digunakan berfungsi sebagai instrumen yang memungkinkan eksperimen yang semakin presisi terhadap sistem.

Sebagaimana akan terlihat dalam bab-bab selanjutnya, pengujian keamanan otomotif mencakup berbagai domain rekayasa secara bersamaan. Pengujian *backend* menyerupai penilaian keamanan cloud dan API pada umumnya. Pengujian nirkabel melibatkan batasan frekuensi radio dan eksperimen pada lapisan fisik (*physical layer*). Pengujian jaringan di dalam kendaraan menuntut kemampuan rekayasa balik protokol dan manipulasi pada tingkat bus data. Analisis ECU mencakup rekayasa balik sistem tertanam (*embedded systems*) serta analisis *firmware*. Terakhir, pekerjaan pada lapisan sensor menggabungkan keamanan siber dengan pemrosesan sinyal dan ketahanan pembelajaran mesin (*machine learning*). Hanya sedikit bidang lain yang menuntut praktisinya untuk mampu bekerja dengan luwes di berbagai teknologi yang begitu beragam. Oleh karena itu, membangun perangkat keamanan ofensif yang mumpuni bukan sekadar mengumpulkan berbagai tools, melainkan mengembangkan

kemampuan untuk mengintegrasikannya ke dalam alur kerja koheren yang melintasi batasan perangkat keras, perangkat lunak, dan jaringan.

Dalam bab ini, kita juga telah menelaah bagaimana kecerdasan buatan (AI) mulai mengubah berbagai aspek dalam alur kerja ini. Saat ini, AI memberikan manfaat praktis terbesar di bidang-bidang yang memiliki data terstruktur dalam jumlah melimpah, seperti analisis API, triase biner, dan pembuatan fuzz harness. Dalam konteks tersebut, sistem AI berperan sebagai akselerator yang mempersingkat waktu yang dibutuhkan untuk memahami komponen perangkat lunak yang kompleks serta menghasilkan input pengujian dalam volume besar. Namun, lapisan yang didominasi oleh antarmuka fisik seperti debugging mikrokontroler (MCU), injeksi kesalahan (*fault injection*), atau eksperimen frekuensi radio (RF) masih sangat bergantung pada instrumentasi dan kondisi laboratorium. Di lingkungan seperti itu, AI lebih berfungsi untuk membantu proses analisis daripada menggantikan eksperimen langsung. Oleh karena itu, memahami di mana AI dapat meningkatkan efektivitas pengujian secara signifikan dan di mana teknik rekayasa konvensional tetap tak tergantikan merupakan keterampilan penting bagi seorang penguji penetrasi (*penetration tester*) modern.

Pada bab-bab selanjutnya, kita akan mulai menerapkan perangkat-perangkat ini dalam skenario serangan nyata di seluruh arsitektur kendaraan. Alih-alih memperkenalkan perangkat baru, fokus akan beralih pada penyusunan rantai serangan yang realistis, mengorelasikan temuan di berbagai lapisan, serta menunjukkan bagaimana kelemahan yang ditemukan selama tahap pengintaian (*reconnaissance*) dapat berkembang menjadi kompromi keamanan yang serius. Dengan demikian, perangkat yang disajikan dalam bab ini menjadi landasan bagi eksperimen ofensif yang akan dibahas berikutnya.

BAGIAN 2
EKSPLOITASI ANTARMUKA EKSTERNAL DAN JARINGAN KENDARAAN

Bagian ini beralih dari teori ke praktik, memeriksa permukaan serangan yang dapat dijangkau secara eksternal yang mengekspos kendaraan modern kepada musuh jarak jauh. Anda akan mempelajari cara menyelidiki dan mengeksploitasi layanan *backend* cloud, antarmuka komunikasi nirkabel, dan protokol jaringan dalam kendaraan, mulai dari lapisan konektivitas terluar hingga ke tulang punggung komunikasi kendaraan. Setiap bab menggabungkan pemeriksaan kelemahan umum dengan teknik pengujian penetrasi yang ditargetkan dan studi kasus dunia nyata yang diambil dari insiden OEM besar. Pada akhir bagian ini, Anda akan memiliki keterampilan praktis untuk menilai dan menantang keamanan tumpukan konektivitas yang mendasari kendaraan yang terhubung.

Bagian ini memiliki bab-bab berikut:

- Bab 4, “Mengatasi Gangguan Visual”
- Bab 5, “Mengatasi Sistem Visual”
- Bab 6, “Mengatasi Komunikasi Internal”

BAB 4
SERANGAN TERHADAP BACKEND KENDARAAN

Kendaraan modern yang terhubung (*connected vehicles*) kini semakin bergantung pada sistem *backend*. Hal ini terjadi seiring dengan meningkatnya kebutuhan pelanggan akan kemudahan akses jarak jauh secara langsung terhadap data dan fungsi kendaraan mereka. Baik itu menerima pembaruan perangkat lunak melalui udara (*over-the-air*/OTA) di rumah, alih-alih harus pergi ke dealer atau membuka kunci mobil dari jarak jauh agar teman bisa mengambil barang dari kursi belakang, layanan *backend* kendaraan terus menjadi nilai jual yang menarik bagi kendaraan modern. Layanan ini tidak hanya menguntungkan konsumen, tetapi juga membantu produsen kendaraan (*Original Equipment Manufacturers*/OEM) dalam menghemat biaya seperti pengurangan biaya layanan di dealer serta memungkinkan pengumpulan data berskala besar (*big data*) yang memberikan wawasan berharga mengenai performa pengemudi maupun kendaraan.

Sayangnya, seperti hampir semua fitur yang menawarkan kemudahan melalui konektivitas yang lebih luas, terdapat konsekuensi tersembunyi terhadap keamanan kendaraan. Mengaktifkan layanan jarak jauh secara otomatis memperluas permukaan serangan (*attack surface*) kendaraan hingga mencakup sistem *backend*. Mengingat sifatnya yang terbuka, fokus kita akan tertuju secara khusus pada arsitektur *backend* kendaraan berbasis cloud yang pada dasarnya merupakan arsitektur *cloud Internet of Things* (IoT) yang disesuaikan untuk kebutuhan kendaraan. Dalam bab ini, kita akan mulai dengan memahami dasar-dasar arsitektur IoT pada *backend* kendaraan beserta protokol komunikasi yang paling umum digunakan. Selanjutnya, kita akan meninjau kelemahan-kelemahan yang sering ditemukan pada sistem ini sebagai landasan untuk memahami teknik eksploitasi. Kemudian, kita akan membahas berbagai perangkat dan teknik pengujian penetrasi (*penetration testing*) yang relevan, sebelum beralih ke studi kasus nyata di mana *backend* kendaraan berhasil disusupi, yang pada akhirnya berdampak langsung pada kendaraan tersebut. Terakhir, kita akan menguraikan praktik terbaik keamanan untuk meningkatkan ketahanan sistem *backend* dengan menerapkan pendekatan *zero-trust*. Bagian selanjutnya dari bab ini dibagi menjadi topik-topik utama berikut:

- Gambaran umum arsitektur *backend* kendaraan
- Memahami protokol konektivitas *backend*
- Kelemahan umum dalam arsitektur *backend* kendaraan
- Teknik pengujian penetrasi (*penetration testing*)
- Studi kasus serangan nyata terhadap *backend* kendaraan
- Praktik terbaik untuk mengamankan infrastruktur *backend* kendaraan

4.1 GAMBARAN UMUM ARSITEKTUR BACKEND KENDARAAN

Arsitektur *backend* kendaraan modern biasanya memanfaatkan platform cloud untuk menghubungkan kendaraan (sebagai perangkat edge) dengan layanan *backend*. Infrastruktur

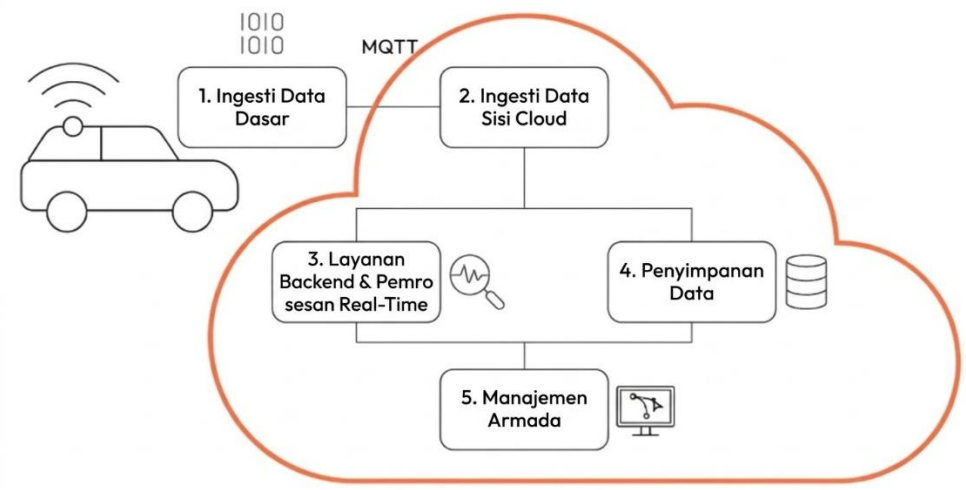
seperti *Microsoft Azure dan Amazon Web Services* (AWS) memungkinkan komunikasi dua arah untuk data telematika, perintah jarak jauh ke kendaraan, analitik, serta integrasi dengan sistem bisnis. Salah satu alasan meningkatnya penggunaan *backend* kendaraan berbasis cloud adalah beragam layanan yang dapat disediakan, antara lain:

- **Pengumpulan dan pemantauan data telematika**: Pengunggahan berkelanjutan telemetri kendaraan (lokasi GPS, kecepatan, pembacaan sensor, status mesin, dan sebagainya) ke cloud. Hal ini mendukung pemantauan kendaraan secara real-time, dasbor manajemen armada, dan analitik pemeliharaan prediktif.
- **Pembaruan OTA (*Over-the-Air*)**: Layanan untuk mengirimkan pembaruan perangkat lunak atau perubahan konfigurasi dari cloud ke *Electronic Control Unit* (ECU) kendaraan.
- **Pelatihan kecerdasan buatan (AI)**: Pengumpulan data gambar berlabel untuk memungkinkan pelatihan model pembelajaran mesin (*machine learning*/ML) terkait skenario mengemudi tertentu.
- **Perintah dan kendali jarak jauh**: Kemampuan bagi pengguna atau operator armada untuk mengirimkan perintah ke kendaraan melalui aplikasi seluler atau portal web. Perintah yang umum meliputi penguncian/pembukaan kunci pintu, menyalakan/mematikan mesin, membunyikan klakson, menyalakan lampu, mengaktifkan sistem pengatur suhu (*climate control*), serta melacak lokasi kendaraan.
- **Pembaruan status jarak jauh**: Integrasi dengan aplikasi seluler pendamping dan layanan pihak ketiga memungkinkan aplikasi tersebut menerima informasi status kendaraan (seperti tingkat bahan bakar atau baterai, tekanan ban), menerima peringatan, menjadwalkan janji temu servis, atau bahkan melakukan pembelian fitur tambahan.

Sebagaimana terlihat dari daftar kasus penggunaan tersebut, layanan-layanan ini menciptakan peluang bagi penyerang untuk menimbulkan dampak serius terkait keselamatan, kerugian finansial, gangguan operasional, serta pelanggaran privasi jika sistem berhasil disusupi. Baik itu membuka kunci kendaraan seseorang dari jarak jauh untuk tujuan pencurian, melacak lokasi korban, maupun mengirimkan pembaruan perangkat lunak berbahaya, kerentanan keamanan pada sistem *backend* dapat dan pasti akan menimbulkan konsekuensi yang merugikan.

Sebagai contoh, perhatikan skenario pengumpulan dan pengiriman data pelatihan kembali ke cloud. Penyerang yang mampu memalsukan informasi tersebut berpotensi meracuni model pembelajaran mesin (*machine learning*/ML), yang pada gilirannya dapat menyebabkan bahaya keselamatan. Untuk menjamin keamanan layanan-layanan ini, aspek keamanan harus dipertimbangkan secara menyeluruh dalam solusi *end-to-end* mulai dari sistem *backend* hingga ke komponen kendaraan terkecil yang menjadi bagian dari layanan tersebut. Pada bab-bab selanjutnya dalam buku ini, kita akan mempelajari segmen-segmen individual dari alur komunikasi ini, namun untuk saat ini, fokus kita sepenuhnya tertuju pada sisi *backend* itu sendiri. Sebelum membahas teknik-teknik eksploitasi terhadap *backend*, kita perlu memahami komponen-komponen penyusun utama serta layanan komunikasi yang

membentuk arsitektur *backend* kendaraan, sebagaimana diilustrasikan dan dijelaskan dalam Gambar 4.1.



Gambar 4.1 Lima komponen penyusun utama arsitektur *backend* kendaraan terhubung (*connected vehicle*)

Terdapat lima komponen penyusun utama yang dijelaskan sebagai berikut:

1. **Pemasukan data dasar (*foundational data ingestion*) – integrasi MQTT dan telematika**: Di sisi kendaraan, data ditangkap dan dikurasi menggunakan sistem telematika internal atau ECU *gateway*. Biasanya, telemetri di dalam kendaraan termasuk performa mesin, keluaran sensor, dan status Sistem Bantuan Pengemudi Lanjutan (*Advanced Driver Assistance Systems*/ADAS) dikirimkan melalui protokol ringan seperti *Message Queuing Telemetry Transport* (MQTT), yang dipilih karena efisiensinya saat digunakan pada tautan jaringan dengan keterbatasan sumber daya.
2. **Pemasukan data di sisi cloud dan orkestrasi berbasis peristiwa (*event-driven orchestration*)**: Di sisi *backend*, broker MQTT berbasis cloud menerima pesan-pesan tersebut dan memprosesnya untuk didistribusikan. Artinya, setelah data mencapai cloud, layanan pemasukan data khusus (dalam Azure, sering kali berupa IoT Hub atau lapisan pemasukan data berbasis MQTT) akan memverifikasi sumber data, menerapkan aturan protokol, serta meneruskan peristiwa (*event*) yang relevan ke inti sistem. Pada tahap ini, perute peristiwa atau event router (seperti Event Grid) berperan untuk mendistribusikan notifikasi tersebut. Dengan mengadopsi arsitektur berbasis peristiwa, para insinyur dapat memisahkan subsistem satu sama lain (*decoupling*), sehingga layanan baru dapat mengakses aliran peristiwa yang sama tanpa perlu mengubah alur pemasukan data (*ingestion pipeline*).
3. **Layanan backend dan pemrosesan waktu nyata (*real-time*)**: Setelah data masuk (*ingestion*), pelanggan peristiwa (*event subscriber*) seperti Azure Functions atau layanan *backend* kustom akan mengurai data yang mengalir tersebut. Dengan menggunakan aturan yang telah ditetapkan sebelumnya, sistem dapat menilai kondisi untuk memicu peringatan (misalnya, anomali tekanan ban) atau menjalankan fungsi

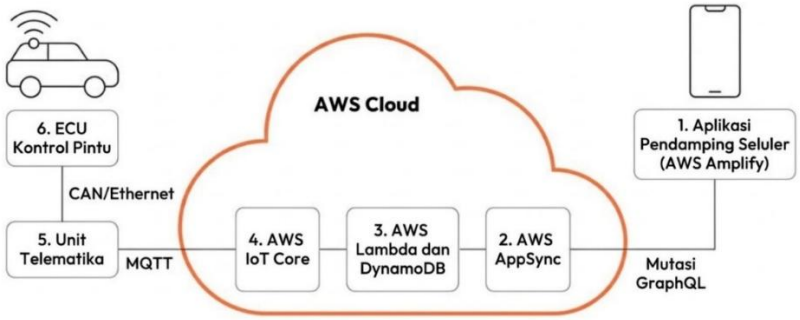
pemrosesan data untuk mengubah data ke dalam format pesan yang diinginkan. Dengan mencocokkan sinyal mentah ini terhadap ambang batas atau aturan yang telah ditentukan, sistem dapat memberikan umpan balik kepada pengguna, seperti peringatan mengenai peristiwa yang krusial bagi keselamatan.

4. **Penyimpanan data untuk wawasan dan akuntabilitas**: Selain notifikasi waktu nyata, terdapat kasus penggunaan yang bergantung pada penyimpanan peristiwa secara permanen (*persistent storage*). Lapisan penyimpanan permanen ini mencakup dua hal berikut:
   - **Basis data operasional (misalnya, Azure Cosmos DB)**: Menyimpan telematika terkini atau pembaruan status, yang memungkinkan pencarian data untuk keperluan diagnostik, pelacakan penggunaan, atau dasbor yang dapat diakses oleh pengguna.
   - **Data lake / penyimpanan dingin (*cold storage*) (misalnya, Azure Data Lake)**: Menyimpan data historis kendaraan dalam volume besar untuk mendukung analitik tingkat lanjut, pemeliharaan prediktif (*predictive maintenance*), dan investigasi pasca-insiden.
5. **Manajemen armada dan visualisasi telemetri**: Setelah arsitektur ini terbentuk, portal manajemen armada atau dasbor OEM dapat mengagregasi dan menampilkan wawasan yang diperoleh dari pemrosesan waktu nyata maupun hasil pemeriksaan data yang tersimpan. Dengan memanfaatkan aliran data waktu nyata, log data, dan deteksi anomali berbasis pembelajaran mesin (*machine learning*), aplikasi-aplikasi ini membantu mengidentifikasi potensi kerusakan, mengoordinasikan pembaruan *firmware* melalui udara (OTA), serta memberikan gambaran menyeluruh mengenai kondisi armada.

Berdasarkan pemantauan singkat di atas, kita dapat mulai memahami proposisi nilai dari paradigma komputasi baru ini, baik bagi OEM maupun pengguna.

Contoh Kasus Penggunaan: Membuka Kunci Kendaraan

Pada bagian ini, kita akan membahas lebih mendalam mengenai detail teknis tentang bagaimana kasus penggunaan pembukaan kunci kendaraan dapat diwujudkan dengan menggunakan AWS *Connected Vehicle Reference Architecture*, sebagaimana ditunjukkan dalam Gambar 4.2.



Gambar 4.2 Arsitektur referensi kendaraan terhubung AWS yang menunjukkan alur pesan menyeluruh (diadaptasi dari)

Proses ini berlangsung dengan tahapan sebagai berikut:

1. Aplikasi pendamping seluler mengirimkan perintah buka kunci kendaraan jarak jauh yang aman melalui AWS Amplify sebagai mutasi GraphQL ke AWS AppSync.
2. AWS AppSync menerima mutasi tersebut dan meneruskannya ke lapisan pemrosesan *backend*.
3. Perintah ini disimpan di Amazon DynamoDB, yang memicu fungsi AWS Lambda untuk memeriksa status konektivitas kendaraan saat ini menggunakan *device shadow* (bayangan perangkat) yang dikelola di AWS IoT Core.
4. Jika kendaraan dalam keadaan daring (*online*), fungsi Lambda akan memublikasikan perintah buka kunci ke AWS IoT Core dengan pengidentifikasi permintaan unik dan topik respons MQTT yang telah ditentukan.
5. Unit telematika kendaraan, yang mempertahankan koneksi MQTT persisten ke AWS IoT Core, menerima perintah tersebut dan bersiap untuk menjalankan operasi buka kunci.
6. Unit telematika mengirimkan pesan *Controller Area Network* (CAN) atau Ethernet yang sesuai ke *Electronic Control Unit* (ECU) pengontrol pintu, yang kemudian secara fisik melakukan proses buka kunci.

Setelah menjalankan perintah, kendaraan memublikasikan pesan respons ke topik respons MQTT yang telah ditentukan. Respons ini diproses oleh fungsi Lambda di sisi *backend*, yang kemudian memperbarui status perintah dan status kendaraan di DynamoDB. Terakhir, status yang telah diperbarui dikirimkan kembali ke aplikasi seluler melalui langganan (*subscription*) AWS AppSync, sehingga pengguna dapat menerima konfirmasi hampir secara waktu nyata (*near real-time*) bahwa kendaraan telah berhasil dibuka kuncinya.

Istilah:

- AWS AppSync adalah layanan terkelola yang membantu Anda membangun dan menjalankan API GraphQL untuk mengakses serta memperbarui data di cloud, sementara AWS Amplify adalah perangkat (*toolkit*) yang mempermudah integrasi layanan tersebut beserta layanan cloud lainnya ke dalam aplikasi seluler atau web Anda.
- GraphQL adalah bahasa kueri untuk API yang memungkinkan klien meminta data yang mereka butuhkan, sehingga interaksi dengan server menjadi lebih fleksibel dibandingkan dengan *Application Programming Interface* (API) berbasis *Representational State Transfer* (REST) tradisional.
- Mutasi (*mutation*) adalah operasi GraphQL yang dikirim oleh klien untuk meminta *backend* melakukan suatu tindakan atau mengubah status sistem.
- *Device shadow* (bayangan perangkat) pada dasarnya merupakan representasi digital dari kendaraan terkait di dalam cloud. Ini berupa dokumen *JavaScript Object Notation* (JSON) yang menyimpan status terakhir kendaraan yang diketahui, serta pembaruan status yang diinginkan.
- Fungsi AWS Lambda adalah layanan komputasi tanpa server (*serverless*) yang memungkinkan Anda menjalankan kode kustom sebagai respons terhadap peristiwa tertentu seperti operasi GraphQL tanpa perlu mengelola server apa pun.

- DynamoDB adalah layanan basis data NoSQL yang dikelola sepenuhnya oleh AWS, yang menyimpan data dalam bentuk model pasangan kunci-nilai (*key-value*) dan dokumen.

Jelas sekali bahwa pengantar singkat ini tidak cukup untuk membekali Anda dalam melakukan rangkaian pengujian penetrasi (*penetration test*) yang efektif terhadap sistem *backend* berbasis cloud. Terdapat berbagai buku yang membahas cara melakukan pengujian keamanan pada sistem *backend* Azure, AWS, dan Google; Anda sangat disarankan untuk mempelajari topik tersebut secara lebih mendalam jika hal itu menjadi fokus pekerjaan Anda. Setelah arsitektur *backend* kendaraan ditetapkan sebagai landasan, bagian selanjutnya akan mengulas protokol konektivitas yang menentukan bagaimana kendaraan, aplikasi seluler, dan layanan *backend* saling berkomunikasi satu sama lain.

4.2 MEMAHAMI PROTOKOL KONEKTIVITAS BACKEND

Layanan *backend* tidak akan mungkin terwujud tanpa protokol konektivitas yang menghubungkan *backend* dengan kendaraan melalui gerbang komunikasi (yang biasanya disebut sebagai *Telematics Control Unit* atau TCU). Kendaraan mengandalkan jaringan seluler untuk menjaga koneksi internet ke cloud menggunakan modem 4G/LTE/5G. Selama bertahun-tahun, berbagai protokol telah digunakan untuk memungkinkan konektivitas antara kendaraan dan *backend*; protokol-protokol tersebut akan kita bahas dalam subbagian berikut.

API HTTP/REST

Banyak layanan kendaraan menggunakan *Application Programming Interface* (API) berbasis *Hypertext Transfer Protocol* (*Secure*) (HTTP(S)), terutama untuk interaksi yang dimulai dari cloud ke kendaraan atau sebaliknya melalui perantara. Sebagai contoh, kendaraan mungkin secara berkala mengirimkan permintaan POST berbasis *Representational State Transfer* (REST) yang berisi data telemetri dalam bentuk kelompok (*batch*). Lebih umum lagi, aplikasi seluler atau aplikasi web menggunakan API REST untuk berkomunikasi dengan *backend* di cloud (misalnya untuk proses masuk pengguna atau pengiriman perintah yang kemudian diteruskan oleh cloud ke kendaraan).

Sebuah API *gateway* di sisi *backend* berfungsi mengarahkan permintaan API yang masuk ke layanan *backend* yang dituju. Komponen ini juga dapat menangani caching (penyimpanan sementara) atas permintaan dan respons guna mengurangi beban pada layanan *backend*. Biasanya, API *gateway* juga menangani autentikasi dan otorisasi menggunakan *Open Authorization* (OAuth) dan JSON Web Token (JWT). Selain itu, fitur keamanan yang umum diterapkan adalah whitelisting alamat *Internet Protocol* (IP), yaitu membatasi akses ke layanan *backend* hanya untuk alamat IP tertentu. Langkah ini sangat bijak mengingat ada layanan-layanan tertentu yang seharusnya hanya dapat diakses oleh kendaraan atau perangkat yang terdaftar, bukan oleh seluruh pengguna internet.

WebSocket

Beberapa layanan real-time menggunakan koneksi WebSocket (yang menyediakan komunikasi *full-duplex* melalui satu koneksi *Transmission Control Protocol* (TCP)). Sebagai contoh, kendaraan dapat menjaga koneksi WebSocket ke endpoint di cloud untuk menerima

pengiriman peristiwa (*event*) secara instan. WebSocket atau protokol serupa juga dapat digunakan untuk transmisi data telemetri secara berkelanjutan (misalnya, untuk dasbor telemetri langsung).

SMS dan Sistem Telepon Lama

Sistem telematika yang lebih lama terkadang masih menggunakan *Short Message Service* (SMS) untuk perintah-perintah tertentu, misalnya saat mengirim pesan untuk membangunkan (*wake-up*) kendaraan. Namun, protokol ini kini dianggap sebagai teknologi lama (*legacy*), karena sebagian besar kendaraan modern yang terhubung lebih memilih menggunakan data berbasis IP untuk seluruh komunikasi.

AMQP, XMPP, dan CoAP

Advanced Message Queuing Protocol (AMQP) didukung oleh Azure IoT Hub sebagai alternatif untuk MQTT/HTTP. Beberapa sistem otomotif mungkin menggunakan AMQP atau bahkan *Extensible Messaging and Presence Protocol* (XMPP) secara historis, namun MQTT kini lebih banyak digunakan karena sifatnya yang ringan. Protokol lain dalam IoT adalah *Constrained Application Protocol* (CoAP), meskipun protokol ini lebih umum ditemukan pada perangkat IoT dengan sumber daya terbatas dan jarang ditemui pada kendaraan.

Protokol Diagnostik

Diagnostik jarak jauh, termasuk diagnostik prediktif, merupakan tren yang diadopsi secara luas di kalangan produsen peralatan asli (OEM). *Diagnostics over* IP (DoIP), yang distandarisasi sebagai ISO 13400, telah umum digunakan dalam perangkat diagnostik dan memungkinkan diagnostik jarak jauh melalui *gateway*; teknologi ini sudah matang dan didukung secara luas oleh unit kontrol elektronik (ECU). Namun, untuk mendukung solusi yang lebih berbasis cloud (*cloud-native*), industri kini beralih ke *Service-Oriented Vehicle Diagnostics* (SOVD), yaitu API diagnostik bergaya REST yang distandarisasi sebagai ISO 17978. SOVD menawarkan berbagai keunggulan karena ramah terhadap cloud (menggunakan HTTP/HTTPS), bersifat stateless, lebih mudah diamankan (misalnya, dengan menggunakan OAuth 2.0 dan mutual *Transport Layer Security* atau mTLS), serta mampu diskalakan untuk seluruh armada kendaraan.

MQTT Sebagai Standar De Facto

Di antara protokol komunikasi yang disebutkan di atas, MQTT telah muncul sebagai protokol standar de facto untuk komunikasi IoT dan kendaraan terhubung (*connected car*). MQTT adalah protokol pesan berbasis *publish/subscribe* yang ringan, sangat cocok untuk perangkat dengan lebar pita (*bandwidth*) terbatas atau konektivitas yang tidak stabil. *Telematics Control Unit* (TCU) pada kendaraan menjalankan klien MQTT yang menjaga koneksi mTLS persisten ke broker cloud di sistem *backend* otomotif, seperti AWS IoT Core atau Azure IoT Hub.

Data telemetri dipublikasikan ke topik-topik tertentu (sering kali menyertakan ID kendaraan dalam nama topiknya), sementara kendaraan dapat berlangganan (*subscribe*) ke topik perintah untuk menerima instruksi kendali jarak jauh. Desain MQTT mampu menangani jaringan yang tidak andal dengan cara mengantre pesan serta mendukung berbagai tingkat

Quality of Service (QoS). Untuk memahami lebih mendalam bagaimana MQTT digunakan, mari kita pantau alur umum yang melibatkan pesan MQTT:

1. Pertama, TCU (*Telematics Control Unit*) pada kendaraan terhubung ke broker MQTT di cloud (contoh: *vehicle-iot.cloud.com*) menggunakan sertifikat klien miliknya.
2. TCU tersebut memublikasikan pesan seperti {"tirePressure": 32, "fuelLevel": 50} ke topik vehicles/<VIN>/telemetry.
3. Setelah menerima pesan ini, sistem cloud akan memproses dan menyimpan data tersebut, serta dapat memicu peringatan jika ditemukan nilai yang tidak wajar.

Dalam contoh lain saat pengguna menekan tombol "Buka Kunci Mobil" (*Unlock Car*) pada aplikasi seluler, sistem *backend* akan memublikasikan pesan MQTT ke topik vehicles/<VIN>/commands/unlock. TCU yang berlangganan (*subscribe*) ke topik tersebut akan menerima perintah dan kemudian melakukan tindakan membuka kunci pada kendaraan. Seluruh pertukaran data ini harus diautentikasi dan dienkripsi dengan benar guna mencegah pesan yang tidak sah atau pesan palsu (*spoofed*). Sudah sewajarnya bahwa semua komunikasi antara kendaraan dan cloud harus diamankan. Praktik terbaik dalam industri mengharuskan penggunaan enkripsi *Transport Layer Security* (TLS) untuk data yang sedang dikirimkan, serta autentikasi yang kuat pada *gateway* konektivitas kendaraan.

Dalam praktiknya, banyak platform menggunakan mTLS (mutual TLS): TCU biasanya menyimpan sertifikat perangkat, dan broker di cloud mewajibkan kendaraan untuk melakukan autentikasi menggunakan sertifikat tersebut, sementara kendaraan juga memverifikasi sertifikat server cloud. Sebagai contoh, AWS mewajibkan penggunaan mTLS dengan sertifikat X.509 bagi kendaraan yang terhubung ke AWS IoT Core. Selain itu, token autentikasi atau kunci API (*API key*) dapat digunakan untuk panggilan API tertentu (misalnya, endpoint REST mungkin memerlukan token akses OAuth 2.0 yang dikaitkan dengan pengguna atau kendaraan terkait).

Istilah:

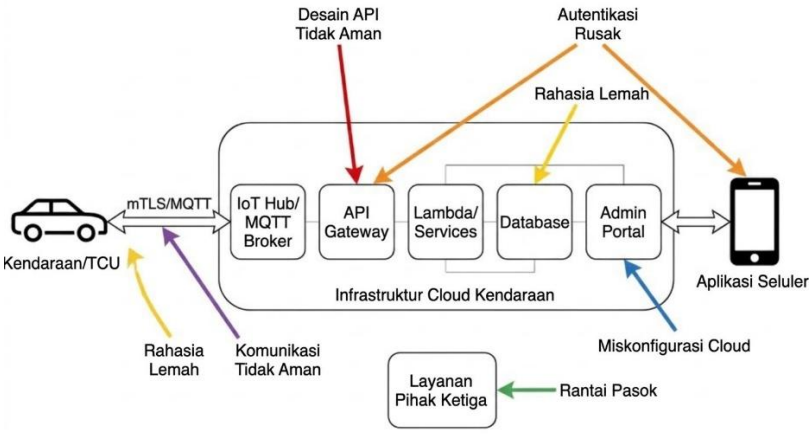
- Sertifikat X.509 adalah sertifikat digital yang mengaitkan kunci publik dengan identitas perangkat, server, atau pengguna, serta ditandatangani oleh otoritas sertifikat (*Certificate Authority*/CA) tepercaya untuk memungkinkan autentikasi dalam komunikasi yang aman. Sertifikat X.509 digunakan secara luas dalam protokol seperti TLS/mTLS untuk memfasilitasi autentikasi, enkripsi, dan kepercayaan antar-sistem yang berkomunikasi, misalnya antara TCU (*Telematic Control Unit*) kendaraan dan layanan cloud.
- Token akses OAuth 2.0 adalah token dengan masa berlaku singkat yang diterbitkan oleh server otorisasi, yang memungkinkan suatu aplikasi untuk mengakses API yang terlindungi atas nama pengguna atau layanan tanpa perlu membagikan kredensial.

Seperti yang telah kita lihat sejauh ini, kendaraan merupakan perangkat IoT yang bergerak di atas roda dan bergantung pada layanan cloud untuk mendapatkan fungsionalitas tambahan. Pembaca yang tertarik untuk mempelajari lebih lanjut tentang cara membangun aplikasi IoT otomotif yang aman disarankan untuk menelusuri sumber daya ini. Bagian selanjutnya akan membahas kelemahan keamanan yang dapat muncul dalam arsitektur *backend* kendaraan tersebut.

4.3 KELEMAHAN UMUM DALAM ARSITEKTUR BACKEND KENDARAAN

Sistem kendaraan terhubung (*connected vehicle*) mewarisi banyak kerentanan yang juga ditemukan pada aplikasi IoT dan aplikasi web. *Backend* cloud pada kendaraan pada dasarnya merupakan kumpulan API web dan endpoint IoT; oleh karena itu, daftar risiko keamanan API teratas dari Open *Worldwide Application Security Project* (OWASP) serta beberapa kelemahan umum menurut MITRE sangatlah relevan. Selain itu, penerapan pendekatan manajemen kerentanan yang tepat guna menyediakan kerangka kerja praktis untuk melakukan triase, pelacakan, dan perbaikan kerentanan menjadi semakin penting bagi organisasi di sektor otomotif.

Dalam beberapa tahun terakhir, berbagai insiden keamanan menunjukkan bahwa kendaraan tidak hanya rentan melalui akses fisik, tetapi juga melalui serangan terhadap layanan cloud dan API. Faktanya, sebuah studi tahun 2023 menemukan peningkatan sebesar 380% dalam insiden yang berkaitan dengan API di sektor otomotif dibandingkan tahun-tahun sebelumnya, yang menyoroti bagaimana kelemahan pada *backend* kini menjadi target utama. Kami menyajikan beberapa kategori kelemahan keamanan umum dalam arsitektur *backend* kendaraan, sebagaimana diilustrasikan dalam Gambar 4.3 dan dijelaskan sebagai berikut:



Gambar 4.3 Gambaran umum kelemahan yang sering ditemukan dalam arsitektur *backend* kendaraan

Desain API yang Tidak Aman dan Kelemahan Dalam Validasi Input

Banyak layanan *backend* kendaraan pada dasarnya merupakan aplikasi web (seperti portal atau API yang menerima parameter). Jika input ini tidak dibersihkan (*sanitasi*), kerentanan injeksi klasik seperti injeksi SQL (*Structured Query Language*) atau injeksi perintah dapat terjadi. Contoh nyata terlihat pada kasus platform telematika Spireon, di mana kerentanan injeksi SQL pada portal administrasi berbasis cloud memungkinkan peneliti untuk mendapatkan akses administrator. Daftar "*OWASP Top 10 API Security Risks*" mencantumkan injeksi dan pengelolaan aset yang tidak tepat sebagai masalah umum, dan insiden di sektor otomotif mengonfirmasi hal ini. Input yang tidak divalidasi dapat memungkinkan penyerang untuk menjalankan kode yang tidak sah pada server *backend* atau memanipulasi data. Sebagai

contoh, parameter yang disaring dengan buruk dapat dimanfaatkan oleh penyerang untuk menyisipkan perintah SQL guna mengekstrak kredensial pengguna.

Sebagaimana telah dijelaskan sebelumnya, kendaraan terhubung (*connected vehicles*) sangat bergantung pada layanan web dan API di cloud (untuk aplikasi seluler, portal web, dll.), yang menjadikannya permukaan serangan utama dalam sebagian besar arsitektur. API ini biasanya menyediakan titik akhir (*endpoint*) yang terkait dengan VIN (*Vehicle Identification Number*), pengenal kendaraan, akun pengguna, dan pengelompokan armada. Ketika desain API gagal memvalidasi input atau membersihkan parameter dengan benar, penyerang dapat memanipulasi permintaan untuk mengakses atau memengaruhi kendaraan di luar cakupan wewenang mereka.

Validasi yang tidak memadai terhadap kolom VIN, ID kendaraan, atau pengenal akun dapat menyebabkan akses data lintas kendaraan, eksekusi perintah jarak jauh tanpa izin, atau terungkapnya data telemetri sensitif seperti riwayat lokasi dan informasi diagnostik. Selain itu, pembatasan laju permintaan (*rate limiting*) yang tidak memadai dapat memfasilitasi serangan enumerasi VIN berskala besar. Kelemahan dalam logika bisnis menjadi sangat krusial dalam sistem otomotif; penyerang dapat mengeksploitasi asumsi alur kerja dalam pemrosesan perintah jarak jauh, logika kontrol pengisian daya, atau mekanisme penjadwalan pembaruan perangkat lunak melalui udara (OTA). Berbeda dengan layanan web konvensional, celah keamanan pada API *backend* kendaraan dapat berdampak operasional yang meluas ke seluruh populasi kendaraan.

Kontrol Autentikasi dan Otorisasi yang Rusak

Sistem *backend* kendaraan mengelola berbagai domain identitas: pengguna akhir, operator armada, teknisi dealer, layanan *backend*, dan kendaraan itu sendiri. Kelemahan autentikasi dapat mencakup alur masuk aplikasi seluler yang diimplementasikan secara tidak tepat, pembuatan token yang tidak aman, autentikasi multi-faktor yang tidak memadai, atau pengelolaan sesi dengan validasi yang buruk. Token yang terkait dengan kepemilikan kendaraan atau peran dalam armada harus memiliki cakupan yang ketat dan dilindungi secara kriptografis. Autentikasi yang tidak tepat dapat menyebabkan akses tidak sah ke API atau layanan kendaraan. Dalam beberapa kasus sebelumnya, API tidak memiliki autentikasi atau mengandalkan pengidentifikasi yang mudah ditebak. Sebagai contoh, API *backend* yang melayani aplikasi pendamping Nissan Leaf diketahui menerima perintah jika VIN (nomor identifikasi kendaraan) yang tepat dimasukkan tanpa memerlukan kata sandi. Ini merupakan contoh API dengan autentikasi buruk yang memungkinkan penyerang memicu fungsi tertentu, seperti pengaturan suhu kabin, dari jarak jauh.

Otorisasi yang rusak (misalnya, *Insecure Direct Object References* atau referensi objek langsung yang tidak aman) merupakan aspek lain di mana API gagal memverifikasi dengan benar apakah pengguna atau perangkat yang mengajukan permintaan memiliki izin untuk mengakses data atau sumber daya kendaraan tertentu. Hal ini dapat menyebabkan akun satu pelanggan mengakses kendaraan milik akun lain hanya dengan mengubah pengidentifikasi dalam permintaan. Artinya, jika API *backend* mengandalkan pengidentifikasi kendaraan yang diberikan oleh klien tanpa memvalidasi ulang kepemilikan atau penugasan armada, penyerang

dapat mengambil data dari atau mengirim perintah ke kendaraan yang bukan milik mereka atau tidak berada di bawah kendali mereka. Selain itu, sistem dealer menghadirkan kompleksitas tambahan, karena akun layanan mungkin memerlukan hak akses yang lebih tinggi untuk keperluan diagnostik atau pembaruan *firmware*. Tanpa segmentasi peran yang ketat, penyerang dapat meningkatkan akses mereka dari tingkat konsumen menjadi tingkat dealer atau administrator armada. Dalam lingkungan kendaraan terhubung berskala besar, kelemahan otorisasi merupakan salah satu risiko sistemik yang paling serius.

Pengelolaan Rahasia atau Kredensial yang Lemah

Ekosistem *backend* otomotif bergantung pada hierarki rahasia yang kompleks, termasuk kunci autentikasi telematika, sertifikat penandatanganan untuk pembaruan melalui udara (*Over-the-Air* atau OTA), token API, dan kredensial akun layanan. Praktik pengelolaan rahasia yang lemah seperti menyematkan kata sandi atau kunci API secara permanen (*hardcoded*) di dalam unit kontrol telematika (TCU) atau aplikasi seluler, menggunakan kembali kredensial statis di berbagai layanan, atau tidak melakukan rotasi kunci penandatanganan dapat merusak integritas seluruh platform kendaraan. Salah satu kasus yang mencolok melibatkan perangkat pelacak GPS MiCODUS MV720 yang digunakan pada kendaraan: para peneliti menemukan bahwa perangkat tersebut beserta API cloud-nya menggunakan kata sandi bawaan "123456", bahkan terdapat kata sandi utama (*master password*) yang tertanam secara permanen (*hardcoded*) yang ternyata masih digunakan oleh 94% perangkat yang diuji. Hal ini berarti penyerang dapat dengan mudah melakukan autentikasi ke sistem *backend* dengan menyamar sebagai perangkat mana pun.

Kunci API atau kredensial rahasia yang tertanam dalam aplikasi seluler atau *firmware* juga menimbulkan risiko serupa; jika penyerang berhasil mendapatkan kredensial tersebut, mereka dapat berpura-pura menjadi kendaraan atau server yang bersangkutan. Selain itu, terkait sistem *backend* otomotif, pernah terjadi kasus di mana kontraktor secara tidak sengaja membocorkan kredensial. Sebagai contoh, pada tahun 2022, sebuah repositori kode sumber yang diunggah ke GitHub ditemukan berisi kunci yang tertanam secara permanen, yang memberikan akses ke basis data berisi informasi pelanggan. Karena sistem otomotif sering kali beroperasi dalam skala global, kredensial yang terekspos dapat memicu risiko kompromi berantai yang berdampak pada armada kendaraan dalam jumlah besar atau bahkan seluruh lini produksi.

Komunikasi yang Tidak Aman

Kendaraan yang terhubung (*connected vehicles*) terus-menerus bertukar data telemetri, data diagnostik, dan konfirmasi perintah jarak jauh dengan infrastruktur *backend* kendaraan. Oleh karena itu, jalur komunikasi antara kendaraan dan *backend* yang biasanya melalui jaringan seluler atau Wi-Fi menjadi salah satu vektor serangan potensial, meskipun enkripsi modern membuat upaya ini jauh lebih sulit. Kelemahan dalam komunikasi dapat mencakup implementasi TLS yang salah konfigurasi, penggunaan algoritma enkripsi (*cipher*) yang lemah, validasi sertifikat yang tidak tepat, atau tidak adanya autentikasi timbal balik (*mutual authentication*) antara kendaraan dan layanan *backend*. Jika komunikasi antara kendaraan dan *backend* tidak dienkripsi dan diautentikasi dengan benar, penyerang dapat

mencegat atau memalsukan pesan. Sistem mobil terhubung generasi awal sempat memiliki masalah seperti tidak diterapkannya HTTPS atau tidak dilakukannya validasi terhadap sertifikat TLS. Kendaraan modern umumnya menggunakan TLS dan sering kali beroperasi melalui *Access Point Name* (APN) privat (jaringan seluler tertutup), yang membuat upaya intersepsi menjadi lebih sulit; namun, kesalahan konfigurasi seperti menerima sertifikat apa pun atau menggunakan algoritma enkripsi yang sudah usang dapat melemahkan keamanan sistem.

Celah pada Infrastruktur Cloud

Platform *backend* kendaraan biasanya dibangun di atas infrastruktur cloud berskala besar yang mendukung pengumpulan data telematika, dasbor armada, orkestrasi pembaruan perangkat lunak melalui udara (OTA), serta sistem layanan dealer. Celah keamanan yang umum ditemukan pada *backend* berbasis cloud meliputi kesalahan konfigurasi, perangkat lunak yang belum diperbarui (*unpatched*), kontrol akses internal yang lemah, serta berbagai kerentanan dalam infrastruktur cloud itu sendiri. Salah satu contoh kejadian ini terjadi pada sebuah produsen otomotif, di mana sistem *Single Sign-On* (SSO) yang dikonfigurasi secara tidak tepat memberikan akses ke ratusan aplikasi internal. Melalui celah ini, para peneliti berhasil mengakses perangkat internal yang sangat krusial (seperti build server, alur *Continuous Integration*/*Continuous Delivery* (CI/CD), basis data internal, dan lain-lain) bahkan mampu menjalankan kode dari jarak jauh (r*emote code execution*) pada beberapa sistem *backend*.

Hal ini menunjukkan bahwa *backend* kendaraan bukan sekadar satu server tunggal, melainkan sebuah ekosistem yang mencakup basis data berisi informasi pelanggan, perangkat pengembangan, dan banyak lagi. Perlu dicatat bahwa penyerang yang menemukan kelemahan dalam infrastruktur TI perusahaan seperti antarmuka administratif yang terbuka atau penggunaan kredensial bawaan pada layanan cloud dapat beralih untuk mendapatkan kendali yang lebih dalam. Mengingat *backend* otomotif sering kali melayani jutaan kendaraan secara bersamaan, kesalahan konfigurasi infrastruktur dapat memicu kerentanan sistemik yang berdampak luas, alih-alih sekadar menjadi insiden yang terisolasi.

Kelemahan dalam Rantai Pasok dan Integrasi Pihak Ketiga

Produsen otomotif sering kali mengandalkan penyedia pihak ketiga untuk layanan konektivitas, seperti penyedia layanan telematika (contohnya, Sirius XM menyediakan layanan kendaraan terhubung untuk berbagai merek), penyedia data peta atau lalu lintas, platform manajemen armada, bahkan layanan pelat nomor digital. Jika sistem-sistem ini memiliki kerentanan atau menggunakan pustaka (*library*) yang rentan, penyerang dapat mengeksploitasinya untuk menyerang seluruh kendaraan atau pengguna yang bergantung pada layanan tersebut. Alih-alih menyerang setiap mobil secara langsung, penyerang dapat menargetkan penyedia layanan pusat yang dipercaya oleh banyak kendaraan. Ini merupakan vektor serangan yang sangat efektif, yang menegaskan pentingnya melakukan pemeriksaan dan pengamanan terhadap setiap layanan kendaraan terhubung dari pihak ketiga.

Selain itu, integrasi pihak ketiga dalam sistem *backend* kendaraan sering kali mengandalkan token OAuth atau kunci API (*API key*). Jika validasi yang memadai tidak diterapkan atau cakupan akses (*scope*) yang diberikan terlalu luas, penyerang dapat menyusup melalui sistem eksternal tepercaya menuju sistem *backend* inti kendaraan. Setelah meninjau

berbagai kelemahan umum yang memengaruhi sistem *backend* kendaraan, kini kita akan mengeksplorasi teknik pengujian penetrasi (*penetration testing*) yang memanfaatkan kelemahan tersebut untuk menyusup ke dalam sistem kendaraan.

4.4 TEKNIK PENGUJIAN PENETRASI

Pengujian penetrasi pada *backend* kendaraan harus dilakukan sebagai kampanye serangan terstruktur yang mencakup tahap pengintaian (*reconnaissance*), penilaian kerentanan (*vulnerability assessment*), dan eksploitasi. Tahap pengintaian bertujuan untuk mengidentifikasi aset *backend* yang terekspos, API, hubungan kepercayaan (*trust relationships*), jalur komunikasi, dan integrasi pihak ketiga. Selanjutnya, penilaian kerentanan dilakukan untuk menentukan di mana letak kelemahan dalam aspek autentikasi, otorisasi, validasi input, pengelolaan data rahasia (*secret handling*), atau konfigurasi. Terakhir, tahap eksploitasi menunjukkan bagaimana kelemahan-kelemahan tersebut dapat disalahgunakan untuk mengakses data kendaraan, mengirimkan perintah jarak jauh, meningkatkan hak akses (*privilege escalation*), atau menyusup lebih dalam ke dalam ekosistem kendaraan terhubung.

Alur kerja ini menggabungkan pengujian keamanan web dan API konvensional dengan teknik-teknik khusus untuk perangkat IoT. Alat-alat seperti Burp Suite, Postman, Frida, dan Shodan dapat digunakan untuk penilaian sistem yang terhubung ke cloud, sementara perangkat debug perangkat keras (*hardware debugger*), ekstraksi *firmware*, dan alat rekayasa balik (*reverse engineering*) dapat mendukung investigasi yang lebih mendalam terhadap komponen tepercaya di sisi kendaraan, seperti TCU (*Telematics Control Unit*) dan sistem pendukung lainnya. Sebagaimana telah dibahas sebelumnya dalam bab ini, sistem target mencakup layanan cloud, aplikasi seluler, dan sistem tertanam (*embedded systems*) untuk konektivitas kendaraan; oleh karena itu, diperlukan serangkaian teknik dan perangkat yang fleksibel serta beragam. Teknik-teknik berikut paling efektif jika diterapkan dalam urutan yang biasanya dilakukan oleh penyerang: pertama memetakan permukaan serangan, kemudian memvalidasi kelemahan, dan terakhir menunjukkan bagaimana kelemahan tersebut dapat dieksploitasi.

Pengintaian Cloud dan Eksploitasi Kesalahan Konfigurasi

Tahap ini mendukung proses pengintaian dengan mengidentifikasi aset telematika yang terbuka, portal dealer, infrastruktur OTA (*Over-the-Air*), sistem staging, serta batasan kepercayaan (*trust boundaries*) lainnya yang membentuk permukaan serangan (*attack surface*) pada sistem *backend*. Dalam konteks *backend* kendaraan, pengintaian cloud berfokus pada identifikasi server telematika, dasbor armada, platform manajemen OTA, dan portal dealer yang terekspos. Penguji keamanan sering kali memanfaatkan *Open Source Intelligence* (OSINT) untuk menemukan titik akhir (*endpoint*) cloud yang relevan. Alat seperti Shodan, *Censys*, dan SecurityTrails membantu dalam menemukan aset yang dapat diakses publik, misalnya broker IoT yang terbuka pada port MQTT atau portal admin yang terekspos.

Alat enumerasi subdomain seperti Amass dapat membantu mengidentifikasi lingkungan pengembangan atau staging yang mungkin mengekspos API *backend* kendaraan. Sebagai contoh, mencari nama domain atau subdomain yang terkait dengan merek mobil

tertentu atau penyedia layanan telematika (menggunakan pola umum seperti *api.oemname.com, telematics.brand.com*, atau platform yang sudah dikenal) dapat menghasilkan target yang berharga. Selain itu, dokumentasi publik atau forum terkadang secara tidak sengaja mengungkapkan detail API. Sebagai ilustrasi, seorang penguji dapat mencari referensi mengenai endpoint API atau kode milik OEM tertentu melalui GitHub. Hasil dari tahap ini adalah peta aset yang dapat dijangkau disusun berdasarkan prioritas serta titik masuk potensial untuk pengujian yang lebih mendalam.

Pengujian Komunikasi dan Pemeriksaan Protokol

Tahap ini menjembatani fase pengintaian dengan penilaian kerentanan (v*ulnerability assessment*) dengan menunjukkan bagaimana kendaraan, aplikasi seluler, perangkat dealer, dan layanan *backend* melakukan autentikasi serta pertukaran perintah, telemetri, dan pembaruan perangkat lunak. Sistem *backend* kendaraan bergantung pada komunikasi yang aman antara kendaraan, aplikasi seluler, perangkat dealer, dan layanan cloud. Pengujian keamanan komunikasi mengevaluasi konfigurasi TLS, logika validasi sertifikat, dan mekanisme autentikasi timbal balik (*mutual authentication*) yang digunakan dalam komunikasi antara kendaraan dan *backend*. Penguji keamanan dapat menggunakan perangkat seperti Wireshark, tcpdump, sslyze, dan Nmap untuk memeriksa jabat tangan TLS (TLS *handshake*) serta mengidentifikasi konfigurasi protokol yang lemah.

Lebih lanjut, dengan menggunakan Wireshark, penguji dapat mendekode protokol seperti MQTT, HTTP, dan sebagainya, yang dapat mengungkap apakah pesan dienkripsi serta protokol apa saja yang sedang digunakan. Pengujian *man-in-the-middle* dapat dilakukan menggunakan Burp Suite, OWASP ZAP, atau mitmproxy untuk menentukan apakah aplikasi seluler memvalidasi sertifikat *backend* dengan benar. Penguji kemudian dapat menilai apakah serangan *replay* (pengulangan) dimungkinkan terhadap API perintah kendaraan jarak jauh atau titik akhir (*endpoint*) permintaan OTA (*Over-the-Air*).

Jika kendaraan menggunakan, misalnya, MQTT, penguji keamanan dapat menggunakan perangkat klien MQTT (seperti MQTT Explorer atau Eclipse Mosquitto) untuk mencoba melakukan koneksi. Tentu saja, hal ini memerlukan kredensial, namun penguji mungkin bisa mendapatkannya dari perangkat atau aplikasi tersebut. Jika penguji berhasil mengekstrak sertifikat perangkat (mungkin melalui pembongkaran perangkat keras atau eksploitasi pada TCU), mereka dapat mencoba terhubung seolah-olah sebagai kendaraan untuk menguji perilaku broker. Selain itu, penguji dapat menargetkan MQTT dengan melakukan *fuzzing* pada nama topik atau ukuran payload untuk melihat apakah broker tersebut tidak menerapkan kebijakan yang ketat.

Penguji juga dapat menggunakan perangkat antarmuka CAN (seperti CANtact atau perangkat PCAN) untuk mengamati bagaimana TCU berinteraksi dengan jaringan di dalam kendaraan saat menerima perintah dari *backend*. Hal ini dapat mengungkap apakah terdapat perintah yang tidak aman yang diteruskan secara langsung ke bus CAN. Jika TCU mendukung Wi-Fi atau Bluetooth, antarmuka tersebut juga dapat menjadi titik masuk untuk mengakses sistem *backend*. Sebagai contoh, beberapa TCU membiarkan antarmuka perintah AT (AT *command*) dapat diakses melalui SMS atau Bluetooth; celah ini dapat dimanfaatkan oleh

penyerang untuk menyuntikkan pesan CAN atau mengubah pengaturan APN yaitu parameter nama titik akses seluler yang menentukan *gateway* jaringan yang digunakan TCU untuk terhubung ke *backend* kendaraan. Memahami jalur komunikasi ini membantu penguji dalam menentukan titik mana yang rentan terhadap pelanggaran kepercayaan, serangan replay, intersepsi, atau penyamaran selama tahap eksploitasi selanjutnya.

Penemuan Rahasia dan Penyalahgunaan Kredensial

Tahap ini mendukung penilaian kerentanan dengan mengidentifikasi kredensial, kunci, sertifikat, dan token yang mungkin memungkinkan penguji untuk melampaui batasan kepercayaan (*trust boundaries*) yang ditetapkan atau menyamar sebagai komponen yang sah. Ekosistem otomotif sangat bergantung pada kunci kriptografis dan kredensial untuk mengautentikasi kendaraan, aplikasi seluler, TCU (*Telematic Control Unit*), serta infrastruktur OTA (*Over-the-Air*). Pengujian penemuan rahasia berfokus pada identifikasi kata sandi atau kunci API yang tertanam secara permanen (*hard-coded*) dalam aplikasi seluler pendamping, kredensial yang tertanam di dalam TCU, serta token akun layanan di dalam repositori *backend*. Alat pemeriksaan seluler seperti MobSF, Frida, Magisk, Jadx, dan APKTool digunakan untuk memeriksa aplikasi pendamping kendaraan guna mencari titik akhir (*endpoint*) telematika yang terekspos atau rahasia yang tertanam di dalamnya. Rekayasa balik (*reverse engineering*) dapat mengungkap kunci API yang digunakan untuk mengautentikasi layanan perintah kendaraan jarak jauh. Rekayasa balik juga dapat menunjukkan bagaimana sistem menghitung tanda tangan (*signature*) atau nonce untuk permintaan tertentu, apakah algoritma tersebut lemah, serta apakah penguji dapat meniru atau memintas mekanisme tersebut.

Penguji keamanan yang memiliki akses ke kendaraan atau TCU juga dapat melakukan pengujian perangkat keras. Perangkat keras seperti debugger atau antarmuka JTAG (*Joint Test Action Group*)/SWD (*Serial Wire Debug*) memungkinkan ekstraksi (*dumping*) *firmware*. Pemeriksaan *firmware* menggunakan alat seperti Binwalk, Ghidra, atau IDA Pro dapat mengungkap kredensial atau URL yang tertanam di dalamnya. Sebagaimana dibahas lebih lanjut, ekstraksi dan rekayasa balik *firmware* dapat mengungkap rahasia serta asumsi kepercayaan yang menghubungkan sistem di sisi kendaraan dengan *backend*.

Rahasia yang tersimpan di cloud dapat dievaluasi dengan memindai repositori menggunakan alat seperti truffleHog atau GitLeaks, serta dengan meninjau konfigurasi CI/CD untuk mencari kunci penandatanganan OTA atau token layanan *backend* yang terekspos. Penguji juga dapat menilai apakah kredensial *backend* kendaraan dirotasi secara berkala dan dibatasi aksesnya sesuai dengan prinsip hak akses minimum (*least privilege*). Perlu diperhatikan bahwa kebocoran kunci penandatanganan atau token telematika dapat memungkinkan interaksi jarak jauh yang tidak sah terhadap kendaraan dalam skala luas.

Eksploitasi API dan Manipulasi Input

Tahap ini beralih dari penilaian ke eksploitasi dengan memvalidasi apakah fungsionalitas *backend* yang terekspos dapat disalahgunakan melalui validasi input yang lemah, otorisasi tingkat objek yang rusak, atau logika bisnis yang cacat. Eksploitasi API pada *backend* kendaraan berfokus pada endpoint yang berinteraksi dengan identitas kendaraan berbasis VIN (*Vehicle Identification Number*), antarmuka perintah jarak jauh, data perjalanan,

telemetri baterai, dan layanan pembaruan OTA (*Over-the-Air*). Karena banyak antarmuka *backend* kendaraan berupa API, penguji keamanan sebaiknya menggunakan perangkat standar untuk pengujian penetrasi (*pentesting*) aplikasi web. Burp Suite atau perangkat lunak sumber terbuka OWASP ZAP sangat penting untuk mencegat dan memodifikasi panggilan API. Postman berguna untuk menyusun dan mengirim panggilan REST API secara manual dengan header atau body kustom, terutama setelah endpoint diketahui. Penguji dapat mengonfigurasi perangkat seluler atau emulator untuk mengarahkan lalu lintas melalui Burp Suite, sehingga mereka dapat memeriksa permintaan dari aplikasi seluler pendamping, unit kepala (*head unit*) kendaraan, maupun layanan *backend*.

Pengujian sering kali dimulai dengan melakukan enumerasi endpoint telematika dan memeriksa bagaimana VIN, ID kendaraan, serta pengenal akun pengguna ditangani. Validasi parameter VIN yang tidak tepat dapat memungkinkan akses tidak sah ke data kendaraan lintas akun. Burp Intruder atau skrip Python kustom dapat digunakan untuk mengotomatiskan upaya enumerasi VIN atau menguji apakah perintah jarak jauh seperti membuka kunci pintu, mengaktifkan kontrol iklim, atau mengelola pengisian daya dibatasi dengan benar hanya untuk pengguna yang berwenang. Pengujian logika bisnis sangat krusial dalam sistem otomotif. Sebagai contoh, penguji harus mengevaluasi apakah API perintah jarak jauh menerapkan pemeriksaan kepemilikan untuk setiap permintaan, apakah API armada membatasi akses hanya pada kendaraan yang ditugaskan, dan apakah pembatasan laju (*rate limiting*) mampu mencegah penyalahgunaan berskala besar. Perangkat seperti curl atau skrip Python (menggunakan pustaka *requests*) juga dapat digunakan untuk mengotomatiskan panggilan-panggilan ini guna melakukan pengujian dalam skala luas. Selain itu, penggunaan fuzzer untuk memberikan input yang tidak sesuai format (*malformed input*) dapat memberikan temuan yang signifikan. Sebagai contoh, perangkat seperti boofuzz atau Peach Fuzzer dapat dikonfigurasi untuk menguji REST API dengan parameter acak guna melihat apakah muncul respons HTTP 500 *Internal Server Error* atau perilaku ganjil lainnya (yang mengindikasikan adanya celah injeksi). Penting untuk dicatat bahwa dalam lingkungan *backend* kendaraan, satu cacat logika saja berpotensi berdampak pada jutaan kendaraan. Dalam praktiknya, sering kali pada tahap inilah penguji memastikan apakah kelemahan pada sistem *backend* dapat berujung pada akses tidak sah terhadap fungsi kendaraan atau data pelanggan dalam skala luas.

Autentikasi, Otorisasi, dan Eskalasi Hak Akses

Tahap ini menunjukkan eksploitasi dengan membuktikan apakah identitas dengan hak akses rendah, token yang telah disusupi, atau akun dengan cakupan akses yang tidak tepat dapat memperoleh akses ke fungsi atau data di luar peran yang seharusnya. Pengujian autentikasi pada sistem *backend* kendaraan berfokus pada proses masuk (*login*) aplikasi seluler, portal operator armada, akun layanan dealer, serta integrasi layanan berbasis API. Alat proxy intersepsi seperti Burp Suite, OWASP ZAP, dan Fiddler memungkinkan penguji keamanan untuk memeriksa penerbitan token, cookie sesi, dan alur autentikasi multi-faktor. Jika aplikasi menggunakan JWT untuk autentikasi, penguji dapat mencoba memalsukan token (misalnya, jika token tidak ditandatangani dengan benar atau jika *backend* tidak memverifikasi tanda

tangan tersebut, sebuah kelemahan yang diketahui dalam beberapa implementasi JWT). Alat seperti jwt_tool dapat membantu dalam memeriksa kerentanan pada token.

Pengujian otorisasi dalam sistem otomotif sering kali berkisar pada kontrol akses berbasis VIN (*Vehicle Identification Number*). Penguji mencoba melakukan eskalasi hak akses horizontal dengan memodifikasi nilai VIN dalam panggilan API untuk menentukan apakah data kendaraan milik pelanggan lain dapat diakses. Skenario eskalasi vertikal dapat mencakup peningkatan akun konsumen menjadi akun tingkat dealer atau administrator armada dengan memanipulasi pengidentifikasi peran (*role identifiers*) atau klaim izin (*permission claims*).

Dalam jaringan dealer dan layanan, penguji mengevaluasi apakah titik akhir diagnostik (*diagnostic endpoints*), API untuk pembaruan perangkat lunak (software *flash*ing), atau basis data riwayat servis telah disegmentasi dengan tepat. Lingkungan armada diperiksa untuk memastikan bahwa satu operator armada tidak dapat mengakses kendaraan milik operator armada lainnya. Karena sistem *backend* kendaraan mengelola perintah yang krusial bagi keselamatan serta data lokasi yang sensitif, kelemahan otorisasi sekecil apa pun dapat menimbulkan konsekuensi operasional yang luas. Dari perspektif rantai serangan (*attack chain*), keberhasilan dalam memintas otorisasi atau melakukan eskalasi hak akses dapat mengubah pijakan awal yang terbatas menjadi kendali operasional yang luas atas layanan kendaraan, fungsi armada, atau kemampuan pengiriman perintah jarak jauh.

Pengujian Keamanan Rantai Pasok dan Integrasi

Tahap ini memperluas model eksploitasi hingga melampaui sistem *backend* OEM itu sendiri, dengan menguji apakah layanan pihak ketiga yang tepercaya atau dependensi perangkat lunak dapat menjadi jalur untuk masuk ke dalam ekosistem kendaraan yang lebih luas. Topik keamanan rantai pasok telah mendapatkan perhatian besar dalam beberapa tahun terakhir akibat berbagai insiden peretasan berskala besar, termasuk kompromi dalam ekosistem npm, di mana paket-paket tepercaya dan akun pengelola (*maintainer*) disalahgunakan untuk menyebarkan kode berbahaya kepada basis pengguna hilir yang luas.

Sistem *backend* otomotif rentan terhadap berbagai masalah rantai pasok yang juga dialami oleh industri lain, karena sistem ini bergantung pada banyak layanan pihak ketiga seperti penyedia data peta atau lalu lintas, penyedia layanan telematika, serta platform manajemen armada. Pengujian rantai pasok sangat penting untuk mengetahui apakah pustaka (*library*) yang memiliki kerentanan dapat berdampak pada sistem *backend* kendaraan. Perangkat seperti OWASP *Dependency-Check* dan Trivy dapat membantu mengidentifikasi komponen-komponen yang rentan di dalam *backend* kendaraan tersebut.

Penguji keamanan juga dapat menguji alur OAuth antara aplikasi seluler dan layanan *backend*, serta integrasi API dengan sistem manajemen dealer. Burp Suite dan Postman dapat digunakan untuk memanipulasi cakupan akses (*access scopes*) dan parameter *callback* guna menentukan apakah integrasi eksternal telah diimplementasikan dengan aman. Perlu dicatat bahwa keamanan *backend* tidak terbatas pada infrastruktur milik OEM saja, melainkan mencakup setiap integrasi tepercaya yang dapat memengaruhi kendaraan, pengguna, atau keputusan di sisi *backend*. Secara keseluruhan, teknik-teknik ini memberikan jalur terstruktur bagi penguji penetrasi (*penetration tester*) mulai dari mengidentifikasi aset *backend* yang

terekspos, memvalidasi kelemahan dalam aspek kepercayaan dan implementasi, hingga mendemonstrasikan potensi eksploitasi serta dampak operasionalnya. Dengan kata lain, penilaian *backend* tidak boleh dipandang sebagai sekumpulan pengujian yang terpisah-pisah, melainkan sebagai sebuah rangkaian kegiatan yang mencakup tahap pengintaian (*reconnaissance*), penilaian kerentanan, dan eksploitasi. Setelah meninjau berbagai teknik pengujian penetrasi yang umum digunakan, kita akan membahas beberapa studi kasus nyata di mana teknik-teknik tersebut diterapkan untuk berhasil menembus sistem *backend* kendaraan beserta sistem pendukungnya.

4.5 STUDI KASUS SERANGAN NYATA TERHADAP BACKEND KENDARAAN

Dalam beberapa tahun terakhir, telah terjadi sejumlah insiden besar yang melibatkan serangan terhadap *backend* kendaraan. Pada bagian ini, kami menyajikan tiga studi kasus tersebut. Setiap studi kasus menguraikan bagaimana satu atau lebih kerentanan ditemukan, bagaimana serangan dijalankan, serta dampaknya terhadap kendaraan. Contoh-contoh nyata ini memperkuat hipotesis bahwa kendaraan terhubung (*connected vehicles*) memiliki kerentanan *backend* yang serupa dengan perangkat IoT lainnya, sekaligus memberikan pelajaran bagi para insinyur mengenai potensi kegagalan sistem.

Studi Kasus 1: Mengeksploitasi Desain API Backend Kendaraan yang Tidak Aman

Kita mulai dengan serangan yang relatif sederhana ini, yang menunjukkan bagaimana desain *backend* yang tidak aman dapat berdampak langsung pada risiko fisik dan privasi di dunia nyata. Singkatnya, para peneliti menemukan bahwa API *backend* kendaraan Nissan hanya mengandalkan VIN (*Vehicle Identification Number*) yang dikirim oleh aplikasi NissanConnect EV untuk memberikan akses. Hal ini memungkinkan penyerang untuk mengirimkan permintaan web yang hanya berisi VIN kendaraan target, sehingga mereka dapat berinteraksi dari jarak jauh dengan kendaraan Nissan Leaf di seluruh dunia. Penyerang bisa mendapatkan akses tidak sah ke data telematika dan riwayat perjalanan, serta mengendalikan fungsi fisik kendaraan seperti sistem pengatur suhu (*climate control*) dari jarak jauh.

Lebih buruk lagi, penyerang dapat mengeksploitasi kelemahan ini dalam skala besar dengan melakukan enumerasi VIN secara sistematis untuk menyasar banyak kendaraan sekaligus. Meskipun kerentanan ini tidak berdampak pada fungsi-fungsi berkendara yang esensial seperti kemudi atau pengereman, hal ini membuat pemilik Nissan Leaf terpapar pada risiko privasi dan potensi serangan yang dapat menguras daya baterai. Alur serangan secara keseluruhan diilustrasikan pada Gambar 4.4.



Gambar 4.4 Alur eksploitasi API *backend* Nissan Leaf

Penemuan dan Pengintaian

Para peneliti memulai dengan memeriksa alur komunikasi aplikasi NissanConnect EV. Mereka mencegat lalu lintas data menggunakan Fiddler, yang memungkinkan mereka memeriksa permintaan yang dikirim dari aplikasi ke sistem *backend* kendaraan. Selama pemeriksaan tersebut, mereka menemukan bahwa operasi sensitif seperti mengambil status baterai dan riwayat perjalanan, atau mengatur sistem pengatur suhu (*climate control*) hanya memerlukan VIN (*Vehicle Identification Number*) sebagai input untuk tujuan autentikasi. Tidak ada kredensial login, cookie sesi, atau token OAuth yang diperlukan untuk membuktikan kepemilikan yang sah. Karena VIN sangat mudah diperoleh (dapat dilihat pada dasbor kendaraan, atau bahkan ditemukan dalam foto maupun daftar kendaraan secara daring), penyerang dapat menargetkan Nissan Leaf mana pun dengan memasukkan VIN tersebut ke dalam sistem *backend* NissanConnect EV.

Akses awal

Akses awal tidak mengharuskan penyerang untuk membobol sistem, melewati mekanisme autentikasi, atau mengeksploitasi kerentanan perangkat lunak dalam pengertian konvensional. API *backend* tersebut dapat diakses oleh publik dan berfungsi persis seperti yang dirancang hanya menggunakan VIN sebagai pengidentifikasi atau bentuk autentikasi, tanpa kontrol keamanan tambahan apa pun. Dengan demikian, penyerang mana pun yang menggunakan peramban web, perangkat baris perintah (*command-line tool*), atau skrip sederhana dapat mengirimkan permintaan HTTP GET yang diformat dengan benar langsung ke server *backend* Nissan. Karena API tersebut secara implisit memercayai VIN, penyerang yang memberikan VIN yang valid akan langsung mendapatkan akses ke data kendaraan terkait serta fungsi-fungsi kendalinya. Selain itu, serangan ini tidak terbatas pada satu wilayah geografis saja; para peneliti mengonfirmasi bahwa implementasi NissanConnect di berbagai wilayah

seperti Inggris, Norwegia, dan Kanada juga terdampak, meskipun terdapat sedikit perbedaan pada URL endpoint yang digunakan.

Eksploitasi

Setelah memastikan bahwa perintah dapat diterima hanya dengan menggunakan VIN (Nomor Identifikasi Kendaraan), para peneliti secara sistematis menguji berbagai titik akhir (*endpoint*) untuk melihat tindakan apa saja yang dapat dilakukan tanpa perlu menyertakan kredensial pengguna. Hasilnya menunjukkan fungsionalitas yang luas:

- **Permintaan status baterai**: Peneliti dapat memperoleh tingkat pengisian daya baterai saat ini, stempel waktu pembaruan terakhir, dan detail operasional lainnya.
- **Aktivasi kontrol iklim**: Permintaan untuk menyalakan AC, kipas angin, atau pemanas kursi diproses seketika. Dalam kondisi nyata, pengaktifan kontrol iklim secara berulang dapat menguras daya baterai.
- **Riwayat perjalanan**: API menyediakan log perjalanan sebelumnya, yang mencakup stempel waktu dan jarak tempuh, sehingga mengungkap pola penggunaan pemilik kendaraan.

Untuk memastikan bahwa kerentanan ini tidak hanya terbatas pada kendaraan mereka sendiri, para peneliti menggunakan fitur Intruder pada perangkat lunak Burp Suite untuk mengirimkan permintaan ke berbagai VIN pada titik akhir status baterai. Hasilnya, mereka menerima respons yang valid dari kendaraan yang bukan milik mereka. Perlu dicatat bahwa tidak satu pun dari permintaan tersebut menyertakan token sesi atau kredensial autentikasi; VIN saja sudah cukup untuk mengidentifikasi dan mengakses kendaraan apa pun yang terdaftar di sistem *backend*. Meskipun fungsi berkendara yang krusial seperti kemudi atau pengereman tidak terdampak, kemampuan-kemampuan ini tetap cukup signifikan untuk melanggar privasi dan berpotensi merugikan pemilik kendaraan dengan menguras daya baterai.

Persistensi

Serangan ini tidak memerlukan interaksi langsung dengan kendaraan, melainkan mengeksploitasi desain *backend* kendaraan yang tidak aman. Dengan demikian, selama API *backend* kendaraan terbuka, serangan ini memungkinkan akses yang berkelanjutan (*persistent access*). Karena semua permintaan baik yang sah maupun berbahaya disalurkan melalui infrastruktur cloud NissanConnect EV yang sama, sistem *backend* tidak memiliki mekanisme untuk membedakan antara pemilik kendaraan yang sebenarnya dan penyerang. Saat permintaan berbasis VIN masuk, *backend* akan langsung meneruskan perintah tersebut ke kendaraan target, yang kemudian memicu tindakan nyata seperti menyalakan pemanas atau mengambil data perjalanan.

Karena tidak ada sesi, token, atau kredensial yang digunakan, sistem *backend* tidak memiliki cara untuk mengakhiri atau mencabut akses tersebut. Selama API tetap tidak aman, penyerang dapat terus mengirimkan permintaan tanpa batas menggunakan VIN yang sama maupun VIN yang berbeda untuk mempertahankan kendali. Sebagai contoh, penyerang dapat dengan mudah membuat skrip untuk memeriksa status baterai secara berkala atau berulang kali mengaktifkan fungsi kontrol iklim.

Dampak

Dampak dari kerentanan ini mencakup aspek kegunaan fisik kendaraan maupun privasi pribadi. Para peneliti menjelaskan bahwa dengan mengaktifkan sistem pengatur suhu (*climate control*) secara berulang saat kendaraan diparkir, penyerang dapat menguras baterai secara signifikan, sehingga kendaraan tidak dapat dioperasikan dan pemiliknya berisiko telantar. Meskipun tidak secara langsung mengancam nyawa, hal ini merupakan gangguan yang nyata dan praktis. Yang lebih kritis, para peneliti menyoroti bagaimana terungkapnya riwayat perjalanan memungkinkan pembuatan profil pemilik kendaraan mengungkap rutinitas harian, pola perjalanan, serta perkiraan waktu saat pemilik tidak berada di rumah yang berujung pada pelanggaran privasi pemilik. Pada akhirnya, serangan ini menunjukkan bagaimana API *backend* yang tidak aman dapat mengubah fitur-fitur pada kendaraan terhubung (*connected vehicle*) menjadi sarana pengawasan dan gangguan berskala besar, bahkan tanpa harus meretas sistem keselamatan inti kendaraan.

Studi Kasus 2: Pengambilalihan Kendaraan dari Jarak Jauh Melalui Kerentanan Backend

Pada September 2024, para peneliti keamanan mengungkapkan kerentanan kritis dalam infrastruktur portal dealer Kia yang memungkinkan pengambilalihan kendaraan Kia (yang dilengkapi perangkat keras terkait) dari jarak jauh, hanya dengan menggunakan nomor pelat kendaraan. Masalah ini telah diidentifikasi dan dilaporkan secara privat kepada Kia sebelumnya pada tahun 2024. Dengan menggabungkan beberapa kelemahan termasuk kesalahan konfigurasi API, hak akses dealer yang terlalu luas, serta penggunaan layanan pihak ketiga untuk pencarian VIN para peneliti menunjukkan bagaimana celah keamanan aplikasi web yang umum dapat meningkat menjadi risiko keamanan fisik nyata dalam skala masif ketika berada dalam ekosistem kendaraan terhubung.

Secara sederhana, penyerang dapat memperoleh VIN kendaraan melalui nomor pelat, mendaftarkan akun dealer palsu, mendapatkan token sesi dengan hak akses khusus, mencabut akses pemilik sah, menetapkan diri mereka sebagai pemilik utama yang baru, dan memperoleh akses ke berbagai fungsi kendaraan dari jarak jauh. Setelah ditetapkan sebagai pemilik baru, penyerang dapat membuka kunci pintu, menyalakan atau mematikan mesin, serta melacak lokasi kendaraan. Serangan tersebut tidak hanya mengungkap kemampuan untuk mengendalikan kendaraan, tetapi juga sejumlah besar Informasi Identitas Pribadi (PII), termasuk nama, nomor telepon, alamat surel, dan alamat rumah. Urutan langkah-langkah serangan digambarkan dalam Gambar 4.5.



Gambar 4.5 Alur eksploitasi API *backend* dealer Kia

Penemuan dan Pengintaian

Para peneliti awalnya memulai dengan memeriksa situs web dan aplikasi seluler Kia yang ditujukan bagi pemilik kendaraan. Selama penyelidikan, mereka mengamati bagaimana kendaraan yang baru dibeli diaktifkan dan ditautkan ke akun pelanggan. Hal ini kemudian mengalihkan fokus mereka ke infrastruktur dealer Kia, khususnya domain *kiaconnect.kdealer.com*, yang digunakan secara internal oleh pihak dealer untuk mendaftarkan kendaraan kepada pelanggan. Selanjutnya, para peneliti memeriksa JavaScript sisi klien (*client-side*) pada portal dealer dan mencegat lalu lintas API menggunakan perangkat seperti BurpSuite. Langkah ini memungkinkan mereka untuk mengidentifikasi endpoint internal serta mempelajari perilaku token autentikasi. Melalui proses tersebut, mereka menemukan referensi ke endpoint API internal yang "khusus untuk karyawan", seperti:

- /dec/dlr/dvl
- /dec/dlr/rvp
- /ownr/dicve
- /prof/authUser

Endpoint-endpoint ini menangani pencarian kendaraan oleh dealer dan pengambilan data pemilik (dvl), pencabutan izin akses kendaraan (rvp), penetapan kepemilikan kendaraan (dicve), serta autentikasi (authUser). Para peneliti menemukan bahwa sistem *backend* secara otomatis menerima permintaan apa pun yang disertai dengan token dealer yang sah. Dengan demikian, serangan ini dimungkinkan karena adanya kelemahan mendasar dalam model kepercayaan (*trust model*), bukan sekadar eksploitasi terhadap kerentanan yang berdiri sendiri.

Akses Awal

Terobosan untuk mendapatkan akses awal terjadi ketika para peneliti menemukan bahwa mereka dapat mendaftarkan akun dealer secara langsung melalui portal dealer. Portal tersebut menerima permintaan pendaftaran yang mengikuti struktur HTTP yang sama dengan yang digunakan untuk pembuatan akun pemilik kendaraan pada umumnya. Yang krusial, tidak ada proses orientasi (*onboarding*) atau verifikasi untuk memastikan bahwa pendaftar adalah dealer resmi Kia. Akibatnya, para peneliti dapat membuat akun dealer palsu dan memperoleh akses ke fungsi-fungsi tingkat dealer tanpa menjadi dealer yang sah.

Setelah pendaftaran, para peneliti melakukan autentikasi melalui endpoint /prof/authUser dan menerima token sesi yang sah. Token ini dianggap oleh sistem *backend* sebagai kredensial dealer yang sah, sehingga memberikan akses ke fungsi API tingkat dealer yang memiliki hak istimewa. Dengan kata lain, token sesi istimewa ini berfungsi layaknya kunci utama (m*aster key*) untuk mengakses sistem *backend* kendaraan Kia. Tidak ada autentikasi tambahan seperti verifikasi di sisi server untuk memastikan apakah permintaan tersebut benar-benar berasal dari jaringan dealer resmi yang dilakukan. Singkatnya, selama token tersebut valid, sistem *backend* akan memercayai permintaan yang masuk.

Eksploitasi

Dengan menggunakan token dealer yang sah dan nomor identifikasi kendaraan (VIN) dari kendaraan target, para peneliti dapat melanjutkan serangan tersebut. Jika penyerang tidak mengetahui VIN tetapi dapat melihat pelat nomor kendaraan target, mereka dapat menggunakan layanan pihak ketiga seperti *PlateToVin.com* untuk mengubah pelat nomor menjadi VIN. Melalui layanan ini, para peneliti menunjukkan bahwa mereka dapat memperoleh VIN yang sesuai dari sebuah nomor pelat kendaraan. Berbekal VIN kendaraan target, para peneliti kemudian dapat melanjutkan ke langkah-langkah berikutnya. Pertama, mereka melakukan kueri ke endpoint /dec/dlr/dvl untuk mendapatkan informasi sensitif mengenai kendaraan target beserta pemilik terdaftarnya dengan menggunakan VIN. Informasi ini mencakup nama pemilik kendaraan, nomor telepon, alamat surel, dan terkadang alamat fisik. Sistem tersebut menunjukkan perilaku *Insecure Direct Object Reference* (IDOR) yang klasik. Artinya, kepemilikan VIN yang sah sudah cukup untuk mengakses data pribadi pemilik sebuah bentuk IDOR dengan akses baca.

Selanjutnya, para peneliti menggunakan endpoint /dec/dlr/rvp untuk menurunkan atau menghapus izin pemilik sah dari akun kendaraan tersebut. Sistem tidak meminta konfirmasi maupun verifikasi tambahan dari pemilik asli, serta tidak menghasilkan pemberitahuan apa pun mengenai perubahan tersebut, sehingga modifikasi kepemilikan dapat terjadi secara diam-diam. Terakhir, melalui endpoint /ownr/dicve, para peneliti dapat menetapkan alamat email mereka sendiri sebagai pemilik utama kendaraan, yang menjadikan mereka sebagai pemilik sah yang baru menurut sistem. Akan tetapi, karena pemilik asli tidak akan bisa lagi mengakses kendaraannya, mereka akan segera menyadari adanya serangan tersebut. Untuk menghindari deteksi, para peneliti menunjukkan bahwa mereka juga dapat menambahkan diri mereka sebagai pengguna sekunder yang "tidak terlihat", di mana pemilik asli tetap memiliki akses ke kendaraan tersebut. Perlu dicatat bahwa dengan memiliki VIN yang

valid, sistem juga mengizinkan modifikasi kepemilikan kendaraan tanpa pemeriksaan otorisasi yang semestinya sebuah kerentanan yang disebut sebagai IDO™ dengan kemampuan tulis (*write capability*). Berikut adalah contoh ilustratif permintaan API untuk mengalihkan kepemilikan kendaraan. Perlu diperhatikan bahwa jalur endpoint dan nama parameter yang sebenarnya digunakan oleh para peneliti berbeda dari contoh ini:

```
POST /api/vehicle/ownership/transfer HTTP/1.1
Host: ://kia.com
Authorization: Bearer [DEALER_TOKEN]

{
"vin": "KNDPMCAL3E7137076",
"new_owner_email": "attacker@example.com"
}
```

Setelah kepemilikan kendaraan dialihkan kepada para peneliti keamanan, mereka mendapatkan akses ke fitur kendali jarak jauh Kia Connect. Mereka dapat melakukan tindakan-tindakan berikut:

- Mengunci atau membuka kunci pintu
- Menyalakan atau mematikan mesin
- Membunyikan klakson dan menyalakan lampu depan
- Mengakses data lokasi GPS
- Berpotensi mengakses fungsi kamera pada model-model tertentu yang mendukungnya

Persistensi

Setelah menambahkan diri mereka sebagai pemilik utama atau sekunder, para peneliti berhasil mendapatkan akses persisten ke fungsi-fungsi kendaraan, karena mereka kini dianggap sebagai pemilik "sah" atas kendaraan target menurut sistem *backend* kendaraan tersebut. Seperti yang telah disebutkan, sistem tidak menghasilkan peringatan khusus atau notifikasi push kepada pemilik sah saat terjadi perubahan kepemilikan. Akibatnya, para peneliti mencatat bahwa penyerang dapat memantau lokasi kendaraan secara diam-diam atau mengeluarkan perintah jarak jauh dari waktu ke waktu tanpa sepengetahuan korban. Karena token dealer memberikan akses ke semua endpoint API dealer di *backend*, penyerang yang memegang token sah dapat berinteraksi dengan kendaraan mana pun yang terdaftar dalam sistem tanpa batasan.

Poin lain yang perlu disoroti adalah bahwa desain keamanan *backend* yang buruk turut berkontribusi terhadap persistensi ini. Akun dealer diperlakukan sebagai pihak yang secara inheren tepercaya dalam ekosistem tersebut, di mana token dealer berfungsi layaknya kunci utama (*master key*) yang memberikan akses ke layanan *backend* dengan hak istimewa tinggi. Selain itu, tidak ada pemeriksaan atau validasi tambahan mengenai kepemilikan kendaraan yang sah untuk permintaan berisiko tinggi, seperti pencabutan atau penetapan kepemilikan kendaraan.

Dampak

Dampak dari kerentanan ini sangat serius dan memengaruhi kendaraan Kia yang dilengkapi perangkat keras tertentu, yang diproduksi setelah tahun 2013 mencakup model tahun 2014 hingga 2025 dari berbagai jenis model. Dari perspektif keamanan fisik, para peneliti menunjukkan bahwa mereka dapat mengendalikan fungsi-fungsi krusial kendaraan dari jarak jauh, termasuk membuka kunci pintu dan menyalakan mesin. Hal ini menimbulkan risiko nyata bagi pemilik kendaraan, seperti pencurian kendaraan, penguntitan, atau potensi pelecehan maupun bentuk bahaya fisik lainnya.

Dari perspektif privasi, para peneliti menekankan bahwa penyerang dapat mengumpulkan sejumlah besar informasi identitas pribadi (*Personally Identifiable Information* atau PII), termasuk nama pemilik kendaraan, nomor telepon, alamat surel, alamat rumah, dan data pelacakan GPS. Beberapa model bahkan mengekspos antarmuka yang berpotensi memberikan akses ke fungsi kamera. Insiden pada portal dealer Kia ini menggambarkan bagaimana validasi identitas yang lemah, token dengan hak akses berlebih, kerentanan IDOR (*Insecure Direct Object Reference*), serta hubungan kepercayaan yang berlebihan di *backend* dapat menyebabkan kelemahan sederhana pada aplikasi web berkembang menjadi serangan dengan dampak berskala luas.

Studi Kasus 3: Pengendalian Kendaraan Jarak Jauh Melalui Kompromi Sistem Backend

Studi kasus ini mendokumentasikan upaya studi keamanan selama satu tahun yang menyasar ekosistem kendaraan terhubung (*connected vehicle*) milik Mercedes-Benz, yang berujung pada kemampuan untuk mengendalikan fungsi-fungsi kendaraan sepenuhnya dari jarak jauh. Berbeda dengan dua studi kasus sebelumnya yang berfokus pada API *backend*, serangan ini memerlukan pemeriksaan menyeluruh dari ujung ke ujung (*end-to-end*), mulai dari perangkat elektronik kendaraan hingga layanan *backend* internal. Para peneliti keamanan mengungkapkan temuan ini dalam ajang Black Hat USA 2020, yang mencakup kerentanan pada komponen perangkat keras di dalam kendaraan maupun layanan *backend* yang di-hosting di cloud.

Studi dimulai dengan melakukan rekayasa balik (*reverse engineering*) perangkat keras pada unit telematika (TCU) "Hermes". Peneliti mengekstrak *firmware*, menemukan kredensial yang tertanam di dalamnya, serta memperoleh akses ke jaringan operator seluler privat milik kendaraan tersebut. Selanjutnya, dengan merangkaikan sekitar 19 kerentanan berbeda termasuk materi kriptografi yang tertanam secara permanen (*hardcoded*), asumsi kepercayaan (*trust assumptions*) yang lemah pada sistem *backend*, serta celah *Server-Side Request Forgery* (SSRF) pada plugin integrasi media sosial mereka berhasil menyamar sebagai perangkat internal tepercaya dan menjalankan perintah kendaraan dari jarak jauh. Pada akhirnya, mereka menunjukkan kemampuan untuk membuka kunci pintu, mengendalikan jendela dan lampu, serta menyalakan mesin kendaraan dari jarak jauh tanpa memerlukan akses fisik, melalui interaksi dengan TCU Hermes tersebut. Langkah-langkah alur serangan untuk studi kasus ini ditunjukkan pada Gambar 4.6.



Gambar 4.6 Kendali kendaraan jarak jauh: alur eksploitasi *backend*

Penemuan dan Pengintaian

Studi dimulai dengan memetakan permukaan serangan (*attack surface*) pada kendaraan. Tiga komponen utama berhasil diidentifikasi:

- Head Unit NTG 5.5 (Windows CE 7 Automotive ARM)
- Antarmuka OBD-2
- Hermes TCU (Linux ARM)

Meskipun *head unit* (sistem infotainment) dan antarmuka OBD-2 turut diperiksa, Hermes TCU segera menjadi target yang paling krusial karena memiliki konektivitas 4G terintegrasi, perangkat lunak berbasis Linux, serta bertanggung jawab menangani layanan jarak jauh seperti membuka kunci pintu dan menyalakan mesin. Untuk melakukan pemeriksaan dengan aman, para peneliti membangun perangkat pengujian khusus yang mereplikasi lingkungan kendaraan. Hal ini mencakup penghubungan TCU dengan head unit serta melakukan *spoofing* (pemalsuan) pesan heartbeat pada bus CAN guna mencegah terpicunya mekanisme penguncian anti-pencurian. Pengaturan laboratorium yang terkendali ini memungkinkan pemeriksaan perangkat keras dan eksperimen *firmware* tanpa risiko merusak unit secara permanen (*bricking*).

Selanjutnya, para peneliti keamanan memulai pemeriksaan terhadap Hermes TCU dengan mengeksplorasi antarmuka debugging standar pada sistem tertanam (*embedded system*). Melalui antarmuka *Universal Asynchronous Receiver-Transmitter* (UART), mereka mengakses log sistem yang mengungkap nama pengguna (*username*) dan URL *backend*. Dengan menggunakan antarmuka *Universal Serial Bus* (USB), mereka menjalankan perintah `AT^SETMODE` untuk mengalihkan perangkat ke mode debug, yang kemudian membuka akses ke port komunikasi tambahan. Melalui perubahan mode USB tersebut, mereka dapat mengirimkan perintah AT lebih lanjut ke antarmuka aplikasi, yang memungkinkan mereka

mengekstrak konfigurasi aplikasi yang bersifat sensitif. Konfigurasi aplikasi ini mengungkap detail penting terkait operator seluler, seperti nama APN, nama pengguna, dan kata sandi yang digunakan untuk terhubung ke jaringan privat kendaraan.

Istilah:
APN adalah pengidentifikasi jaringan unik yang digunakan oleh perangkat seluler untuk terhubung ke jaringan seluler privat, alih-alih ke internet publik. Dengan menetapkan APN tertentu, unit telematika pada kendaraan akan mengarahkan data melalui jalur khusus menuju infrastruktur *backend*. Pengaturan ini secara efektif mengisolasi dan melindungi lalu lintas sensitif dari paparan internet secara luas, sehingga memperkecil area serangan (*attack surface*) dan memungkinkan operator untuk menerapkan kebijakan keamanan yang terperinci seperti pemfilteran IP atau aturan firewall pada tingkat operator seluler maupun *gateway* privat.

Lebih lanjut, para peneliti menggunakan perangkat seperti OpenOCD dengan adaptor FT2232 untuk mengidentifikasi pin JTAG dan mengoperasikan antarmuka debug. Akan tetapi, meskipun mereka berhasil menggunakan antarmuka JTAG untuk membaca atau menulis memori prosesor modul 4G, mereka mendapati bahwa antarmuka tersebut terlalu tidak stabil untuk melakukan dump *firmware* secara menyeluruh. Hal ini disebabkan oleh ketiadaan driver NAND khusus untuk antarmuka JTAG serta adanya gangguan dari watchdog sistem.

Istilah:

- Antarmuka *Universal Asynchronous Receiver-Transmitter* (UART) adalah antarmuka komunikasi serial yang umum digunakan dalam sistem tertanam (*embedded systems*) untuk mengakses keluaran konsol, mengirim perintah, dan melakukan penelusuran kesalahan (*debugging*) melalui koneksi perangkat keras yang sederhana.
- Antarmuka *Universal Serial Bus* (USB) adalah antarmuka komunikasi perangkat keras yang digunakan untuk menghubungkan perangkat guna keperluan transfer data dan penyediaan daya; dalam sistem tertanam, antarmuka ini sering digunakan untuk akses *firmware*, pertukaran konfigurasi, dan penelusuran kesalahan.
- Perintah mode "AT set" adalah perintah perangkat yang mengubah status operasional perangkat keras untuk mengaktifkan saluran komunikasi penelusuran kesalahan tambahan yang dapat diakses melalui USB.
- Antarmuka *Joint Test Action Group* (JTAG) adalah antarmuka penelusuran kesalahan perangkat keras yang menyediakan akses tingkat rendah ke prosesor dan memori perangkat untuk keperluan pengujian, pemrograman, dan pemeriksaan *firmware*.

Oleh karena itu, para peneliti melakukan pembongkaran fisik terhadap Hermes TCU. Dengan menggunakan stasiun perbaikan *Ball Grid Array* (BGA), mereka melepas cip memori *flash* NAND dari papan sirkuit. Para peneliti kemudian membuat adaptor papan sirkuit cetak (PCB) khusus yang dapat dihubungkan langsung ke memori mentah tersebut, yang memungkinkan mereka untuk menyalin (*dump*) *firmware* menggunakan perangkat pembaca NAND, Proman. Dengan memeriksa hasil salinan tersebut, mereka kemudian merekonstruksi tabel partisi dan

sistem berkas, sehingga memperoleh akses penuh ke berkas biner aplikasi OEM, berkas konfigurasi, sertifikat terenkripsi, dan skrip.

Namun, saat para peneliti melakukan rekayasa balik (*reverse-engineering*) terhadap berkas biner aplikasi, mereka menemui kendala besar. Berkas-berkas eksekutabel tersebut tidak dapat dijalankan di dalam emulator, dan terdapat referensi di dalam kode yang hilang atau keliru. Mereka menyadari bahwa *dump* (salinan data) *firmware* tersebut mengalami masalah *bit-flipping* (perubahan nilai bit), yang tampaknya disebabkan oleh suatu fenomena pada memori *flash* NAND masalah yang biasanya dikoreksi oleh *Error Correction Code* (ECC) di dalam perangkat keras pengontrol NAND. Selama tahap awal rekonstruksi sistem berkas, para peneliti melewatkan proses ECC, sehingga galat-galat tersebut tetap ada dalam berkas yang diekstraksi. Lebih jauh lagi, jika para peneliti mencoba menulis kembali data yang telah dimodifikasi ke dalam NAND misalnya, mengubah baris perintah (*command line*) untuk mendapatkan akses *root shell* data tersebut akan menjadi rusak. Hal ini terjadi karena algoritma ECC pada perangkat keras NAND justru akan mengubah data yang "benar" menjadi "salah" akibat adanya *bit-flipping* tersebut. Oleh karena itu, para peneliti harus terlebih dahulu memulihkan algoritma ECC agar dapat mengekstraksi dump *firmware* yang akurat tanpa adanya bit-flipping.

Selanjutnya, mereka memodifikasi parameter boot dan konfigurasi baris perintah untuk mendapatkan akses root shell interaktif pada perangkat tersebut. Karena mekanisme *secure boot* hanya memeriksa prosedur booting, para peneliti dapat dengan leluasa mengubah berkas-berkas sistem untuk memasang backdoor berupa root shell ini. Mereka kemudian menuliskan kembali data mentah yang telah dimodifikasi tersebut ke dalam memori *flash* NAND. Melalui root shell ini, para peneliti menemukan sebuah program mode teknisi (*engineer mode*) yang memiliki akses ke bus CAN, yang memungkinkan mereka mengirimkan sinyal seperti perintah untuk mengunci atau membuka pintu kendaraan.

Akses Awal

Para peneliti kemudian memeriksa TCU (unit telematika) Hermes dan menemukan bahwa perangkat tersebut dilengkapi dengan *embedded Subscriber Identity Module* (eSIM). Dengan mencabut eSIM tersebut secara fisik dan memindahkannya ke papan pengembangan (*development board*) atau router 4G, serta mengonfigurasinya menggunakan nama APN yang sebelumnya telah mereka peroleh melalui USB, mereka berhasil mereplikasi identitas seluler kendaraan tersebut ke dalam perangkat keras milik mereka sendiri. Untuk menghindari pemblokiran oleh jaringan, mereka memastikan bahwa *International Mobile Equipment Identity* (IMEI) yang digunakan sama dengan IMEI pada TCU asli.

Langkah ini memungkinkan para peneliti untuk terhubung langsung ke jaringan operator internal privat yang sama dengan yang digunakan oleh kendaraan resmi. Dari sudut pandang sistem *backend*, perangkat yang mereka susun di laboratorium kini tampak sama persis dengan kendaraan yang sesungguhnya. Namun, jaringan operator memiliki kebijakan keamanan di mana perangkat-perangkat diisolasi satu sama lain, sehingga pemindaian terhadap kendaraan lain yang terhubung tidak dimungkinkan. Sebagai gantinya, para peneliti menggunakan URL *backend* yang sebelumnya telah diekstraksi melalui UART untuk menelusuri

bagaimana cara terhubung ke *backend* kendaraan. Koneksi ke *backend* kendaraan memerlukan protokol HTTPS dengan autentikasi mTLS, sehingga dibutuhkan sertifikat klien.

Sertifikat terenkripsi diekstraksi dari dump *firmware* dan diperiksa lebih lanjut. Sertifikat-sertifikat tersebut disimpan dalam wadah *Public-Key Cryptography Standards 12* (PKCS 12) yang disebut sebagai berkas PFX, yang dilindungi oleh kata sandi. Kata sandi untuk mengakses berkas PFX tersebut disimpan di dalam berkas passwd, yang kemudian dienkripsi menggunakan algoritma AES-256-CBC. Para peneliti menemukan bahwa HERMES akan memuat berkas PFX dan berkas passwd saat proses booting, kemudian aplikasi tersebut akan memuat sertifikat wilayah yang sesuai berdasarkan area pasar. Lebih lanjut, para peneliti menemukan bahwa sebuah pustaka sistem dapat mendekripsi berkas passwd menggunakan algoritma AES-256-CBC serta kunci AES-256 dan nilai *Initialization Vector* (IV) yang tertanam di dalam kode (*hardcoded*). Oleh karena itu, para peneliti menggunakan OpenSSL beserta nilai-nilai yang tertanam tersebut untuk mendekripsi berkas passwd guna memperoleh kata sandi berkas PFX, yang kemudian memungkinkan mereka mengakses sertifikat klien. Dengan kredensial APN yang telah diekstraksi dan sertifikat klien yang telah didekripsi, mereka dapat melakukan autentikasi ke layanan *backend* sebagai perangkat tepercaya. Pada tahap ini, mereka berhasil menyamar sebagai unit telematika (TCU) Hermes yang sah di dalam infrastruktur *backend* kendaraan.

Eksploitasi

Setelah berhasil masuk ke dalam jaringan infrastruktur kendaraan pribadi, para peneliti mengalihkan fokus mereka pada kerentanan di sisi *backend*. Pertama, mereka mengidentifikasi sebuah plugin media sosial pada head unit sebagai vektor serangan potensial. Komponen ini menerima URL yang dimasukkan pengguna untuk mengambil gambar profil atau data. Akibat validasi URL yang tidak memadai, muncul kerentanan *Server-Side Request Forgery* (SSRF). Para peneliti dapat menyusun URL sedemikian rupa untuk memaksa server *backend* melakukan permintaan HTTP ke tujuan arbitrer, seperti berkas lokal di server tersebut atau server di dalam intranet.

Karena permintaan ini berasal dari infrastruktur *backend*, permintaan tersebut dapat mengakses layanan internal yang tidak terpapar langsung ke internet publik. Dengan kata lain, kerentanan SSRF secara efektif mengubah komponen *backend* yang rentan ini menjadi sebuah proksi, yang memungkinkan penyerang menjangkau sistem internal tepercaya. Sebagai contoh, para peneliti menunjukkan bagaimana mereka dapat membaca data tertentu dari file lokal di server Penyedia Layanan Telematika (*Telematics Service Provider*/TSP), misalnya file lokal `/etc/passwd`.

Lebih lanjut, dengan menggunakan SSRF untuk melakukan pergerakan lateral (*lateral movement*), para peneliti memindai dan berinteraksi dengan layanan *backend* kendaraan. Yang krusial, beberapa layanan internal ini tidak memiliki autentikasi tambahan; layanan tersebut secara implisit memercayai setiap permintaan yang berasal dari jaringan internal. Di antara layanan-layanan tersebut, para peneliti menemukan komponen *backend* yang kritis, yaitu endpoint kendali perintah kendaraan jarak jauh. *Endpoint* ini menyediakan fungsionalitas untuk perintah seperti mengunci/membuka kunci pintu, membuka/menutup atap,

menyalakan/mematikan lampu, serta menghidupkan/mematikan mesin (pada kendaraan yang mendukung). Memicu endpoint ini akan meneruskan permintaan ke TCU (*Telematics Control Unit*) terkait untuk menjalankan tindakan yang sebenarnya pada kendaraan (seperti mengunci/membuka kunci, membuka/menutup atap, menghidupkan/mematikan mesin, dll.).

Rangkaian serangan ini menunjukkan kegagalan dalam model kepercayaan *(trust model*) *backend*, di mana lokasi dalam jaringan internal dianggap sebagai bukti legitimasi yang memadai. Artinya, tidak ada autentikasi timbal balik (*mutual authentication*) atau validasi token yang diterapkan di antara layanan-layanan tersebut, sehingga memungkinkan para peneliti untuk memicu layanan internal setelah mereka berhasil mendapatkan akses ke infrastruktur *backend* internal. Singkatnya, eksploitasi ini memerlukan penggabungan langkah-langkah berikut:

- Rekayasa balik (*reverse engineering*) *firmware*
- Ekstraksi kunci AES-256-CBC yang tertanam secara permanen (*hardcoded*)
- Dekripsi sertifikat klien
- Penemuan kredensial APN
- Pemindahan eSIM dan replikasi IMEI
- Eksploitasi SSRF pada plugin media sosial
- Penyalahgunaan layanan internal yang tidak memerlukan autentikasi

Masing-masing masalah mungkin tampak terbatas dalam cakupannya, namun jika digabungkan menjadi satu rangkaian serangan yang utuh, hal tersebut memungkinkan eksekusi perintah jarak jauh secara penuh terhadap kendaraan.

Persistensi

Dalam kasus ini, persistensi dicapai dengan mempertahankan akses yang andal ke infrastruktur *backend* kendaraan. Setelah mengekstrak kredensial APN, mendekripsi sertifikat klien, dan memindahkan eSIM ke perangkat keras 4G milik mereka sendiri, para peneliti mampu menyamar sebagai TCU (*Telematics Control Unit*) Hermes yang sah di jaringan operator privat. Dari sudut pandang *backend* kendaraan, sistem yang digunakan oleh para peneliti tersebut tidak dapat dibedakan dari TCU yang asli. Selama mereka tetap memiliki kredensial APN yang valid, sertifikat klien untuk autentikasi mTLS, serta eSIM yang berfungsi dengan identitas IMEI yang sesuai, mereka dapat terus-menerus terhubung kembali ke *backend* kendaraan. Kerentanan SSRF (*Server-Side Request Forgery*) pada plugin media sosial kemudian berfungsi sebagai titik tumpu (*pivot point*). Dengan mengeksploitasi kerentanan ini, mereka dapat meneruskan permintaan ke layanan *backend* yang tidak memiliki autentikasi memadai dan memicu endpoint perintah jarak jauh.

Dengan demikian, persistensi bergantung pada dua kondisi: akses berkelanjutan ke jaringan privat sebagai perangkat tepercaya, dan keberadaan celah SSRF. Selama model kepercayaan (*trust model*) pada *backend* tidak diubah, mereka dapat berulang kali membangun kembali akses dan memicu perintah jarak jauh pada kendaraan. Persistensi ini bersifat arsitektural bukan berbasis perangkat karena mengandalkan kelemahan desain *backend*, bukan melalui pemasangan perangkat lunak berbahaya (*malware*) di dalam kendaraan.

Dampak

Eksploitasi yang berhasil terhadap kombinasi kerentanan tersebut memungkinkan kendali jarak jauh penuh atas fungsi-fungsi krusial kendaraan tanpa memerlukan akses fisik. Para peneliti mendemonstrasikan kemampuan seperti berikut:

- Mengunci dan membuka kunci pintu
- Mengoperasikan atap kendaraan dan lampu
- Menyalakan mesin dari jarak jauh

Lebih lanjut, para peneliti menyatakan bahwa serangan ini berpotensi berdampak pada sekitar dua juta kendaraan Mercedes-Benz yang terhubung di pasar Tiongkok. Para peneliti keamanan mengikuti prosedur pengungkapan yang bertanggung jawab (*responsible disclosure*), dan Mercedes-Benz segera memulai proses penanganan insiden. Hasilnya, paparan langsung dapat diatasi dalam waktu 48 jam melalui penghentian layanan dan kontrol di tingkat jaringan, diikuti dengan perbaikan arsitektural menyeluruh dalam beberapa minggu berikutnya. Setelah menelaah bagaimana desain *backend* yang tidak aman, model kepercayaan API yang terlalu longgar, serta kredensial yang tertanam secara permanen (*hardcoded*) dapat dieksploitasi untuk mengendalikan kendaraan dari jarak jauh dalam skala besar, bagian selanjutnya akan menguraikan praktik terbaik untuk mengamankan infrastruktur *backend* kendaraan terhadap vektor serangan ini maupun serangan serupa lainnya.

4.6 STRATEGI PENGAMANAN BACKEND KENDARAAN

Studi kasus yang kami sajikan di sini hanyalah sebagian kecil dari serangan nyata yang pernah dipublikasikan terhadap sistem *backend* kendaraan. Dengan meninjau berbagai kasus tersebut, kita dapat merumuskan beberapa praktik terbaik keamanan yang harus dipertimbangkan. Bagi penguji penetrasi (*penetration tester*), mencari ketiadaan kontrol-kontrol ini bisa menjadi metode yang efektif untuk mengidentifikasi kerentanan dan kelemahan dalam sistem tersebut.

Autentikasi yang Kuat dan Kontrol Akses

Pertama, sangat penting untuk memastikan bahwa semua titik akhir (*endpoint*) dan pesan telah diautentikasi. Kendaraan harus melakukan autentikasi ke cloud menggunakan kredensial unik (sebaiknya berupa sertifikat perangkat atau kunci kriptografis unik) serta menerapkan autentikasi dua arah (*mutual authentication*, contohnya mTLS). Jangan mengandalkan keamanan melalui kerahasiaan (*security through obscurity*). Sebagai contoh, API *backend* tidak boleh hanya mengandalkan pengidentifikasi sederhana seperti VIN atau alamat email sebagai satu-satunya bukti identitas; sebaliknya, selalu terapkan autentikasi pengguna/sesi serta autentikasi kendaraan (misalnya, menggunakan token yang terikat pada pengguna dan kendaraan tertentu). Selain itu, desain API harus memprioritaskan keamanan dengan menghindari eksposur operasi sensitif melalui titik akhir yang dapat diakses publik.

Penting juga untuk membatasi hak akses seminimal mungkin sesuai dengan prinsip hak akses paling rendah (*principle of least privilege*). Selain itu, *Role-Based Access Control* (RBAC) harus diterapkan untuk memastikan bahwa hanya pengguna yang berwenang yang dapat mengakses dan melakukan tindakan tertentu sesuai dengan peran mereka. Di samping itu,

diperlukan model kepercayaan yang lebih eksplisit, seperti penerapan *Zero-Trust Architecture (ZTA), di mana setiap permintaan divalidasi (misalnya, melalui pemeriksaan kepemilikan* sumber daya), alih-alih langsung memercayai semua permintaan yang berasal dari dalam infrastruktur *backend*.

Dalam praktiknya, hal ini berarti menggunakan protokol seperti OAuth 2.0 untuk autentikasi pengguna ke cloud (sehingga perintah dari aplikasi seluler membawa token akses yang sah) serta memvalidasi bahwa pengguna yang memegang token tersebut memang memiliki wewenang atas kendaraan yang dituju, guna mencegah serangan *Insecure Direct Object Reference* (IDOR). Untuk portal admin internal, praktik terbaik yang disarankan mencakup penggunaan autentikasi multi-faktor (*multi-factor authentication*) dan daftar putih alamat IP (IP *whitelisting*) untuk mengurangi risiko penyalahgunaan kredensial yang mungkin telah dikompromikan, serta membatasi akses keseluruhan ke portal tersebut.

Mengamankan Komunikasi

Seluruh komunikasi antara kendaraan dan sistem *backend* harus diamankan menggunakan protokol modern (seperti TLS 1.2 atau yang lebih baru). Produsen wajib menerapkan enkripsi dan autentikasi yang memadai untuk setiap komunikasi antara kendaraan dan *backend*. Hal ini mencakup perlindungan topik MQTT dengan TLS, serta enkripsi pada muatan data (*payload*) untuk informasi yang bersifat sensitif. Kendaraan harus memvalidasi sertifikat server *backend* (guna mencegah serangan *man-in-the-middle*), sementara pihak *backend* harus menolak data apa pun dari kendaraan yang tidak dienkripsi atau tidak ditandatangani secara kriptografis. Selain itu, data sensitif yang tersimpan di dalam sistem *backend* (seperti informasi pribadi atau riwayat lokasi) harus dienkripsi dan aksesnya dibatasi.

Praktik terbaik yang disarankan adalah menghindari penggunaan kriptografi yang bersifat eksklusif (*proprietary*) atau lemah; sebaliknya, gunakan standar dan pustaka yang telah teruji (contohnya, gunakan AES untuk enkripsi penyimpanan pada perangkat, serta TLS/DTLS untuk transmisi data). Manajemen kunci juga sama pentingnya. Sertifikat dan kunci kriptografis harus dirotasi serta diperbarui secara berkala. Selain itu, perlu ditetapkan proses kedaluwarsa dan pembaruan untuk sertifikat kendaraan, sehingga sertifikat yang telah disusupi dapat menjadi tidak berlaku. Terapkan pula *perfect forward secrecy* pada koneksi TLS untuk memastikan bahwa komunikasi di masa lalu tetap terlindungi, bahkan jika kunci jangka panjang berhasil disusupi oleh pihak lain.

Validasi Input dan Pengembangan API yang Aman

Layanan *backend* harus memvalidasi secara ketat semua input dan perintah berdasarkan format serta rentang yang diharapkan, terutama yang berasal dari kendaraan atau aplikasi seluler. Langkah ini membantu mencegah kerentanan web yang umum terjadi, seperti *SQL injection, command injection*, SSRF, atau *buffer overflow* pada layanan yang memproses data kendaraan. Selain itu, pastikan untuk melakukan sanitasi terhadap input seperti data yang digunakan dalam jalur file (*file path*) atau perintah sistem pada *server backend* serta terapkan pemeriksaan otorisasi yang tepat untuk setiap permintaan. Dengan kata lain, sistem *backend* tidak boleh berasumsi bahwa pesan yang diterima melalui saluran "terpercaya" dari kendaraan

atau aplikasi adalah pesan yang sah; selalu verifikasi izin pengirim serta isi pesan tersebut. Sebagai contoh, jika terdapat permintaan untuk "membuka kunci kendaraan X", sistem *backend* harus memastikan bahwa pihak yang mengajukan permintaan tersebut telah terautentikasi sebagai pemilik kendaraan X (serta mungkin menambahkan pemeriksaan konteks, seperti pembatasan frekuensi perintah atau pendeteksian anomali, misalnya permintaan untuk membuka kunci kendaraan dari lokasi atau waktu yang tidak lazim).

Selain itu, terapkan praktik pengembangan yang aman untuk semua layanan cloud dan perangkat lunak di dalam kendaraan yang memproses input eksternal, serta gunakan praktik penulisan kode yang aman dan metode pengujian keamanan yang telah teruji. Sebagai contoh, *fuzz testing* pada API dapat membantu menemukan kerentanan injeksi atau kesalahan dalam pemrosesan data (*parsing*) sebelum sistem diterapkan. Patuhi rekomendasi OWASP dan lakukan pemantauan kode (*code review*) dengan mempertimbangkan aspek keamanan. Penggunaan pemeriksaan ketat serta kerangka kerja (*framework*) yang mampu melakukan pembersihan input (*input sanitization*) secara otomatis jika memungkinkan merupakan langkah krusial. Selain itu, nonaktifkan semua antarmuka debug dan lindungi fungsi administratif dalam sistem produksi, mengingat banyak masalah muncul akibat keberadaan API administratif yang tertinggal dan tidak ditujukan untuk penggunaan publik.

Menghilangkan Kredensial Bawaan dan Melindungi Rahasia

Jangan pernah menggunakan kata sandi atau token bawaan yang bersifat universal untuk perangkat apa pun. Setiap kendaraan atau perangkat harus memiliki rahasia atau sertifikat unik yang disediakan secara aman, misalnya selama proses manufaktur melalui *Hardware Security Module* (HSM). Jika penggunaan kredensial bawaan tidak dapat dihindari pada tahap awal, pastikan untuk mewajibkan perubahan kredensial saat pertama kali digunakan. Lakukan audit secara berkala untuk memastikan tidak ada kredensial milik pengembang atau kontraktor yang tertanam di dalam kode maupun berkas konfigurasi. Terapkan manajemen kredensial yang ketat dengan menggunakan vault (penyimpanan rahasia) dan hindari penggunaan kunci dalam bentuk teks biasa (*plaintext*) di dalam kode sumber. Gunakan pemindaian otomatis (seperti perangkat lunak GitGuardian) untuk mendeteksi kredensial sebelum kode dirilis.

Selain itu, lakukan segmentasi tugas agar jika kredensial satu perangkat berhasil disusupi, kredensial tersebut tidak dapat digunakan untuk mengakses perangkat lain atau meningkatkan hak akses (*privilege escalation*) pada sistem *backend*. Dengan kata lain, setiap kendaraan hanya boleh memiliki akses ke data atau perintah yang ditujukan untuk dirinya sendiri. Dalam infrastruktur cloud, terapkan prinsip hak akses minimum (*principle of least privilege*) untuk akun layanan (*service account*) dan kunci API. Artinya, jika satu layanan *backend* disusupi, hal tersebut tidak boleh secara otomatis memberikan hak akses administratif penuh ke seluruh sistem.

Segmentasi dan Keamanan Jaringan

Sangat penting untuk merancang arsitektur *backend* dengan segmentasi antar komponen kritis. Sebagai contoh, antarmuka yang berkomunikasi dengan kendaraan (seperti IoT Hub atau broker MQTT) harus diisolasi dari logika bisnis dan basis data menggunakan

firewall atau pemisahan melalui *Virtual Private Cloud* (VPC). Dengan demikian, jika antarmuka eksternal pada *backend* disusupi, data internal tidak akan langsung terekspos. Di dalam kendaraan, modul telematika juga harus diisolasi dari jaringan CAN yang bersifat kritis bagi keselamatan (sistem cloud tidak boleh mengirimkan perintah secara langsung untuk mengendalikan fungsi vital, seperti sistem pengereman kendaraan). Saat ini, penyedia layanan cloud telah menyediakan fitur security group (aturan kontrol akses jaringan bawaan cloud) dan layanan *Web Application Firewall* (WAF); gunakan fitur-fitur tersebut untuk menyaring permintaan berbahaya, seperti upaya injeksi SQL (SQL *injection*) atau user agent yang diketahui berbahaya. Terakhir, terapkan pembatasan laju permintaan (*rate limiting*) untuk membatasi volume permintaan yang berlebihan guna mengurangi risiko penyalahgunaan dan serangan *Denial-of-Service* (DoS).

Pemantauan, Deteksi, dan Respons Insiden

Bentuklah *Vehicle Security Operations Center* (VSOC) atau integrasikan log *backend* kendaraan ke dalam sistem *Security Operations Center* (SOC) yang sudah ada. Setiap tindakan sensitif (seperti perintah buka kunci, pengiriman pembaruan *firmware*, dll.) harus menghasilkan log yang disertai konteks (siapa, apa, kapan, dan kendaraan mana yang terlibat) untuk kemudian dipantau guna mendeteksi aktivitas mencurigakan. Selain itu, disarankan untuk menerapkan deteksi anomali, baik di sisi kendaraan (melalui agen *Intrusion Detection System* (IDS) di dalam TCU, jika memungkinkan) maupun di sisi *backend* (melalui layanan pemantauan API berbasis kecerdasan buatan/AI). Langkah ini bertujuan untuk menangkap penyimpangan, misalnya sebuah akun yang mengeluarkan perintah ke banyak kendaraan, atau kendaraan yang mengirimkan data tidak sesuai format yang bisa menjadi indikasi adanya kompromi keamanan. Dengan kata lain, praktik terbaik mencakup pemantauan berkelanjutan terhadap penggunaan API untuk mendeteksi aktivitas yang tidak wajar.

Melindungi Data Sensitif dan Privasi

Sistem *backend* sering kali menyimpan data pribadi (profil pengguna, riwayat lokasi kendaraan, dll.) serta informasi rahasia (kunci enkripsi, kunci privat yang terkait dengan sertifikat, dll.). Data ini harus dilindungi baik saat disimpan maupun saat digunakan. Gunakan enkripsi basis data atau setidaknya enkripsi kolom-kolom yang berisi data sensitif. Kata sandi harus diproses menggunakan hashing dengan tambahan salt, sementara kunci privat sebaiknya disimpan dalam brankas kunci (*key vault*) yang aman dan didukung oleh *Hardware Security Module* (HSM), bukan di dalam basis data. Terapkan kontrol akses yang ketat pada basis data sehingga, jika terjadi upaya injeksi SQL (SQL *injection*), penyerang tidak dapat dengan mudah mengambil seluruh isi data.

Pantau kueri data dalam jumlah besar atau yang tidak lazim, lalu picu peringatan untuk aktivitas yang mencurigakan tersebut agar dapat ditindaklanjuti dengan tepat. Jika penyerang berhasil menembus sistem *backend*, enkripsi data dapat meminimalkan kerusakan (penyerang tidak dapat dengan mudah memanfaatkan data yang dicuri). Prinsip *privacy-by-design* (privasi sejak tahap perancangan) harus diterapkan: kumpulkan hanya data yang diperlukan, lakukan anonimisasi atau pseudonimisasi jika memungkinkan, serta patuhi regulasi yang relevan,

seperti *General Data Protection Regulation* (GDPR). Tujuannya adalah memastikan bahwa jika terjadi serangan, dampak terhadap data pribadi pengguna dapat diminimalkan.

Manajemen Rantai Pasok dan Pihak Ketiga

Lakukan penilaian keamanan terhadap layanan telematika atau layanan cloud pihak ketiga mana pun yang terintegrasi dengan sistem *backend* kendaraan. Produsen kendaraan (OEM) harus mewajibkan penerapan langkah-langkah keamanan yang kuat melalui kontrak. Sebagai contoh, jika menggunakan layanan dari mitra telematika pihak ketiga, mintalah agar dilakukan pengujian penetrasi (*penetration test*) secara berkala dan pengungkapan kerentanan. Pantau kerentanan pada komponen sumber terbuka (*open-source*) yang digunakan dalam sistem *backend* (sebagai contoh, kerentanan log4j yang berdampak pada jutaan server) dan pastikan terdapat proses untuk segera menambal kerentanan yang dapat dieksploitasi dalam sistem cloud, sama seperti yang dilakukan pada kendaraan.

Prinsip Pertahanan Berlapis (*Defense-In-Depth*)

Dengan mengikuti praktik-praktik terbaik yang disebutkan di atas, perusahaan otomotif dapat mengurangi secara signifikan area yang rentan terhadap serangan (*attack surface*) pada sistem cloud *backend*. Sebagai pendekatan umum, prinsip pertahanan berlapis (*defense-in-depth*) harus diterapkan saat merancang keamanan siber untuk sistem *backend* kendaraan. Dengan demikian, arsitektur *backend* kendaraan yang aman akan memiliki beberapa lapisan penghalang yang harus ditembus oleh penyerang: jika satu lapisan pelindung gagal (misalnya, akibat bug validasi input yang tidak terdeteksi), kontrol lain (seperti autentikasi atau pemantauan) tetap dapat meminimalkan kerusakan yang terjadi. Tujuannya adalah untuk mencegah jenis eksploitasi sepele seperti yang terlihat dalam studi kasus kami. Para insinyur harus memahami dan menerapkan prinsip-prinsip seperti "*jangan pernah memercayai data yang berasal dari klien (perangkat atau aplikasi) tanpa verifikasi*" dan "*asumsikan bahwa layanan apa pun yang terekspos akan ditemukan dan diserang oleh pihak penyerang*."

4.7 KEAMANAN BACKEND KENDARAAN TERHUBUNG

Dalam beberapa tahun terakhir, berbagai insiden pelanggaran keamanan kendaraan yang menonjol menunjukkan bahwa infrastruktur *backend* berbasis cloud beserta API terkait merupakan target yang menarik bagi penyerang yang ingin mendapatkan akses ke dalam kendaraan. Dalam bab ini, kita telah menelaah bagaimana kelemahan keamanan yang lazim ditemukan pada sistem IoT seperti autentikasi yang rusak, enkripsi yang lemah, dan API yang rentan juga muncul dalam arsitektur *backend* otomotif. Melalui pemantauan terhadap desain *backend* kendaraan terhubung (*connected vehicle*) yang umum digunakan di platform seperti Azure dan AWS, terlihat bahwa kendaraan pada dasarnya adalah perangkat IoT yang bergerak, yang mengandalkan layanan cloud untuk memperluas fungsionalitasnya. Kita juga telah membahas bagaimana cacat dalam desain dan implementasi dapat terjadi mulai dari memercayai pengidentifikasi yang diberikan pengguna (seperti VIN), membiarkan portal admin terbuka tanpa perlindungan, hingga mendistribusikan perangkat dengan kata sandi bawaan (*default*).

Studi kasus nyata mengonfirmasi bahwa kelemahan-kelemahan ini bukan sekadar teori. Dalam setiap kasus, penyerang (dalam konteks ini, para peneliti) mampu mengeksploitasi jenis kerentanan yang sama dengan yang sering ditemukan pada sistem TI pada umumnya, yang memungkinkan mereka untuk melacak lokasi, membuka kunci, bahkan melumpuhkan kendaraan dari jarak jauh. Bagi penguji penetrasi (*penetration tester*), memeriksa aplikasi seluler pendamping serta layanan web merupakan langkah wajib saat menguji kerentanan keamanan yang berkaitan dengan *backend*. Bagi insinyur keamanan defensif, kelemahan yang dipaparkan beserta praktik terbaik keamanan yang relevan memberikan wawasan berharga untuk membangun layanan *backend* yang lebih tangguh, yang pada akhirnya dapat mengurangi *attack surface* (permukaan serangan) pada kendaraan secara efektif. Dengan *backend* yang telah diperkuat, baik pengemudi di jalan maupun para insinyur di balik layar dapat memiliki keyakinan yang lebih besar terhadap keamanan kendaraan modern secara keseluruhan. Setelah membahas *backend* kendaraan, kini saatnya untuk mendalami arsitektur E/E (*Electrical*/*Electronic*) kendaraan dan menelaah sistem nirkabel pada kendaraan di bab berikutnya.

BAB 5
KEAMANAN DAN PENGUJIAN SISTEM NIRKABEL KENDARAAN

Tidak diragukan lagi bahwa target utama dalam serangan terhadap kendaraan adalah serangan yang dapat dilancarkan dari jarak jauh dan menimbulkan konsekuensi besar, termasuk dampak terhadap keselamatan, kerugian finansial, serta gangguan operasional. Faktor kunci yang memungkinkan terjadinya serangan jarak jauh adalah keberhasilan pihak luar dalam menyusupi satu atau lebih antarmuka jaringan nirkabel pada kendaraan. Dapat dipahami mengapa produsen kendaraan (OEM) sangat mengkhawatirkan kemungkinan serangan jarak jauh, mengingat jenis serangan ini memiliki potensi terbesar untuk menyebar ke banyak kendaraan sekaligus. Tren peningkatan layanan konektivitas pada kendaraan yang mencakup jaringan seluler, Wi-Fi, Bluetooth, *Near-Field Communication* (NFC), hingga saluran frekuensi radio (RF) konvensional semakin memperluas peluang terjadinya serangan semacam itu. Melindungi protokol nirkabel kendaraan merupakan elemen krusial dalam strategi pertahanan berlapis; oleh karena itu, bab ini berfokus pada pemahaman mengenai kelemahan dan kerentanan umum yang sering ditemukan pada protokol-protokol tersebut.

Kita akan memulainya dengan meninjau berbagai protokol konektivitas nirkabel yang umumnya didukung oleh kendaraan, kemudian mengeksplorasi kelemahan dan kerentanan yang mungkin ada pada protokol-protokol tersebut. Selanjutnya, kita akan membahas berbagai perangkat dan teknik yang dapat digunakan untuk menyusupi protokol-protokol ini, dengan tujuan untuk mendapatkan akses awal ke dalam sistem kendaraan. Dalam prosesnya, kita akan merujuk pada beberapa contoh serangan nyata serta memberikan praktik terbaik untuk menghapus antarmuka konektivitas yang tidak diperlukan, sekaligus memperkuat antarmuka yang memang wajib ada agar lebih tahan terhadap serangan siber. Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

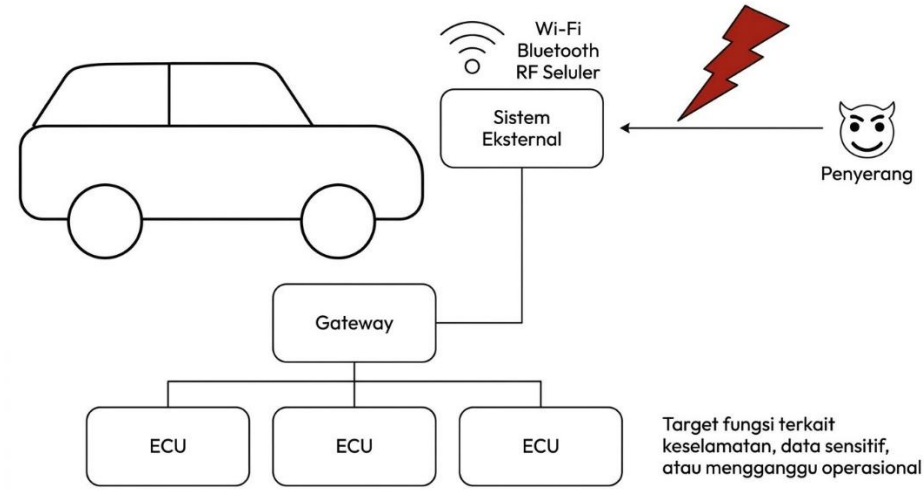
- Gambaran umum sistem kendaraan nirkabel
- Memahami protokol sistem kendaraan nirkabel
- Kelemahan umum pada sistem kendaraan nirkabel
- Teknik pengujian penetrasi untuk sistem kendaraan nirkabel
- Studi kasus serangan nyata terhadap sistem kendaraan nirkabel
- Praktik terbaik untuk mengamankan sistem kendaraan nirkabel

5.1 GAMBARAN UMUM SISTEM NIRKABEL PADA KENDARAAN

Dahulu, istilah "nirkabel" dan "kendaraan" jarang disebut dalam konteks yang sama. Namun, dalam beberapa tahun terakhir, jumlah sistem nirkabel yang diintegrasikan ke dalam kendaraan telah meningkat pesat. Mulai dari sistem pemantauan tekanan ban (*Tire Pressure Monitoring Systems*/TPMS) nirkabel, teknologi *Ultra-Wideband* (UWB), dan komunikasi *Vehicle-to-Everything* (V2X), hingga teknologi yang lebih umum seperti Wi-Fi, Bluetooth, dan *Global Navigation Satellite System* (GNSS), ragam antarmuka nirkabel pada kendaraan ternyata jauh lebih luas daripada yang dibayangkan kebanyakan orang. Faktanya, konektivitas nirkabel

telah menjadi landasan bagi kendaraan modern, yang memungkinkan berbagai fitur mulai dari akses kendaraan tanpa kunci (*remote keyless entry*) hingga fitur pemanggilan kendaraan (*vehicle summons*), di mana pemilik dapat mengambil mobilnya dari area parkir hanya dengan menggunakan ponsel mereka. Sebagaimana telah dibahas pada bab sebelumnya, kendaraan yang terhubung (*connected vehicles*) memanfaatkan berbagai antarmuka nirkabel untuk berkomunikasi dengan sistem internal maupun eksternal. Untuk mendukung beragam kegunaan, komunikasi nirkabel mencakup sistem yang berinteraksi dengan pihak luar seperti unit telematika atau sistem hiburan dalam kendaraan (*infotainment*) hingga sistem jaringan internal kendaraan, seperti *gateway* dan *Electronic Control Unit* (ECU).

Meskipun antarmuka nirkabel ini menawarkan kemudahan dan fungsionalitas, teknologi tersebut juga menciptakan banyak titik masuk bagi penyerang, sehingga memperluas area serangan (*attack surface*). Serangan nirkabel menyasar berbagai antarmuka yang digunakan oleh kendaraan, termasuk Wi-Fi, Bluetooth, jaringan seluler, dan komunikasi berbasis frekuensi radio (RF) lainnya. Dalam banyak kasus, serangan terhadap antarmuka ini berfungsi sebagai taktik akses awal yang memungkinkan pihak penyerang untuk menanamkan pijakan di dalam salah satu komponen elektronik kendaraan, sebelum melancarkan serangan yang lebih canggih untuk mencapai tujuan mereka, sebagaimana diilustrasikan dalam Gambar 5.1:



Gambar 5.1 Contoh alur serangan: mulai dari akses nirkabel hingga penargetan ECU di dalam kendaraan

Serangan nirkabel terhadap kendaraan yang terhubung (*connected vehicles*) dapat menimbulkan konsekuensi yang luas, baik dari segi privasi data, kerugian finansial, operasional, maupun keselamatan kendaraan. Kompromi pada sistem komunikasi nirkabel kendaraan dapat mengekspos informasi sensitif seperti data lokasi, kredensial pengguna, atau data diagnostik kendaraan yang bersifat rahasia. Selain itu, serangan yang menargetkan sistem seperti *keyless entry* (akses tanpa kunci fisik) dapat memungkinkan pelaku mencuri kendaraan. Lebih buruk lagi, serangan nirkabel yang berhasil dapat memberi penyerang kendali jarak jauh

atas sistem-sistem krusial seperti rem, kemudi, atau *powertrain* (sistem penggerak) sehingga sistem tersebut tidak berfungsi sebagaimana mestinya atau bahkan terganggu, yang dapat memicu situasi yang mengancam jiwa.

Studi keamanan terkini menunjukkan bahwa *attack surface* (permukaan serangan) nirkabel merupakan target utama, baik untuk pencurian kendaraan maupun sebagai titik awal untuk menyusup ke dalam jaringan internal kendaraan dalam rangkaian serangan yang kompleks (*multi-chain attacks*). Pada bagian studi kasus di bab ini, kita akan menelaah lebih dalam beberapa serangan nyata tersebut, namun sebelumnya mari kita pantau berbagai attack surface nirkabel yang relevan dengan kendaraan modern yang terhubung.

5.2 ANTARMUKA NIRKABEL DAN ANCAMAN KEAMANAN PADA KENDARAAN MODERN

Pada bagian ini, kita akan membahas antarmuka dan protokol nirkabel yang umum digunakan dalam sistem kendaraan modern. Meskipun protokol-protokol ini memungkinkan berbagai fitur bermanfaat yang meningkatkan pengalaman pengguna, protokol tersebut juga memperluas *attack surface* dan menjadi titik masuk yang potensial bagi penyerang jarak jauh. Memahami kegunaan setiap antarmuka, posisinya dalam arsitektur kendaraan, serta layanan tingkat tinggi apa yang biasanya berjalan di atasnya, merupakan prasyarat untuk memahami bagaimana antarmuka tersebut dapat menjadi rentan terhadap serangan. Gambaran umum mengenai antarmuka nirkabel yang lazim digunakan dalam sistem kendaraan disajikan dalam Tabel 5.1, diikuti dengan pembahasan singkat mengenai setiap kelompok protokol:

Tabel 5.1 Gambaran Umum Antarmuka Nirkabel

Antarmuka	**Jangkauan akses**	**Frekuensi**	**Teknologi**	**Aplikasi**	**Relevansi keamanan**
Wi-Fi	Hingga 100 m	2,4 GHz, 5 GHz, 6 GHz	IEEE 802.11a/b/g/n /ac/ax	pembaruan, Infotainment	Antarmuka IP yang dapat diakses dari luar; memperluas permukaan serangan; memerlukan autentikasi yang kuat, enkripsi, dan segmentasi jaringan.
Bluetooth	~10 m	2,4 GHz	Bluetooth Klasik, BLE	Panggilan bebas genggam, streaming audio, sensor, dukungan kunci digital	Rentan terhadap penyalahgunaan pemasangan (*pairing*), akses GATT yang tidak sah, serta serangan relai (*relay attacks*); keamanan bergantung pada pengikatan (*bonding*)

					dan pengendalian akses yang tepat.
Seluler	Hingga beberapa kilometer (sel makro)	Berbagai pita frekuensi (700 MHz–3,5 GHz, bergantung pada wilayah)	GSM, UMTS, LTE, 5G NR	Telematika, panggilan darurat, pembaruan OTA	Antarmuka *backend* utama; mengandalkan TLS, mTLS, dan otorisasi berbasis token; kesalahan konfigurasi dapat memicu serangan jarak jauh berskala besar.
™F	Bervariasi (10 m–30 m)	Berbagai pita frekuensi (Sub-GHz dan pita ISM (contoh: 315/433/868/915 MHz))	Protokol RF propieter pada pita ISM	Gantungan kunci (*key fob*), TPMS	Secara historis rentan terhadap serangan replay dan relay; kriptografi yang terbatas serta desain yang bersifat eksklusif menimbulkan risiko pencurian yang tinggi.
UWB	Bervariasi (10 m–30 m)	Rentang 3,1–10,6 GHz	IEEE 802.15.4z	Kunci digital (*Digital Key*), pengukuran jarak yang aman (*secure ranging*)	Memberikan perlindungan yang kuat terhadap serangan relay melalui pengukuran jarak; risiko yang ada mencakup manipulasi pengukuran jarak serta kelemahan dalam integrasi berbagai teknologi radio.
NFC	~4 cm	13,56 MHz	ISO/IEC 14443, ISO/IEC 18092	Membuka kunci kendaraan, akses cadangan	Jarak pendek mengurangi paparan, namun penyalahgunaan kloning dan mekanisme cadangan (*fallback*) dapat melampaui mekanisme akses yang lebih kuat jika tidak dikendalikan.

GNSS	Global (kilometer)	L1 (1,575 GHz), L2 (1,227 GHz), L5 (1,176 GHz)	GPS, GLONASS, Galileo, BeiDou, NavIC, ϙZSS	Navigasi, pelacakan	Rentan terhadap pemalsuan (*spoofing*) dan gangguan sinyal (*jamming*); hal ini berdampak pada fitur berbasis lokasi serta dapat memengaruhi aspek keselamatan dan integritas forensik.
V2X	Hingga 1 km	5,9 GHz (DSRC/C-V2X)	IEEE 802.11p, LTE-V2X	Berkendara kooperatif, pesan keselamatan	Antarmuka yang krusial bagi keselamatan; memerlukan autentikasi pesan yang kuat serta deteksi perilaku menyimpang guna mencegah pemalsuan (*spoofing*) atau peringatan palsu.

Wi-Fi

Hotspot Wi-Fi menyediakan akses internet bagi penumpang dan sering kali dihubungkan ke kendaraan melalui salah satu unit kontrol elektronik (ECU) berkinerja tinggi. Selain akses Wi-Fi yang disediakan oleh produsen kendaraan (OEM), perangkat Wi-Fi tambahan (*aftermarket*) juga dapat dihubungkan melalui *port On-Board Diagnostics* (OBD) untuk menghadirkan fitur konektivitas atau telemetri. Di samping hotspot, ECU tertentu dapat menawarkan konektivitas Wi-Fi untuk berkomunikasi dengan ponsel pengguna, misalnya saat proses parkir jarak jauh atau alur kerja servis. Beberapa sistem di dalam kendaraan juga mengandalkan tautan Wi-Fi khusus untuk komunikasi internal, alih-alih menggunakan jaringan kabel konvensional. Dengan demikian, Wi-Fi dapat hadir dalam kendaraan baik sebagai antarmuka yang menghadap ke luar maupun sebagai mekanisme transmisi internal.

Dongle Wi-Fi kendaraan memiliki berbagai tingkat jaminan keamanan, menjadikannya target yang diinginkan oleh pihak lawan. Karena sudah terpasang di port OBD, kerentanan yang dapat dieksploitasi pada dongle tersebut menempatkan penyerang langsung pada jaringan internal kendaraan, yang cukup berguna jika tujuan penyerang adalah untuk mencapai ECU atau sensor tertanam pada bus *Controller Area Network* (CAN). Contoh protokol yang berjalan di atas Wi-Fi meliputi *Transport Layer Security* (TLS), *Hypertext Transfer Protocol* (*Secure*) (HTTP(S)), WebSockets, *Message Influence Telemetry Transport* (MTTT), *Diagnostics over IP*

(DoIP), dan *Service-Oriented Vehicle Diagnostics* (SOVD), yang juga diperkenalkan secara singkat di bagian.

Bluetooth

Bluetooth banyak digunakan di kendaraan untuk menghubungkan ponsel pintar dan perangkat terdekat lainnya untuk komunikasi bebas genggam, streaming media, sinkronisasi data, dan interaksi layanan jarak pendek. Dalam sistem otomotif, Bluetooth dapat mendukung fitur yang menghadap pengguna dan komunikasi latar belakang antara kendaraan dan perangkat pendamping. Bluetooth Classic umumnya digunakan untuk integrasi audio dan telepon, sedangkan *Bluetooth Low Energy* (BLE) sering digunakan untuk pertukaran data berdaya rendah dan terputus-putus seperti pembaruan status, data sensor, dan penemuan kunci digital.

Penyerang sering menyalahgunakan kerentanan atau konfigurasi yang tidak aman dalam tumpukan perangkat lunak Bluetooth untuk menyuntikkan kode dan data mereka sendiri sebelum menemukan cara lain untuk meningkatkan hak akses dan melakukan serangan yang lebih invasif terhadap komponen kendaraan lainnya. Contoh profil Bluetooth umum untuk kasus penggunaan otomotif meliputi *Hands-Free Profile* (HFP), *Advanced Audio Distribution Profile* (A2DP), dan *Phone Book Access Profile* (PBAP). Selain itu, BLE, yang merupakan teknologi nirkabel berdaya rendah yang dirancang untuk pertukaran data jarak pendek dan terputus-putus, digunakan untuk aplikasi seperti *Generic Attribute Profile* (GATT) untuk status kendaraan dan data sensor. Selain itu, protokol Kunci Digital *Car Connectivity Consortium* (CCC) menggunakan kombinasi BLE, NFC, dan UWB untuk meningkatkan keamanan akses kendaraan.

Seluler

Konektivitas seluler merupakan salah satu mekanisme utama yang menghubungkan kendaraan dengan layanan cloud eksternal. Konektivitas ini umumnya digunakan untuk telematika, panggilan darurat, konektivitas armada, diagnostik jarak jauh, pembaruan perangkat lunak, serta fitur aplikasi seluler seperti membuka kunci kendaraan dari jarak jauh atau pengaturan suhu kabin (*climate control*). Dalam sebagian besar arsitektur, konektivitas ini disediakan melalui unit telematika atau unit kontrol konektivitas yang berfungsi sebagai jalur komunikasi jarak jauh utama bagi kendaraan.

Dalam beberapa tahun terakhir, unit telematika telah beralih menggunakan *embedded Subscriber Identity Module* (eSIM) versi digital dari kartu SIM konvensional yang digunakan pada perangkat seluler untuk autentikasi dan koneksi ke jaringan seluler. Berbeda dengan kartu SIM fisik, eSIM tertanam langsung di dalam sirkuit modem seluler dan mendukung jaringan 4G/5G. Celah keamanan kritis dalam desain SIM/eSIM terbukti dapat memfasilitasi berbagai jenis serangan, seperti penyamaran (*impersonation*), serangan *man-in-the-middle*, dan penyadapan lalu lintas data. Protokol yang berjalan di atas jaringan seluler sering kali serupa dengan yang digunakan pada Wi-Fi, mencakup TLS, HTTP(S), WebSockets, MQTT, DoIP, dan SOVD.

Frekuensi Radio (RF)

Sistem *Remote Keyless Entry* (RKE) menggunakan sinyal RF untuk mengunci dan membuka pintu kendaraan secara nirkabel, sementara sistem *Passive Keyless Entry and Start* (PKES) mengandalkan komunikasi berbasis jarak dekat untuk memungkinkan akses tanpa perlu mengeluarkan kunci serta untuk menyalakan mesin. Sistem *immobilizer* menggunakan komunikasi RF frekuensi rendah sebagai bagian dari proses otorisasi penyalaan mesin, sedangkan *Tire Pressure Monitoring Systems* (TPMS) menggunakan sensor RF untuk mengirimkan data tekanan ban ke kendaraan.

Banyak dari sistem RF otomotif ini beroperasi pada pita frekuensi *Industrial, Scientific, and Medical* (ISM) yang tidak memerlukan lisensi, serta sering kali mengandalkan protokol khusus dari vendor tertentu alih-alih menggunakan standar yang berlaku umum. Saluran RF kerap dieksploitasi untuk tindak pencurian kendaraan maupun untuk mendapatkan akses awal ke unit kontrol elektronik (*Electronic Control Unit* atau ECU) yang mendukung komunikasi berbasis RF.

***Ultra-wideband* (UWB)**

UWB menggunakan pulsa RF pita lebar (*wide-band*) berdurasi singkat untuk memungkinkan pengukuran jarak dan arah yang sangat akurat antara kendaraan dan perangkat eksternal. Dalam sistem otomotif, UWB terutama digunakan untuk akses kendaraan yang aman, khususnya pada implementasi kunci digital modern yang memerlukan pengukuran jarak secara presisi guna mencegah serangan *relay* (perantara). Berbeda dengan sistem *keyless entry* berbasis RF konvensional, UWB memungkinkan kendaraan untuk memverifikasi bahwa kunci tersebut benar-benar berada di dekat kendaraan dan di lokasi spesifik relatif terhadap kendaraan tersebut. UWB sering dikombinasikan dengan BLE dan NFC untuk keperluan penemuan perangkat (*device discovery*), autentikasi, serta mekanisme akses cadangan.

Dari perspektif keamanan siber, meskipun UWB secara signifikan meningkatkan perlindungan terhadap pencurian berbasis *relay*, teknologi ini tetap rentan terhadap berbagai serangan, seperti manipulasi pengukuran jarak, kelemahan dalam implementasi protokol, penyalahgunaan penyediaan kunci (*key provisioning*), serta kerentanan dalam logika peralihan (*handover*) antar-radio.

***Near Field Communication* (NFC)**

NFC semakin populer dalam sistem otomotif karena memungkinkan interaksi jarak dekat yang sederhana antara kendaraan dan perangkat eksternal. Contoh penggunaan yang umum meliputi penyandingan perangkat (*device pairing*), penyediaan kredensial, dan alur kerja kunci digital, seperti membuka kunci dengan cara menempelkan perangkat (*tap-to-unlock*) atau sebagai akses cadangan. Karena NFC memerlukan kedekatan fisik, teknologi ini sering digunakan untuk tindakan yang memang disengaja oleh pengguna yang hadir di lokasi, bukan untuk komunikasi nirkabel yang berkelanjutan.

Dalam banyak arsitektur kunci digital, NFC bekerja bersama BLE dan UWB sebagai bagian dari desain akses multi-radio yang lebih luas. Sistem NFC menghadapi beberapa potensi serangan, seperti penyadapan (*eavesdropping*), di mana pihak yang tidak berwenang mencegat komunikasi menggunakan antena dari jarak yang lebih jauh daripada jangkauan

tipikal NFC (10 cm). NFC juga rentan terhadap serangan relai (*relay attack*) yang memanfaatkan beberapa perangkat untuk memperluas jangkauan NFC, sehingga memungkinkan terjadinya transaksi yang tidak sah.

***Global Navigation Satellite System* (GNSS)**

GNSS digunakan dalam sistem otomotif untuk navigasi, serta semakin sering dimanfaatkan untuk lokalisasi dan sinkronisasi waktu. Pada kendaraan modern, data GNSS tidak hanya digunakan oleh sistem navigasi pada perangkat hiburan (*infotainment*), tetapi juga oleh sistem telematika, layanan manajemen armada, serta fungsi bantuan pengemudi tingkat lanjut (ADAS) atau sistem kemudi otonom. Meskipun GNSS sering dianggap sebagai fitur navigasi, teknologi ini juga dapat berfungsi sebagai input pendukung bagi sistem kendaraan lain yang bergantung pada informasi posisi atau waktu yang akurat.

Sistem GNSS rentan terhadap berbagai serangan yang mengeksploitasi daya rendah dan sifat sinyalnya yang terbuka. Ancaman utama meliputi jamming yaitu menutupi sinyal GNSS dengan derau (*noise*) yang lebih kuat, sehingga menyebabkan gangguan layanan seperti serangan *Denial of Service* (DoS) serta spoofing, di mana pihak yang tidak berwenang memancarkan sinyal palsu untuk mengecoh penerima, yang mengakibatkan pergeseran data waktu maupun lokasi. Karena banyak kendaraan mengandalkan GNSS untuk menyinkronkan unit kontrol elektronik (ECU) internal ke sumber waktu yang sama, serangan spoofing dapat berdampak serius terhadap keselamatan kendaraan jika sumber waktu tersebut menjadi tidak dapat diandalkan.

***Vehicle-to-Everything Communication* (V2X)**

Tujuan dari komunikasi V2X adalah untuk meningkatkan keselamatan jalan raya, efisiensi lalu lintas, dan kenyamanan berkendara dengan memungkinkan kendaraan untuk bertukar informasi dengan kendaraan lain infrastruktur di tepi jalan, pejalan kaki, serta layanan jaringan. Hal ini mencakup komunikasi antar-kendaraan (*Vehicle-to-Vehicle*/V2V), kendaraan ke infrastruktur (*Vehicle-to-Infrastructure*/V2I), kendaraan ke pejalan kaki (*Vehicle-to-Pedestrians*/V2P), dan kendaraan ke jaringan (*Vehicle-to-Networks*/V2N). Sistem V2X dirancang untuk mendukung pertukaran data lalu lintas, bahaya, dan kesadaran situasional dengan latensi rendah, sehingga sangat relevan untuk fungsi-fungsi yang berorientasi pada keselamatan dan mengemudi kooperatif.

Keamanan siber, tentu saja, telah menjadi perhatian sejak awal pengembangan protokol V2X, mengingat adanya potensi pelanggaran privasi pengguna serta kemungkinan pemalsuan *Basic Safety Messages* (BSM) yang dapat menimbulkan konsekuensi berbahaya. Serangan pasif terhadap V2X melibatkan penyadapan saluran komunikasi untuk mengumpulkan data sensitif seperti lokasi dan perilaku kendaraan yang mengancam privasi pengguna dan memungkinkan pelacakan. Di sisi lain, serangan aktif melibatkan penyisipan pesan palsu, pemalsuan identitas, atau gangguan komunikasi, yang berpotensi menyebabkan kecelakaan, kemacetan lalu lintas, atau kegagalan fungsi sistem.

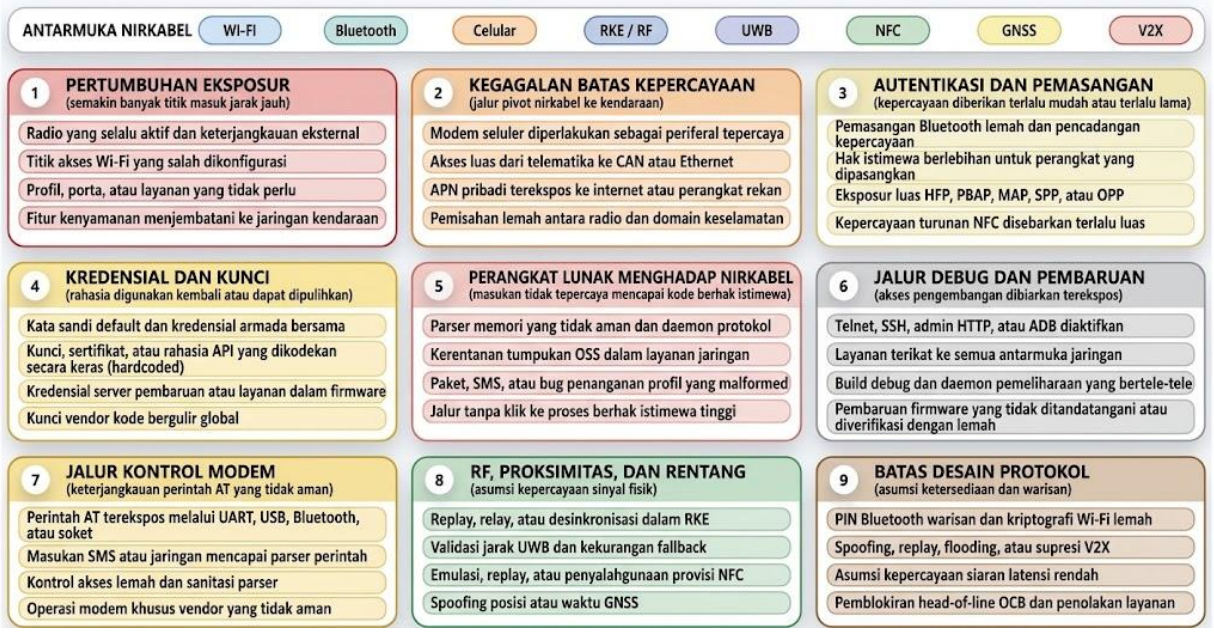
Secara keseluruhan, antarmuka nirkabel ini mendukung berbagai fungsi kendaraan, mulai dari fitur kenyamanan dan telematika hingga kontrol akses serta mengemudi kooperatif. Dengan memahami lanskap protokol ini, kita dapat menelaah kelemahan-kelemahan yang

kerap muncul dalam teknologi nirkabel tersebut, serta asumsi kepercayaan (*trust assumptions*) yang diperkenalkannya ke dalam arsitektur kendaraan.

5.3 KELEMAHAN PADA SISTEM NIRKABEL KENDARAAN MODERN

Sebagaimana telah kita lihat sebelumnya, konektivitas nirkabel telah menjadi karakteristik utama kendaraan modern. Berbagai fitur seperti layanan jarak jauh, pembaruan perangkat lunak secara nirkabel (*over-the-air*), integrasi ponsel pintar, dan sistem mengemudi kooperatif, semuanya bergantung pada komunikasi berkelanjutan dengan jaringan eksternal. Meskipun kemampuan ini memberikan manfaat fungsional dan komersial yang signifikan, hal tersebut juga meningkatkan kerentanan kendaraan terhadap serangan jarak jauh sebuah risiko yang sebagian besar tidak ada pada arsitektur otomotif generasi sebelumnya.

Bagian ini mengkaji kelemahan-kelemahan umum yang ditemukan dalam sistem nirkabel kendaraan. Alih-alih berfokus pada kerentanan individual, kami menganalisis pola-pola berulang yang muncul di berbagai teknologi, perangkat lunak, dan sistem. Kelemahan-kelemahan ini berakar pada asumsi arsitektural, pilihan desain warisan (*legacy design*), serta tantangan dalam mengintegrasikan teknologi nirkabel kelas konsumen ke dalam lingkungan yang mengutamakan keselamatan (*safety-critical environments*). Tujuan dari bagian ini adalah memberikan pemahaman yang jelas mengenai bagaimana dan mengapa sistem nirkabel kendaraan dapat diserang, sekaligus menjadi landasan bagi teknik pengujian penetrasi praktis yang akan dibahas pada bagian berikutnya. Gambar 5.2 menyajikan gambaran umum mengenai kelemahan-kelemahan yang sering ditemukan dalam sistem nirkabel kendaraan:

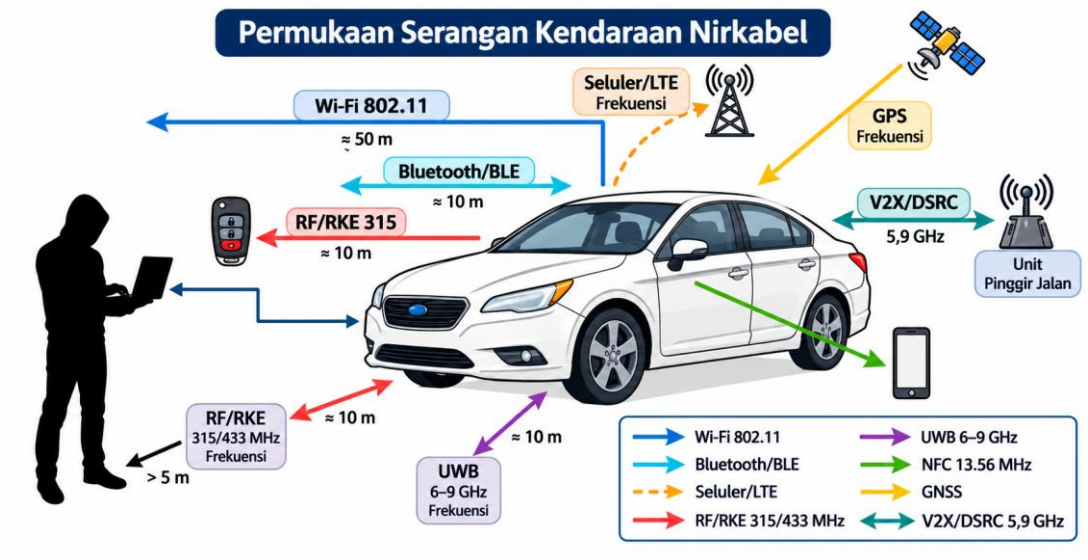


Gambar 5.2 Kelemahan umum pada sistem nirkabel kendaraan

Perluasan Permukaan Serangan Nirkabel

Mungkin kelemahan pertama dan paling mendasar dari antarmuka nirkabel pada kendaraan adalah bahwa antarmuka tersebut memperluas permukaan serangan kendaraan.

Meskipun menghilangkan akses nirkabel bukan lagi sebuah pilihan, setidaknya meminimalkan permukaan serangan hingga ke tingkat akses yang paling terbatas tetaplah langkah yang diinginkan. Gambaran umum mengenai perluasan permukaan serangan nirkabel ditunjukkan pada Gambar 5.3:



Gambar 5.3 Perluasan permukaan serangan nirkabel pada kendaraan modern

Di seluruh sistem akses berbasis Wi-Fi, Bluetooth, seluler, NFC, UWB, V2X, dan RF, muncul serangkaian kelemahan yang berulang: kepercayaan berlebih terhadap input eksternal, isolasi yang lemah antara domain nirkabel dan sistem internal kendaraan, perilaku *fallback* (cadangan) yang tidak aman, pengelolaan kredensial atau kunci yang buruk, serta validasi data nirkabel yang tidak memadai sebelum data tersebut memengaruhi fungsi-fungsi dengan hak akses lebih tinggi. Mari kita cermati bagaimana kelemahan-kelemahan ini berlaku pada setiap antarmuka nirkabel:

- Pada sistem kendaraan yang dilengkapi Wi-Fi, kelemahan sering kali muncul akibat titik akses (*access point*) yang salah dikonfigurasi, skema enkripsi yang kedaluwarsa, penggunaan kredensial bersama, atau terbukanya layanan internal secara tidak sengaja setelah klien terhubung. Karena Wi-Fi sering kali menjadi jembatan yang menghubungkan perangkat eksternal ke platform kendaraan dengan *bandwidth* lebih tinggi, segmentasi yang buruk dapat menyebabkan kompromi pada fitur kenyamanan berkembang menjadi akses yang lebih luas ke dalam sistem.
- Kelemahan terkait Bluetooth sering kali bersumber dari alur kerja pemasangan (*pairing*) yang tidak aman, hubungan kepercayaan yang terlalu persisten, hak akses berlebih yang diberikan kepada perangkat yang dipasangkan, serta paparan layanan yang jauh melampaui kebutuhan fungsionalnya. Dalam praktiknya, implementasi profil, penyimpanan *cache* kepercayaan (*trust caching*), dan kontrol akses yang tidak memadai terhadap layanan *Classic Bluetooth* atau BLE dapat mengubah peristiwa pemasangan sementara menjadi jalur serangan yang bertahan lama. Sebagai contoh, kendaraan mungkin mengekspos kontak *hands-free*, pesan, media, diagnostik, atau

layanan eksklusif melalui Bluetooth terkadang tanpa autentikasi atau dengan memanfaatkan mekanisme pemasangan yang lemah. Profil Bluetooth yang relevan untuk mendukung kasus penggunaan semacam ini mencakup *Hands-Free Profile* (HFP), *Phone Book Access Profile* (PBAP), *Message Access Profile* (MAP), *Audio/Video Remote Control Profile* (AVRCP), *Serial Port Profile* (SPP), dan *Object Push Profile* (OPP). Setelah perangkat dipasangkan, berkat adanya penyimpanan *cache* kepercayaan, akses persisten umumnya dapat diperoleh, menjadikan Bluetooth sebagai titik masuk yang menarik bagi penyerang.

- Sistem akses berbasis teknologi nirkabel sering kali memiliki kelemahan dalam hal ketahanan terhadap serangan *replay* (pengulangan pesan), pengelolaan kunci kriptografis, logika sinkronisasi, serta asumsi desain protokol terkait kedekatan (*proximity*) dan keaslian (*authenticity*). Karena sistem ini secara langsung memengaruhi otorisasi untuk masuk ke dalam kendaraan dan menyalakan mesin, kelemahan desain sekecil apa pun dapat menimbulkan konsekuensi yang sangat besar.
- Kelemahan yang terkait dengan UWB (*Ultra-Wideband*) sering ditemukan tidak hanya pada protokol pengukuran jarak itu sendiri, melainkan juga pada desain sistem secara keseluruhan. Masalah yang umum terjadi meliputi validasi pengukuran jarak yang tidak memadai, transfer kepercayaan (*trust transfer*) yang tidak aman antar perangkat radio, penanganan mekanisme cadangan (*fallback*) yang lemah, serta cacat implementasi dalam logika yang menggabungkan UWB dengan teknologi lain seperti BLE, NFC, atau mekanisme akses tanpa kunci (*keyless-entry*) konvensional.
- Kelemahan yang terkait dengan NFC biasanya muncul dalam alur kerja penyediaan kredensial, inisialisasi kepercayaan (*trust bootstrapping*), otorisasi di sisi *backend*, atau cara kepercayaan yang diperoleh melalui NFC diperluas ke fungsi-fungsi kendaraan dengan hak akses lebih tinggi. Meskipun antarmuka radio NFC memiliki jangkauan yang terbatas, operasi yang dipicu oleh peristiwa NFC tetap dapat berdampak luas terhadap keamanan jika tidak dibatasi dengan ketat.
- Kelemahan pada GNSS (*Global Navigation Satellite System*) muncul akibat ketergantungan yang berlebihan pada data posisi dan waktu yang diterima dari luar, tanpa adanya validasi, korelasi, atau logika cadangan yang memadai. Jika data GNSS dipercaya secara berlebihan, manipulasi lokasi atau waktu dapat memengaruhi navigasi, sinkronisasi, *geofencing*, serta fungsi-fungsi kendaraan lainnya yang bergantung pada data tersebut.
- Kelemahan yang terkait dengan V2X (*Vehicle-to-Everything*) umumnya bersumber dari asumsi kepercayaan pada tingkat protokol, terbatasnya peluang untuk menerapkan autentikasi berbasis sesi yang kuat, tantangan dalam memvalidasi pesan keselamatan yang disiarkan (*broadcast*), serta kesulitan operasional dalam melakukan pencabutan akses (*revocation*) dan menangani perilaku yang tidak semestinya dalam skala besar. Karena sistem ini mendukung pengambilan keputusan yang berkaitan dengan keselamatan, validasi pesan yang lemah atau kepercayaan berlebih terhadap data yang diterima dapat menciptakan risiko yang sangat besar.

Arsitektur Modul Seluler dan Batasan Kepercayaan (*Trust Boundaries*)

Unit telematika seluler sering kali dianggap sekadar sebagai perangkat periferal komunikasi, padahal pada kenyataannya, unit tersebut merupakan platform komputasi yang canggih. Modul 4G atau 5G pada umumnya dilengkapi dengan prosesor, memori, penyimpanan, dan sistem operasi sendiri sering kali berupa sistem Linux tertanam (*embedded Linux*). Modul ini dapat berkomunikasi dengan ECU *gateway* atau terhubung langsung ke jaringan internal kendaraan.

Salah satu kelemahan mendasar dari antarmuka seluler terletak pada kepercayaan implisit yang diberikan kepada modul-modul tersebut. Setelah diintegrasikan, modul ini sering kali diberi akses luas ke jaringan kendaraan, sementara di saat yang sama dapat dijangkau dari infrastruktur seluler eksternal. Hal ini meruntuhkan batasan kepercayaan antara jaringan eksternal dan sistem internal kendaraan, sehingga menjadikan modul seluler sebagai target bernilai tinggi yang dapat digunakan oleh penyerang jarak jauh sebagai titik masuk.

Kegagalan Isolasi Jaringan dan APN Privat

Private Access Point Name (APN) merupakan metode untuk menciptakan jaringan khusus yang terisolasi bagi kelompok perangkat tertentu, di mana kebijakan keamanan dapat diterapkan dengan lebih ketat. Tujuan utamanya adalah memastikan bahwa hanya perangkat dan pengguna yang berwenang yang memiliki akses ke segmen jaringan terlindung ini, sehingga mengurangi paparan terhadap ancaman eksternal. Sebagai contoh, APN dapat mengisolasi lalu lintas data armada kendaraan dari internet publik dengan mengarahkannya melalui jaringan yang dikendalikan oleh operator. Namun, konfigurasi jaringan operator dan APN privat yang tidak tepat dapat menggagalkan fungsi APN tersebut, yang mengakibatkan layanan tertentu pada sistem kendaraan nirkabel menjadi dapat diakses melalui internet publik.

Dalam kasus lain jaringan mungkin tidak memiliki isolasi yang memadai antarperangkat, sehingga kendaraan yang terhubung ke APN yang sama dapat saling menemukan dan berkomunikasi secara langsung. Kurangnya segmentasi ini memungkinkan terjadinya pergerakan lateral (*lateral movement*) jika salah satu perangkat di dalam jaringan tersebut berhasil disusupi. Penyerang yang mendapatkan akses ke APN yang digunakan oleh armada kendaraan misalnya melalui kartu SIM yang telah disusupi atau perangkat ilegal dapat memindai layanan yang terbuka dan mengeksploitasi berbagai kendaraan tanpa perlu berinteraksi dengan internet publik sama sekali.

Antarmuka Perintah AT dan Jalur Kendali yang Tidak Aman

Perintah *Attention* (AT), yang distandarisasi dalam 3GPP TS 27.007 pada awalnya dirancang untuk mengendalikan modem *dial-up*. Seiring berjalannya waktu, produsen modul telah memperluas perintah ini dengan menambahkan perintah khusus (*proprietary commands*), yang beberapa di antaranya mengekspos fungsi tingkat sistem tanpa validasi input yang memadai. Perintah-perintah ini berbasis teks dan memungkinkan perangkat lunak untuk mengonfigurasi parameter radio, mengelola pesan *Short Message Service* (SMS), serta mengendalikan sesi data. Karena perintah AT telah distandarisasi sampai batas tertentu oleh badan seperti 3GPP, sebagian besar modul seluler memiliki sekumpulan perintah inti yang

sama, namun mungkin juga menyertakan ekstensi khusus dari vendor tertentu. Dalam banyak sistem kendaraan, perintah AT (*AT commands*) dapat diakses melalui berbagai jalur, termasuk *Universal Asynchronous Receiver/Transmitter* (UART), *Universal Serial Bus* (USB), Bluetooth, dan soket jaringan.

Kelemahan utamanya terletak pada kurangnya kontrol akses yang memadai serta minimnya validasi input. Pengurai (*parser*) perintah AT dapat menerima perintah dari sumber yang tidak tepercaya. Sebagai contoh, pada beberapa perangkat, pesan SMS yang dikirim melalui jaringan (*over-the-air*) dapat dirancang khusus untuk menyisipkan perintah AT, yang berpotensi memungkinkan penyerang untuk mengubah pengaturan sistem atau memicu tindakan berbahaya. Selain itu, logika penguraian yang buruk dan kurangnya sanitasi data semakin meningkatkan risiko injeksi perintah serta perilaku sistem yang tidak diinginkan.

Kerentanan Perangkat Lunak Pada Komponen yang Berhubungan Dengan Nirkabel

Perangkat lunak yang berhubungan dengan nirkabel merupakan sumber umum dari kerentanan kritis. *Network daemon*, pengurai protokol (*protocol parser*), dan penangan pesan (*message handler*) memproses input yang tidak tepercaya pada tingkat hak akses (*privilege level*) yang tinggi, serta sering kali ditulis menggunakan bahasa pemrograman yang tidak aman dalam hal pengelolaan memori, seperti bahasa pemrograman C. Kerentanan seperti *buffer overflow*, penulisan di luar batas (*out-of-bounds write*), dan masalah serupa lainnya masih terus ditemukan. Selain itu, perangkat lunak sumber terbuka (*Open-Source Software* atau OSS) sering digunakan untuk berbagai komponen, seperti *stack* komunikasi (contohnya BlueZ) dan layanan jaringan (contohnya dnsmasq, ConnMan). Karena praktik pengembangan yang aman serta pendekatan pengujian dapat bervariasi di antara berbagai proyek OSS, sistem pada kendaraan pun mewarisi kelemahan OSS yang sama, yang juga berdampak pada rantai pasok otomotif secara lebih luas.

Istilah:

- *Network daemon* (daemon jaringan) dalam sistem Linux tertanam (*embedded Linux*) adalah proses latar belakang yang mengelola layanan atau fungsionalitas terkait jaringan. Daemon ini memfasilitasi komunikasi antarperangkat, menangani protokol jaringan, atau menyediakan layanan khusus seperti akses jarak jauh, berbagi berkas, atau sinkronisasi waktu.
- *Buffer overflow* (luapan penyangga) adalah jenis kerentanan perangkat lunak yang terjadi ketika sebuah program menulis data ke dalam *buffer* (blok memori berukuran tetap) melebihi kapasitas yang dapat ditampungnya.
- Data berlebih ini dapat menimpa lokasi memori di sekitarnya, yang menyebabkan perilaku yang tidak diinginkan, seperti kerusakan sistem (*crash*), kerusakan data, atau eksekusi kode berbahaya.
- *Out-of-bounds write* (penulisan di luar batas) adalah kerentanan perangkat lunak di mana sebuah program menulis data di luar batas yang dialokasikan untuk suatu objek memori (misalnya, *array* atau *struct*), sehingga mengubah memori yang seharusnya tidak diakses. Kerentanan ini dapat menyebabkan program mengalami kerusakan (*crash*), kerusakan data, atau dieksploitasi oleh penyerang untuk menjalankan kode arbitrer (*arbitrary code*) maupun meningkatkan hak akses (*privilege escalation*).

Ketika kerentanan semacam itu dapat diakses dari jarak jauh, kerentanan tersebut memungkinkan serangan *zero-click* yang tidak memerlukan interaksi pengguna sama sekali. Dengan menggunakan berbagai teknik eksploitasi, penyerang dapat melampaui langkah-langkah mitigasi modern dan memperoleh kendali permanen atas sistem.

Istilah:

Serangan *zero-click* adalah serangan yang mengeksploitasi kerentanan keamanan pada perangkat lunak atau perangkat keras tanpa memerlukan interaksi apa pun dari korban. Berbeda dengan eksploitasi TI pada umumnya yang sering kali mengharuskan pengguna untuk mengeklik tautan, membuka lampiran, atau melakukan tindakan tertentu, serangan *zero-click* terpicu secara otomatis biasanya saat sistem yang rentan menerima input yang telah dirancang secara berbahaya.

Kredensial yang Lemah dan Rahasia yang Tertanam Dalam Kode (*Hard-Coded*)

Masalah yang sering muncul dalam sistem kendaraan nirkabel adalah ketergantungan pada mekanisme autentikasi statis. Sebagai contoh, kata sandi bawaan (*default*), kredensial yang digunakan bersama, dan kunci kriptografi yang tertanam langsung di dalam kode (*hard-coded*) sering ditemukan pada modul seluler, sistem hiburan (*infotainment*), dan layanan diagnostik. Kredensial ini dirancang untuk melindungi akses ke *shell*, mengamankan komunikasi, mekanisme pembaruan *firmware*, atau API. Salah satu contoh kerentanan adalah ketika sistem kendaraan nirkabel memuat kredensial *File Transfer Protocol* (FTP) yang tertanam dalam kode untuk mengakses server pembaruan jarak jauh. Penyalahgunaan kredensial FTP ini memungkinkan penyerang untuk menargetkan server pembaruan tersebut guna mengunggah perangkat lunak berbahaya.

Lebih buruk lagi, terdapat kasus di mana fungsi pembaruan *firmware* pada sistem kendaraan nirkabel tidak memiliki verifikasi tanda tangan (*signature verification*) yang memadai, sehingga penyerang dapat memasang perangkat lunak berbahaya hanya dengan mengirimkan paket yang tidak ditandatangani atau ditandatangani secara tidak tepat ke kendaraan tersebut. Kerentanan umum lainnya adalah penggunaan kembali kredensial yang sama di seluruh armada kendaraan, yang secara signifikan memperbesar dampak jika terjadi kompromi keamanan. Artinya, begitu satu kata sandi atau kunci berhasil diperoleh dari satu kendaraan misalnya melalui analisis *firmware*, rekayasa balik (*reverse engineering*), atau kebocoran dokumentasi kredensial tersebut sering kali dapat digunakan secara universal untuk seluruh armada.

Layanan Jaringan dan Antarmuka *Debug*

Subsistem nirkabel sering kali mengekspos layanan jaringan tambahan di luar fungsi utamanya. Sebagai contoh, layanan seperti Telnet, *Secure Shell* (SSH), antarmuka administrasi berbasis HTTP, serta *daemon* untuk *debugging* seperti *Android Debug Bridge* (ADB) sering kali diaktifkan untuk mendukung pengembangan dan pemeliharaan jarak jauh. Jika layanan-layanan ini tetap aktif pada sistem produksi, hal tersebut menciptakan titik akses jarak jauh yang persisten.

Selain itu, kesalahan konfigurasi dapat memperparah kerentanan ini. Layanan mungkin diatur untuk mendengarkan (*listen*) pada semua antarmuka jaringan, alih-alih dibatasi hanya

pada *localhost*. Kondisi ini membuat layanan tersebut terpapar terhadap ancaman eksternal. Perlu dicatat pula bahwa kerentanan ini sering kali muncul bersamaan dengan kredensial lemah dan rahasia yang tertanam dalam kode, sebagaimana disebutkan sebelumnya. Layanan-layanan ini sering kali menggunakan kata sandi umum seperti "oelinux123" atau bahkan tidak memerlukan autentikasi sama sekali. Begitu berhasil diakses, layanan ini sering kali memberikan kendali dengan hak akses tinggi atas sistem yang mendasarinya, dan dapat berfungsi sebagai batu loncatan untuk serangan lebih lanjut. Sebagai contoh, penyerang dapat memasang *backdoor*, menyebarkan perangkat lunak berbahaya (*malware*), atau mencuri data sensitif.

Sistem RF dan Kriptografi *Rolling Code*

Sistem *Remote Keyless Entry* (RKE) sering kali mengandalkan *rolling code* (kode yang terus berubah) untuk mencegah serangan *replay* (pengulangan). Pada prinsipnya, mekanisme *rolling code* bekerja dengan cara membuat perangkat kendali jarak jauh (*key fob*) dan penerima (di dalam kendaraan) menghasilkan urutan kode yang hanya berlaku satu kali; setelah satu kode digunakan, kedua pihak akan beralih ke nilai berikutnya. Pendekatan ini mencegah serangan *replay*, di mana penyerang menangkap dan mengirimkan kembali kode yang sebelumnya sah untuk membuka kunci atau menyalakan kendaraan.

Namun, dalam implementasi RKE tertentu, kunci kriptografi statis atau global ditanamkan ke dalam semua perangkat yang diproduksi oleh vendor terkait. Jika penyerang mampu melakukan rekayasa balik (*reverse engineering*) terhadap sistem tersebut dan memperoleh kunci global ini, mereka dapat menghasilkan *rolling code* yang sah kapan pun diinginkan. Begitu kunci-kunci ini terkompromi, mekanisme *rolling code* tidak lagi memberikan perlindungan yang berarti, sehingga memungkinkan penyerang untuk membuka kunci kendaraan dari jarak jauh sesuka hati.

Keterbatasan Desain Protokol Nirkabel

Selama bertahun-tahun, protokol nirkabel telah mengalami berbagai kelemahan desain. Sebagai contoh, versi awal Bluetooth menggunakan pemasangan (*pairing*) berbasis PIN (sering kali terdiri dari 4 digit), yang membuatnya rentan terhadap serangan *brute force*. Selain itu, teknologi Wi-Fi generasi awal memiliki kelemahan desain kriptografi pada *Wired Equivalent Privacy* (WEP), yang memungkinkan penyerang menghitung kunci enkripsi berdasarkan informasi yang diperoleh dari lalu lintas nirkabel yang berhasil ditangkap. Di sisi lain, protokol komunikasi V2X (*Vehicle-to-Everything*) mungkin meminimalkan beban autentikasi karena lebih mengutamakan latensi rendah dan ketersediaan guna mendukung kasus penggunaan yang krusial bagi keselamatan. Hal ini membuat protokol tersebut rentan terhadap serangan *spoofing* (penyamaran), *denial-of-service* (penolakan layanan), dan penekanan pesan (*message suppression*).

Sebagai contoh, mekanisme konfirmasi (*acknowledgment*) *unicast* dalam standar IEEE 802.11p pada mode *Outside the Context of a Basic Service Set* (OCB) dapat menyebabkan kondisi *head-of-line blocking*. Kondisi ini mampu menghentikan seluruh transmisi keluar dari kendaraan atau *Roadside Unit* (RSU) termasuk pesan keselamatan yang disiarkan (*broadcast*) tanpa memerlukan adanya kerentanan dalam implementasinya. Saat menyusun rangkaian

serangan nirkabel, sangat bermanfaat untuk menganalisis protokol nirkabel yang digunakan guna menentukan apakah terdapat kelemahan desain mendasar atau kesalahan implementasi yang dapat dieksploitasi.

5.4 PENGUJIAN PENETRASI PADA SISTEM NIRKABEL KENDARAAN

Serangan nirkabel pada kendaraan bukanlah sekadar teori, melainkan didasarkan pada pengintaian sistematis, analisis cermat terhadap batasan kepercayaan (*trust boundaries*), serta penggabungan berbagai celah keamanan secara kreatif untuk membentuk jalur serangan yang efektif. Bagian ini memperkenalkan aktivitas pengujian penetrasi konkret yang memanfaatkan beberapa kelemahan yang telah dijelaskan sebelumnya.

Di sini dipaparkan perangkat, teknik, dan alur kerja yang digunakan oleh peneliti keamanan serta penguji profesional untuk mengevaluasi sistem nirkabel kendaraan dalam model ancaman yang realistis. Fokus utama diberikan pada serangan jarak jauh (*remote*) dan serangan dengan interaksi rendah (*low-interaction*), yang mencerminkan risiko paling kritis bagi armada kendaraan dalam skala luas. Dengan merinci apa yang dicari oleh penguji, bagaimana mereka menguji setiap antarmuka nirkabel, dan bagaimana temuan-temuan individual digabungkan menjadi skenario kompromi yang lebih luas, bagian ini membekali pihak pembela maupun penguji dengan pemahaman praktis mengenai pengujian penetrasi sistem nirkabel kendaraan.

Penemuan dan Validasi Permukaan Serangan Nirkabel

Pengujian penetrasi pada sistem nirkabel kendaraan dimulai dengan penemuan dan validasi sistematis terhadap semua antarmuka radio dan protokol nirkabel yang ada pada kendaraan. Fase ini berfokus untuk menentukan apa saja yang benar-benar terekspos, bagaimana perilakunya dalam kondisi serangan, serta apakah integrasinya sesuai dengan rancangan arsitektur yang dimaksudkan. Hasil dari fase ini adalah inventaris berbasis bukti yang mencakup perangkat radio yang dapat diakses, layanan yang terekspos, perilaku pemasangan (*pairing*) atau asosiasi, format pesan yang teramati, serta hubungan kepercayaan antara antarmuka nirkabel dan sistem internal kendaraan.

- Pengujian Wi-Fi biasanya dimulai dengan penemuan pasif maupun aktif menggunakan perangkat seperti Kismet, airodump-ng, dan iw. Dengan perangkat tersebut, penguji penetrasi dapat mengidentifikasi titik akses (*access point*), mode enkripsi, *Service Set Identifier* (SSID) yang tersembunyi, serta perilaku klien. Dalam praktiknya, penguji menangkap lalu lintas beacon dan probe, menentukan apakah kendaraan mengiklankan atau secara otomatis bergabung dengan jaringan yang dikenal, lalu memvalidasi apakah asosiasi ke jaringan tersebut mengekspos layanan IP atau jalur diagnostik. Setelah konektivitas terbentuk, perangkat seperti Nmap, masscan, dan netcat digunakan untuk melakukan enumerasi terhadap layanan yang dapat diakses serta topologi jaringan internal. Salah satu aspek yang menjadi fokus adalah apakah konektivitas Wi-Fi terhubung langsung ke jaringan Ethernet internal atau jaringan yang berdekatan dengan CAN (*CAN-adjacent networks*).

- Pengujian Bluetooth mencakup pemindaian perangkat Classic dan BLE menggunakan alat seperti Ubertooth, bluetoothctl, btmgmt, hcitool, dan gatttool, serta kerangka kerja yang lebih canggih seperti Btlejack dan utilitas debugging BlueZ. Penguji penetrasi (*penetration tester*) menganalisis alur kerja pemasangan (*pairing*), penemuan layanan (*service discovery*), dan izin atribut. Untuk BLE, proses ini meliputi enumerasi layanan GATT, pembacaan dan penulisan karakteristik (jika diizinkan), serta pemeriksaan apakah operasi sensitif memerlukan pemasangan (*pairing*), pengikatan (*bonding*), atau otorisasi pada lapisan aplikasi.
- Sistem *Keyless Entry* (KE) berbasis frekuensi radio (RF) diperiksa menggunakan perangkat *software-defined radio* (SDR) seperti HackRF, *Yard Stick One*, atau USRP, yang dikombinasikan dengan perangkat lunak seperti GNU Radio dan Universal Radio Hacker. Penguji penetrasi menggunakan perangkat ini untuk menangkap transmisi dari *key fob* (remote kunci). Langkah awal biasanya berupa penangkapan pasif (*passive capture*) untuk menentukan frekuensi, modulasi, struktur paket, pengaturan waktu (*timing*), serta apakah pesan berubah secara dapat diprediksi setiap kali tombol ditekan. Selanjutnya, penguji akan menganalisis perilaku modulasi dan protokol, serta mengevaluasi ketahanan terhadap serangan *replay*, sinkronisasi *rolling code*, dan kekuatan kriptografinya. Fokus pengujian tidak hanya terbatas pada serangan *replay*, tetapi juga mencakup kemungkinan serangan *relay*, desinkronisasi, serta identifikasi apakah kunci yang sama digunakan kembali untuk berbagai peristiwa atau sistem otentikasi yang berbeda.
- Pengujian UWB memperluas analisis RF hingga mencakup pengukuran *time-of-flight* (waktu tempuh sinyal) dan validasi jarak. Perangkat pengembangan UWB khusus serta penganalisis protokol digunakan untuk mengamati pertukaran data penentuan jarak dan memvalidasi perhitungan jarak tersebut. Pengujian praktis dilakukan dengan membandingkan jarak yang dilaporkan terhadap jarak fisik yang terkontrol, penambahan penundaan (*delay*), serta perilaku sistem saat penentuan jarak melalui UWB gagal atau menjadi tidak dapat diandalkan. Fokus pengujian adalah untuk mengetahui apakah UWB dapat dimanipulasi, ditunda, atau dilewati (*bypassed*), serta apakah mekanisme *fallback* (pengalihan) ke sistem *Remote Keyless Entry* (RKE) lama menciptakan celah yang dapat dieksploitasi melalui penurunan tingkat keamanan (*downgrade*).
- Antarmuka NFC diuji menggunakan perangkat seperti Proxmark3, dan Chameleon Mini untuk meniru kartu (*emulasi*), melakukan *replay* pada pertukaran data, serta memeriksa alur kerja otentikasi. Alih-alih hanya berfokus pada kekuatan kriptografi, penguji penetrasi dapat meneliti bagaimana kepercayaan yang berasal dari NFC disebarkan ke dalam sistem kendaraan. Pertanyaan praktisnya adalah apakah suatu peristiwa NFC sekadar membuka alur kerja lokal, atau apakah peristiwa tersebut memberikan kepercayaan yang dapat digunakan kembali kepada layanan *backend*, Bluetooth, UWB, atau layanan di dalam kendaraan.

- Pengujian GNSS dilakukan menggunakan simulator GNSS atau perangkat *spoofing* berbasis *Software-Defined Radio* (SDR) untuk menyuntikkan sinyal lokasi dan waktu yang terkendali. Penguji biasanya memulai dengan simulasi berdampak rendah di lingkungan yang terkontrol, dengan mengubah posisi, kecepatan, dan waktu secara bertahap untuk mengidentifikasi fitur kendaraan mana yang menggunakan data GNSS serta apakah fitur tersebut mampu menolak perubahan yang tidak masuk akal. Penguji penetrasi kemudian mengamati bagaimana sistem navigasi, fitur keamanan, dan layanan terkait merespons data anomali yang disuntikkan. Fokus utamanya adalah mendeteksi asumsi yang tidak aman atau korelasi kepercayaan antar-sistem.
- Penemuan *attack surface* (permukaan serangan) V2X melibatkan penangkapan dan analisis pesan siaran (*broadcast messages*) menggunakan perangkat radio V2X khusus, SDR, dan penganalisis protokol. Penguji penetrasi mengevaluasi keaslian pesan, ketahanan terhadap serangan *replay* (pengulangan pesan), serta perilaku sistem target saat menghadapi banjir pesan (*message flooding*) atau penekanan pesan secara selektif. Karena sistem V2X sering kali tidak memiliki semantik sesi tradisional akibat komunikasi dengan latensi rendah, fokus pengujian terletak pada ketahanan protokol dan deteksi perilaku yang tidak wajar.

Alat lain yang patut disebutkan adalah Wireshark, yang dapat digunakan untuk analisis setelah penangkapan data. Setelah lalu lintas nirkabel ditangkap dan dikonversi ke dalam format *Packet Capture* (PCAP) kecuali untuk teknologi seperti NFC dan GNSS yang tidak berbasis paket secara langsung Wireshark memungkinkan dekoding dan pemeriksaan komunikasi untuk menentukan protokol, profil, atau layanan jaringan apa yang sedang digunakan, serta menilai *attack surface* yang terpapar.

Untuk semua antarmuka nirkabel, fase ini berfokus pada pemahaman tentang bagaimana sistem berkomunikasi dengan mengamati lalu lintas nyata, alih-alih hanya mengandalkan arsitektur yang diasumsikan. Dengan memvalidasi paparan, batasan kepercayaan (*trust boundaries*), dan jalur integrasi, penguji penetrasi dapat membangun landasan untuk aktivitas eksploitasi yang lebih mendalam, sebagaimana dijelaskan dalam subbagian berikutnya.

Kompromi Modul Seluler dan Analisis Hak Akses

Unit telematika seluler harus dianggap dan diuji sebagai sistem komputasi yang independen. Penguji penetrasi pertama-tama mencoba mendapatkan akses logis dengan menargetkan layanan yang terbuka melalui antarmuka seluler; sering kali ditemukan portal manajemen HTTP, *daemon* SSH, atau layanan pembaruan khusus (*proprietary*). Jika akses ke layanan ini memerlukan autentikasi, dilakukan analisis untuk memeriksa apakah terdapat kredensial yang lemah atau *hardcoded* (ditanam secara permanen dalam kode), sebagaimana dijelaskan pada subbagian berikutnya. Selain itu, pengujian dengan metode *brute-force* terhadap kata sandi juga dapat dilakukan.

Langkah selanjutnya adalah ekstraksi *firmware*. Penguji penetrasi memperoleh citra *firmware* melalui server pembaruan vendor, antarmuka diagnostik, atau akses fisik. Perangkat seperti binwalk, *firmware-mod-kit*, dan strings digunakan untuk menganalisis konten sistem

berkas, mengidentifikasi layanan yang sedang berjalan, serta menemukan berkas konfigurasi. Setelah akses diperoleh, penguji akan menilai bagaimana modul seluler berkomunikasi dengan jaringan internal kendaraan untuk menyelidiki apakah modul tersebut memiliki akses langsung ke lalu lintas data melalui CAN atau Ethernet. Tujuan utamanya adalah menentukan apa saja yang dapat diakses oleh modul seluler yang telah disusupi: apakah hanya terbatas pada bidang manajemen (*management plane*) modul itu sendiri, layanan yang menghadap ke *backend*, jaringan Ethernet internal, *gateway* yang terhubung ke bus CAN, fungsi diagnostik, atau domain kendaraan dengan hak akses yang lebih tinggi.

Pergerakan Lateral (Lateral Movement) dan Penyalahgunaan Jaringan pada APN Privat

Pertama, penguji penetrasi (*penetration tester*) berupaya mendapatkan akses ke APN privat menggunakan kartu SIM yang telah disusupi atau titik akhir seluler ilegal (*rogue cellular endpoint*), seperti femtocell privat. Setelah mendapatkan akses ke APN privat, penguji dapat menilai apakah kendaraan yang terhubung dalam APN yang sama terisolasi satu sama lain, atau apakah komunikasi lateral dimungkinkan. Mereka menggunakan perangkat penemuan jaringan (*network discovery tools*) seperti Nmap untuk mendata perangkat lain dalam jaringan, mengidentifikasi infrastruktur bersama, serta memindai layanan yang terbuka di seluruh APN privat tersebut. Pengujian inti di sini adalah untuk melihat apakah APN tersebut berperilaku layaknya jaringan akses yang terisolasi, atau justru seperti jaringan privat datar (*flat private network*) di mana satu perangkat yang disusupi dapat menemukan dan berkomunikasi dengan perangkat lainnya.

Jika komunikasi lateral dimungkinkan, penguji akan mengevaluasi apakah penyusupan awal dapat digunakan untuk menjangkau kendaraan lain layanan armada (*fleet services*), atau sistem manajemen *backend*. Hal ini membantu menentukan apakah APN tersebut mampu membatasi cakupan dampak (*blast radius*), atau apakah satu perangkat yang terekspos dapat menjadi titik tumpu (*pivot point*) untuk menyusup ke lingkungan kendaraan terhubung yang lebih luas.

Injeksi perintah AT (AT Command Injection) dan Penyalahgunaan Saluran Kendali

Antarmuka perintah AT diuji dengan mengidentifikasi semua jalur yang memungkinkan dilakukannya injeksi perintah. Ini mencakup antarmuka UART, USB, Bluetooth, dan layanan jaringan lainnya, seperti pemrosesan SMS. Perangkat seperti minicom, screen, dan skrip Python kustom (menggunakan pySerial) digunakan untuk berinteraksi langsung dengan antarmuka AT guna menyuntikkan perintah. Penguji harus terlebih dahulu memastikan apakah saluran AT dapat diakses secara langsung, diakses secara tidak langsung melalui parser lain, atau hanya dapat diakses setelah mendapatkan akses lokal. Perbedaan ini penting karena antarmuka AT yang terekspos secara lokal dapat mendukung eskalasi hak akses (*privilege escalation*), sementara jalur AT yang dapat diakses dari jarak jauh dapat menjadi vektor akses awal.

Untuk pengujian berbasis SMS, penguji penetrasi menyimulasikan pesan yang dikendalikan oleh penyerang menggunakan peralatan uji seluler atau lingkungan laboratorium terkendali dengan perangkat seperti OpenBTS atau Osmocom. Mereka mengevaluasi apakah pesan yang masuk melalui proses autentikasi, pembersihan data (*sanitization*), atau justru

diteruskan begitu saja ke modem tanpa pemeriksaan. Pengujian berbasis SMS sering kali mengungkap kerentanan seperti *command chaining* (perangkaian perintah) di mana beberapa perintah dapat dieksekusi dalam satu pesan, *buffer overflow* pada pengurai (*parser*) perintah SMS, serta kesalahan logika yang menyebabkan perintah diproses dalam kondisi yang tidak diinginkan atau tidak aman. Keberhasilan eksploitasi selama pengujian ini menunjukkan bagaimana saluran kendali modem tingkat rendah dapat berubah menjadi vektor serangan jarak jauh, terutama jika pesan tidak diautentikasi dengan memadai, tidak diurai dengan benar, atau tidak diisolasi dari fungsi-fungsi yang terhubung langsung dengan kendaraan.

Menemukan Kerentanan Pada Perangkat Lunak yang Berinteraksi Dengan Nirkabel

Komponen perangkat lunak yang berinteraksi dengan nirkabel merupakan target utama karena komponen tersebut memproses input yang dikendalikan oleh penyerang baik yang diterima melalui Wi-Fi, Bluetooth, jaringan seluler, gerbang RF (*frekuensi radio*), radio V2X, maupun saluran pembaruan (*update*). Jika penguji penetrasi (*penetration tester*) memiliki akses ke kode sumber, mereka dapat menggunakan perangkat analisis statis untuk memeriksa perangkat lunak tersebut dan mengidentifikasi potensi kerentanan. Selain itu, perangkat *Software Composition Analysis* (SCA) biasanya digunakan untuk mendeteksi kerentanan yang sudah diketahui dalam komponen perangkat lunak sumber terbuka (*Open Source Software* atau OSS). Perangkat ini mampu melakukan pemindaian otomatis terhadap kode sumber atau berkas biner, lalu mencocokkannya dengan basis data kerentanan, seperti *National Vulnerability Database* (NVD).

Lebih lanjut, penguji penetrasi dapat melakukan *fuzz testing* atau *fuzzing* menggunakan perangkat seperti AFL++, libFuzzer, atau *harness* (kerangka pengujian) khusus yang dirancang untuk protokol tertutup guna menemukan kerentanan yang belum diketahui. Dalam proses ini, input yang tidak sesuai format atau tidak terduga dikirimkan ke perangkat lunak dengan menyasar *daemon* jaringan, pengurai pesan (*message parser*), serta fungsi pembaruan untuk melihat apakah hal tersebut menyebabkan perangkat lunak mengalami kerusakan (*crash*) atau berperilaku tidak wajar. Alur kerja *fuzzing* yang efektif dimulai dengan mengidentifikasi pengurai atau layanan yang menangani input nirkabel, membangun *harness* atau mekanisme *replay* untuk menyalurkan pesan ke komponen tersebut, serta memantau apakah terjadi *crash*, kerusakan memori (*memory corruption*), sistem yang membeku (*hang*), atau transisi status yang tidak aman.

Jika kerentanan tertentu ditemukan misalnya kerentanan *buffer overflow* kerentanan tersebut dapat dianalisis lebih dalam untuk mengevaluasi potensi eksploitasi serta dampaknya. Jika termasuk dalam lingkup penugasan pengujian penetrasi, penguji juga dapat mengembangkan eksploit untuk kerentanan tersebut guna menunjukkan dampak nyata dari serangan yang mungkin terjadi. Aktivitas ini menghubungkan tahap penemuan kerentanan dengan eksploitasi, dengan menunjukkan apakah input nirkabel yang tidak sesuai format dapat berkembang lebih jauh tidak sekadar menyebabkan *crash*, tetapi berujung pada eksekusi kode, peningkatan hak akses (*privilege escalation*), penolakan layanan (*denial of service*), atau menjadi pijakan yang stabil untuk tahapan berikutnya dalam rantai serangan.

Kelemahan Autentikasi dan Ekstraksi Rahasia

Pengujian mekanisme autentikasi berfokus pada identifikasi kredensial statis, rahasia yang digunakan bersama (*shared secrets*), serta asumsi kepercayaan (*trust assumptions*) yang tidak tepat. Kata sandi bawaan (*default passwords*) diuji pada layanan yang terbuka, sementara analisis *firmware* dilakukan untuk mengekstrak kredensial yang tertanam secara permanen (*hardcoded*) serta materi kriptografis. Penguji penetrasi memeriksa berkas konfigurasi, berkas biner, dan skrip inisialisasi untuk menemukan rahasia yang tertanam di dalam kode atau sistem. Alur kerja praktisnya adalah mengumpulkan kandidat rahasia dari *firmware*, berkas konfigurasi, aplikasi seluler, paket pembaruan, log, dan skrip diagnostik, kemudian menguji apakah rahasia-rahasia tersebut dapat digunakan untuk melakukan autentikasi ke layanan lokal, titik akhir (*endpoint*) cloud, atau antarmuka pada sisi kendaraan.

Perangkat *reverse engineering* seperti Ghidra, IDA Pro, dan Radare2 digunakan untuk menganalisis logika autentikasi ketika kredensial tidak disimpan dalam bentuk teks polos (*plaintext*). Dalam banyak kasus, kredensial disamarkan (*obfuscated*) dan disertakan di dalam berkas biner, alih-alih disimpan dengan aman menggunakan perangkat keras khusus; hal ini memungkinkan penguji untuk memulihkan kredensial tersebut. Penguji penetrasi (*penetration tester*) juga harus menilai apakah rahasia yang berhasil dipulihkan dapat digunakan kembali pada kendaraan lain guna memastikan apakah dampaknya mencakup seluruh armada. Aktivitas ini sering kali mengungkap apakah kegagalan autentikasi disebabkan oleh kesalahan arsitektural, seperti penggunaan kredensial yang sama untuk seluruh armada, kepercayaan berlebih terhadap modem, atau penggunaan kembali rahasia yang sama di berbagai kendaraan maupun layanan *backend*.

Eksploitasi Layanan Jaringan dan Fungsionalitas Debug

Layanan jaringan yang diekspos oleh subsistem nirkabel diuji menggunakan teknik enumerasi layanan dan eksploitasi standar. Penguji penetrasi (*penetration tester*) dapat menggunakan perangkat seperti Nmap, netcat, curl, dan Metasploit untuk mengidentifikasi layanan dengan konfigurasi yang salah serta versi perangkat lunak yang sudah usang. Antarmuka debug seperti Telnet, SSH, dan ADB dievaluasi untuk mencari celah pintasan autentikasi (*authentication bypass*) dan jalur peningkatan hak akses (*privilege escalation*). Selain itu, penguji penetrasi dapat memanfaatkan rahasia yang diperoleh dari kelemahan autentikasi sebelumnya atau aktivitas ekstraksi rahasia untuk mendapatkan akses ke layanan-layanan tersebut.

Metodenya sederhana namun penting: mengidentifikasi layanan yang aktif (*listening services*), menentukan versi perangkat lunak (*fingerprinting*), menguji autentikasi, memeriksa apakah kredensial yang ditemukan sebelumnya dapat digunakan, lalu menentukan tingkat hak akses dan jangkauan jaringan yang diperoleh setelah masuk atau melakukan eksploitasi. Penguji penetrasi harus memberikan perhatian khusus pada layanan yang terikat (*bound*) ke semua antarmuka bukan hanya ke *localhost* karena hal ini sering kali mengekspos akses tingkat root secara tidak sengaja kepada komunikasi jarak jauh yang masuk. Aktivitas ini bermanfaat karena artefak pengembangan sering kali menyediakan jalur tercepat dari akses nirkabel

menuju eksekusi perintah, terutama jika layanan debug berjalan dengan hak akses tinggi atau dapat diakses dari antarmuka yang bukan ditujukan untuk proses debug.

Serangan RF dan *Rolling Code*

Sistem berbasis RF diuji menggunakan perangkat SDR (*Software-Defined Radio*) untuk menangkap, menganalisis, dan memutar ulang (*replay*) sinyal nirkabel. Perangkat seperti GNU Radio dan Universal Radio Hacker digunakan untuk mengidentifikasi frekuensi operasi, skema modulasi, struktur paket, perilaku waktu (*timing behavior*), serta menentukan apakah protokol tersebut menggunakan kode statis, *rolling code*, mekanisme *challenge-response*, atau mekanisme khusus lainnya.

Alur kerja praktis dimulai dengan penangkapan pasif terhadap beberapa transmisi dari perangkat kendali jarak jauh (*key fob*) atau sensor yang sama. Penguji membandingkan hasil tangkapan tersebut untuk menentukan bagian mana yang tetap konstan, bagian mana yang berubah, serta apakah penghitung urutan (*sequence counter*), jendela sinkronisasi, atau pengidentifikasi tertentu dapat disimpulkan. Pengujian *replay* kemudian dilakukan dalam lingkungan terkendali untuk melihat apakah pesan yang sebelumnya ditangkap akan diterima, ditolak, atau justru menyebabkan hilangnya sinkronisasi (*desynchronization*).

Tujuannya adalah untuk menentukan apakah implementasi sistem tersebut bergantung pada rahasia statis, perilaku *rolling code* yang dapat diprediksi, logika sinkronisasi yang lemah, atau materi kriptografi yang dapat digunakan kembali. Jika diizinkan dalam lingkup pengujian, penguji juga dapat menganalisis *firmware* pada *key fob* atau penerima RF untuk menentukan apakah rahasia dapat diekstraksi atau apakah mekanisme pembaruan *firmware* dapat disalahgunakan.

Serangan Pada Tingkat Protokol Nirkabel

Serangan pada tingkat protokol nirkabel berfokus pada kelemahan dalam model komunikasi itu sendiri, bukan pada bug dalam implementasi tertentu. Dalam subbagian ini, fokus utamanya adalah komunikasi V2X (*Vehicle-to-Everything*), karena sistem V2X sangat bergantung pada pesan siaran (*broadcast*) dengan latensi rendah yang harus segera diproses oleh pihak-pihak di sekitarnya. Untuk sistem V2X, penguji penetrasi (*penetration tester*) akan menangkap lalu lintas dasar, mengidentifikasi jenis pesan, lalu mengevaluasi apakah pesan yang dipalsukan (*spoofed*), diputar ulang (*replayed*), ditunda (*delayed*), dikirim secara berlebihan (*flooded*), atau ditekan secara selektif (*selectively suppressed*) dapat memengaruhi sistem target. Proses ini melibatkan pengiriman pesan uji terkontrol menggunakan perangkat radio V2X khusus, perangkat radio yang ditentukan oleh perangkat lunak (SDR), atau peralatan simulasi, sembari mengamati apakah sistem penerima menerima pesan tersebut, menolaknya, menandainya sebagai sesuatu yang tidak wajar, atau mengubah perilaku kendaraan maupun respons yang ditujukan kepada pengemudi.

Pengujian ini menilai apakah asumsi dalam desain protokol memungkinkan penyerang untuk mengganggu aliran pesan keselamatan, menyamar sebagai infrastruktur, menyisipkan informasi yang menyesatkan, atau menurunkan ketersediaan sistem. Alih-alih mengeksploitasi bug implementasi, pengujian pada tingkat protokol mengungkap apakah sistem tetap tangguh ketika lingkungan nirkabel itu sendiri menjadi bersifat memusuhi (*adversarial*). Setelah

meninjau berbagai teknik pengujian penetrasi yang umum digunakan, bagian selanjutnya akan membahas studi kasus serangan nyata di mana teknik-teknik tersebut diterapkan untuk menyusupi sistem nirkabel pada kendaraan.

5.5 STUDI KASUS EKSPLOITASI NIRKABEL PADA SISTEM KEAMANAN KENDARAAN MODERN

Studi kasus berikut ini merupakan hasil kerja para peneliti keamanan yang membutuhkan waktu berbulan-bulan untuk mencapai temuan yang signifikan. Mengingat keterbatasan ruang, pembahasan per studi kasus tidak akan mencakup seluruh aspek serangan maupun detail lengkap mengenai bagaimana para peneliti tersebut berhasil mencapai tujuannya. Oleh karena itu, fokus utama akan diberikan pada aspek-aspek serangan yang berkaitan erat dengan komunikasi nirkabel, sembari tetap menyajikan aspek lain yang relevan dengan masing-masing studi kasus. Pembaca sangat disarankan untuk mempelajari materi referensi yang disediakan guna memperoleh pemahaman yang lebih mendalam mengenai bagaimana serangan-serangan tersebut dilakukan.

Studi kasus 1: Kloning Perangkat *Remote Keyless Entry* (RKE) Dengan Kode Bergulir (*Rolling Code*) Melalui Eksploitasi Kunci Master Global

Para peneliti keamanan menemukan kerentanan kritis dalam sistem RKE yang digunakan pada jutaan kendaraan Volkswagen Group yang diproduksi antara tahun 1995 dan 2016. Sistem ini menggunakan skema kode bergulir, di mana setiap penekanan tombol akan mengirimkan kode yang diturunkan dari rahasia kriptografis bersama. Rekayasa balik (*reverse engineering*) terhadap *firmware* unit kontrol elektronik (ECU) mengungkapkan bahwa semua kendaraan dalam lini model tertentu menggunakan sekumpulan kunci master global yang sama, alih-alih menggunakan rahasia yang unik untuk setiap kendaraan.

Akibatnya, penyerang yang berhasil menangkap satu transmisi frekuensi radio (RF) dari perangkat *remote* milik pemilik kendaraan dapat memperoleh nilai penghitung (*counter*), mengkloning *remote* tersebut, dan mendapatkan akses tidak sah ke dalam kendaraan. Para peneliti mendemonstrasikan perangkat keras lengkap dengan biaya sekitar Rp 400.000, yang terdiri dari penerima USB RTL-SDR dan modul pemancar RF berbiaya murah. Gambar 5.4 menyajikan alur langkah-langkah serangan untuk studi kasus ini:



Gambar 5.4 Sistem akses kendaraan tanpa kunci (*remote keyless entry*): Alur eksploitasi kode bergulir (*rolling code*) berbasis frekuensi radio (RF)

Penemuan dan Pengintaian

Penelitian ini dimulai dengan pemantauan pasif terhadap transmisi RF dari perangkat kendali jarak jauh (*key fob*) kendaraan grup VW. Para peneliti menggunakan berbagai perangkat *Software-Defined Radio* (SDR) termasuk HackRF, USRP, dan RTL-SDR serta penerima USB untuk merekam transmisi kode bergulir pada frekuensi yang digunakan di berbagai varian model: 433,92 MHz untuk sistem generasi awal (VW-1) dan 434,4 MHz untuk kendaraan yang diproduksi kira-kira mulai tahun 2004 dan seterusnya (VW-2 hingga VW-4). Analisis terhadap paket RF yang tertangkap menunjukkan struktur yang konsisten, yang terdiri dari *Unique Identifier* (UID), kode tombol yang ditekan, dan nilai penghitung yang terus bergulir (*rolling counter*).

Dengan menggunakan perangkat pemrograman otomotif standar, para peneliti mengekstrak *firmware* dari unit kontrol elektronik (ECU) yang diperoleh, lalu melakukan analisis statis terhadap citra *firmware* tersebut untuk mencari konstanta dan pola struktural yang berkaitan dengan algoritme enkripsi simetris yang sudah dikenal. Proses ini memungkinkan mereka untuk menemukan dan memulihkan implementasi algoritme enkripsi (sandi) khusus yang digunakan untuk mengenkripsi setiap paket kode bergulir. Algoritme tersebut diidentifikasi sebagai AUT64 yang digunakan pada varian VW-1 hingga VW-3 (mencakup kendaraan dari sekitar tahun 1995 hingga 2009) dan XTEA, yang digunakan pada varian VW-4 untuk model-model yang lebih baru. Poin krusialnya, analisis ini mengonfirmasi bahwa kunci utama kriptografis tidak dibuat secara unik untuk setiap kendaraan, melainkan digunakan secara global untuk semua kendaraan yang memakai sistem yang sama. Artinya, kunci yang diekstrak dari satu ECU dapat diterapkan pada kendaraan lain mana pun dalam lini model yang sama.

Akses Awal

Akses awal dalam studi kasus ini sepenuhnya mengandalkan penyadapan pasif terhadap transmisi frekuensi radio (RF). Karena sistem RKE menyiarkan sinyal dari *key fob* (remote kunci) secara terbuka melalui udara tanpa autentikasi, mekanisme penyandingan (*pairing*), ataupun enkripsi saluran maka setiap penerima sinyal RF yang beroperasi dalam rentang frekuensi yang relevan dapat menangkap transmisi tersebut. Para peneliti hanya perlu mencegat satu kali penekanan tombol pada *key fob* milik pemilik kendaraan untuk mendapatkan paket *rolling code* yang sah, yang berisi UID dan nilai penghitung (*counter value*).

Eksploitasi

Dengan menggunakan paket *rolling code* yang berhasil ditangkap, para peneliti mendekripsi *payload* tersebut menggunakan kunci utama (*master key*) global yang diperoleh melalui analisis *firmware*. Implementasi sandi AUT64 dan XTEA memiliki kelemahan mendasar yang sama: semua kendaraan dalam lini model tertentu menggunakan kunci utama yang identik, sehingga proses dekripsi tidak memerlukan data rahasia yang unik untuk setiap kendaraan. Dekripsi tersebut menghasilkan UID dan nilai penghitung (*counter*) terkini. Selanjutnya, para peneliti membuat *rolling code* baru yang sah dengan cara menambah nilai penghitung tersebut, lalu mengenkripsi ulang hasilnya menggunakan kunci utama yang sama. Kode-kode yang telah dibuat ini kemudian dikirimkan melalui saluran frekuensi radio (RF) menggunakan perangkat SDR, dan kendaraan target menerimanya sebagai perintah yang sah. Dengan demikian, para peneliti mampu mengunci, membuka kunci, serta memberikan perintah lain dari jarak jauh tanpa perlu memiliki akses fisik ke kendaraan tersebut.

Persistensi

Remote hasil kloning yang diperoleh dari satu sinyal yang disadap dapat berfungsi tanpa batas waktu, selama penghitung (*counter*) pada kendaraan tidak melampaui status yang diketahui oleh penyerang. Perlu dicatat bahwa para peneliti menemukan konsekuensi terkait persistensi pada sistem VW-4: jika penyerang mengirimkan setidaknya dua kode bergulir (*rolling code*) yang sah dengan nilai penghitung yang jauh lebih maju daripada nilai yang diharapkan oleh kendaraan, maka kunci remote asli akan diblokir. Akibatnya, kendaraan akan menolak setiap kode bergulir yang memiliki nilai penghitung lebih dari dua langkah di belakang nilai yang diharapkan, sehingga kunci remote milik pemilik kendaraan tidak dapat lagi digunakan.

Pemulihan kondisi ini memerlukan prosedur sinkronisasi ulang manual sebagaimana dijelaskan dalam buku panduan pemilik kendaraan, sebuah proses yang tidak mudah bagi kebanyakan pengguna. Dalam praktiknya, ini berarti serangan tersebut dapat memberikan akses nirkabel yang persisten kepada penyerang, sekaligus melumpuhkan fungsi kunci remote milik pemilik yang sah.

Dampak

Kerentanan pada sistem *Remote Keyless Entry* (RKE) ini memengaruhi kendaraan yang diproduksi selama beberapa dekade dan mencakup berbagai lini model. Para peneliti menunjukkan bahwa, secara praktis, akses tidak sah ke kendaraan dapat dilakukan dengan peralatan sederhana yang tersedia di pasaran, tanpa perlu kontak fisik dengan kendaraan, dan

tanpa interaksi apa pun dari pengguna. Pada varian VW-4, serangan ini juga menyebabkan kondisi *denial-of-service* (penolakan layanan) terhadap kunci remote yang sah.

Akar permasalahannya terletak pada keputusan desain untuk menggunakan kunci kriptografi utama (*master key*) yang sama di seluruh lini model, alih-alih menetapkan kunci rahasia yang unik untuk setiap kendaraan. Keputusan ini menghilangkan manfaat keamanan tambahan apa pun terhadap penyerang: begitu kunci global tersebut berhasil didapatkan, semua kendaraan yang menggunakan sistem yang sama akan ikut terkompromi.

Studi Kasus 2: Peretasan Modul 4G – Menyerang Gerbang Eksternal Kendaraan

Studi kasus ini mengkaji bagaimana para peneliti menargetkan modul 4G pada kendaraan yang berfungsi sebagai platform komputasi yang dapat diakses dari luar untuk mencapai tujuan mereka dalam memengaruhi jaringan internal kendaraan. Modul-modul ini biasanya tertanam dalam unit telematika untuk memfasilitasi layanan jarak jauh, seperti pemantauan status kendaraan, pembaruan perangkat lunak secara nirkabel (*over-the-air*), pembukaan kunci jarak jauh, dan layanan darurat.

Para peneliti terlebih dahulu mengumpulkan serta menganalisis modul 4G dan citra *firmware* untuk memahami perangkat keras, sistem operasi, layanan yang terbuka, kredensial, dan mekanisme pembaruannya. Selanjutnya, mereka memanfaatkan akses jaringan seluler, eksposur APN privat, autentikasi layanan yang lemah, serta jalur perintah AT (*AT command*) yang dapat diakses melalui SMS untuk menunjukkan bagaimana penyerang jarak jauh dapat mengambil alih modul telematika tersebut. Setelah modul berhasil dikompromikan, mereka menunjukkan bagaimana kelemahan dalam pembaruan *firmware*, kredensial yang tertanam secara permanen (*hardcoded*), dan isolasi jaringan yang tidak memadai dapat mengubah kompromi pada satu perangkat menjadi risiko yang lebih luas, yang berdampak pada seluruh armada kendaraan. Urutan langkah-langkah serangan ditunjukkan pada Gambar 5.5:



Gambar 5.5 Modul 4G: alur eksploitasi gerbang (*gateway*) eksternal

Penemuan dan Pengintaian

Selama tahap penemuan, para peneliti terlebih dahulu mengidentifikasi modul-modul 4G yang digunakan pada berbagai *platform* kendaraan dan memperlakukan setiap modul sebagai sistem tertanam (*embedded system*) mandiri yang layak untuk dianalisis secara menyeluruh, baik dari sisi perangkat keras maupun perangkat lunaknya. Mereka menunjukkan bahwa modul 4G dari berbagai vendor dan platform kendaraan memiliki arsitektur serta tumpukan perangkat lunak (*software stack*) yang sangat mirip.

Para peneliti mengumpulkan dan menganalisis sejumlah besar modul 4G yang diambil dari kendaraan, sistem hiburan (*infotainment*), dan unit telematika. Analisis perangkat keras menunjukkan struktur yang konsisten di antara modul-modul 4G tersebut: biasanya berupa *System-on-Chip* (SoC) berbasis ARM yang mengintegrasikan fungsi *baseband*, pemrosesan aplikasi, dan penyimpanan *flash* NAND. Dari sisi perangkat lunak, modul-modul ini menjalankan sistem operasi Linux tertanam (*embedded Linux*) dengan perangkat *userland* standar, sehingga lebih mudah dianalisis dibandingkan ponsel pintar modern.

Selanjutnya, *firmware* diperoleh melalui berbagai saluran: paket pembaruan dari OEM, perangkat *flashing* milik vendor, atau ekstraksi fisik NAND dari kendaraan yang sudah tidak terpakai maupun kendaraan uji. Setelah diekstraksi, para peneliti menggunakan *binwalk* untuk mengidentifikasi struktur sistem berkas (*filesystem*) dan skrip awal (*startup scripts*), serta menggunakan *UBI reader* untuk membongkar isi sistem berkas tersebut. Proses ini mengungkap kredensial yang tertanam secara permanen (*hardcoded*) serta kunci kriptografis yang tersimpan di dalam *firmware*. Secara 178arallel, para peneliti menganalisis jaringan operator dan APN privat yang digunakan oleh OEM otomotif. Mereka juga mengumpulkan panduan pengguna serta dokumentasi pengembang untuk menganalisis perintah AT (AT commands) yang didukung dan opsi konfigurasi yang tersedia, yang menjadi landasan bagi eksploitasi lebih lanjut.

Akses Awal

Akses awal ke modul 4G pada kendaraan tidak memerlukan kedekatan fisik dengan mobil tersebut. Sebaliknya, beberapa vektor jarak jauh dieksplorasi untuk mendapatkan akses awal. Salah satu metode melibatkan manipulasi jaringan seluler. Para peneliti menyebarkan stasiun pangkalan (*base station*) palsu menggunakan BladeRF dan YateBTS. Dengan memaksa penurunan koneksi ke 2G atau memanipulasi parameter pemilihan ulang sel (*cell reselection*), modul kendaraan di sekitar akan terhubung ke stasiun pangkalan yang dikendalikan oleh peneliti, sehingga memberikan visibilitas dan kendali penuh atas lalu lintas jaringan.

Selain itu, para peneliti menemukan layanan jaringan yang terbuka pada modul-modul yang diuji. Mereka juga mengonfirmasi bahwa pada jaringan operator tertentu yang tidak menerapkan firewall atau isolasi IP, layanan pada modul tersebut dapat diakses langsung dari luar. Para peneliti menggunakan alat pemindaian cepat *masscan* dan menemukan beberapa modul dengan *port* yang terbuka, termasuk *port* 23 untuk Telnet, *port* 5555 untuk ADB, layanan SSH dengan kata sandi kosong atau lemah, serta antarmuka manajemen berbasis web yang hanya dilindungi oleh kredensial yang lemah. Para peneliti juga mengidentifikasi

penanganan perintah AT (*AT command*) berbasis SMS sebagai titik masuk dan mengeksplorasi hal tersebut lebih lanjut selama proses eksploitasi.

Eksploitasi

Setelah terhubung, para peneliti meningkatkan hak akses (*privilese*) dan memperoleh kendali penuh atas modul 4G. Pertama, para peneliti keamanan menemukan bahwa banyak modul menggunakan kredensial bawaan (*default*) atau yang tertanam secara permanen (*hardcoded*) untuk layanan manajemen jarak jauh. Sebagai contoh, kata sandi bawaan `oelinux123` ditemukan pada beberapa modul, yang memungkinkan akses *shell* secara langsung tanpa memerlukan eksploitasi lebih lanjut.

Pada salah satu unit telematika, para peneliti mengidentifikasi sebuah proses yang berjalan pada *port internal* yang seharusnya ditujukan untuk komunikasi antarproses namun justru terikat pada semua antarmuka (bukan hanya *localhost*), sehingga dapat diakses dari luar. Proses ini menggunakan protokol berbasis *Public Key Infrastructure* (PKI) yang menggabungkan verifikasi sertifikat RSA dengan enkripsi AES. Melalui analisis *firmware*, para peneliti menemukan tiga kelemahan dalam implementasi ini: kunci AES tertanam langsung di dalam *binary*, kunci privat RSA disimpan dalam sistem berkas (*filesystem*), dan kata sandi yang melindungi kunci privat tersebut juga tertanam di dalam *binary*.

Dengan memulihkan kunci privat RSA dan menurunkan kunci publik yang bersesuaian, para peneliti berhasil melewati autentikasi TLS pada port tersebut. Setelah autentikasi terlewati, mereka mengirimkan perintah melalui port tersebut untuk mengaktifkan layanan Telnet tersembunyi pada unit telematika. Untuk masuk ke layanan Telnet, diperlukan kata sandi *root*. Para peneliti mengekstrak berkas `/etc/passwd` dari *firmware* dan menjalankan perangkat lunak peretas kata sandi, *hashcat*, dengan memanfaatkan GPU berperforma tinggi untuk memecahkan kata sandi tersebut dalam waktu satu hari. Kata sandi tersebut cukup kompleks, mencakup huruf besar dan kecil, angka, serta karakter khusus.

Setelah mendapatkan akses *shell* dengan hak akses *root*, para peneliti mampu menyuntikkan pesan terkait CAN dari unit telematika ke unit kendali elektronik (*Electronic Control Unit* atau ECU) di dalam kendaraan, yang menunjukkan kemampuan untuk mengendalikan fungsi kendaraan seperti penguncian pintu dan penyalaan mesin. Selain itu, para peneliti menunjukkan bahwa perintah AT (*AT commands*) dapat dikirimkan ke modul 4G melalui SMS. Pengiriman perintah AT via SMS memerlukan kata sandi; para peneliti memperoleh kata sandi ini melalui rekayasa balik (*reverse-engineering*) terhadap *firmware*, di mana kata sandi tersebut ditemukan tertanam secara permanen. Lebih lanjut, mereka menunjukkan bahwa penangan perintah AT tertentu yang dipicu oleh SMS tidak memiliki mekanisme pembersihan input (*input sanitization*) yang memadai, sehingga memungkinkan penyisipan perintah tingkat sistem ke dalam instruksi modem yang sah. Saat diproses oleh *firmware*, perintah-perintah yang disisipkan ini dijalankan dengan hak akses root, yang pada akhirnya menghasilkan eksekusi kode jarak jauh.

Persistensi

Persistensi dicapai dengan mengompromikan mekanisme pembaruan dan *booting* pada modul 4G, sehingga kendali jangka panjang tetap dapat dipertahankan bahkan setelah

perangkat dimulai ulang (*reboot*) atau diatur ulang (*reset*). Para peneliti menemukan kredensial FTP yang tertanam secara permanen (*hardcoded*) di dalam *firmware* modul, lalu menggunakannya untuk mengakses server *Firmware Over-the-Air* (FOTA) milik produsen secara langsung. Selanjutnya, para peneliti menunjukkan bahwa mereka dapat mengganti *firmware* resmi dengan versi yang telah disisipi *backdoor*, mengingat akun FTP tersebut juga memiliki hak akses tulis (*write access*).

Selain itu, pembaruan *firmware* tidak dilindungi oleh verifikasi tanda tangan digital; artinya, perangkat apa pun dari merek tersebut akan memasang *firmware* yang telah disisipi *backdoor* tanpa sepengetahuan pengguna. Mereka membuktikan bahwa akses persisten dapat diperoleh pada semua modul dari merek tertentu. Pada tingkat perangkat, para peneliti juga menunjukkan bahwa konten memori *flash* NAND serta skrip inisialisasi seperti `init.rc` dapat dimodifikasi secara langsung untuk menanamkan *backdoor* yang bersifat persisten.

Dampak

Para peneliti menunjukkan bahwa modul 4G yang telah dikompromikan dapat digunakan untuk mengendalikan fungsi kendaraan dari jarak jauh. Sebagai contoh, mereka memperlihatkan bahwa perintah yang berkaitan dengan bus CAN (*Controller Area Network*) dapat dikirim dari modul yang telah dikompromikan menuju *Electronic Control Unit* (ECU) di dalam kendaraan untuk mengendalikan berbagai fungsi, termasuk penguncian pintu dan penyalaan mesin. Dalam skala yang lebih luas, para peneliti menunjukkan bahwa *Private APN* (*Access Point Name*) yang tidak diisolasi dengan baik memungkinkan akses ke sejumlah besar kendaraan dalam jaringan yang sama secara bersamaan. Dikombinasikan dengan temuan terkait FOTA, mereka menunjukkan bahwa satu server FOTA yang telah dikompromikan dapat digunakan untuk menyebarkan *firmware* yang berisi backdoor ke seluruh modul dari merek tertentu dalam waktu kurang dari satu hari.

Studi Kasus 3: Eksekusi Kode Jarak Jauh Tanpa Interaksi Pengguna (*Zero-Click*) Pada Sistem Infotainmen

Studi kasus ini berfokus pada teknik serangan yang canggih serta metode penyampaian untuk melakukan eksekusi kode jarak jauh pada sistem infotainmen. Serangan terhadap sistem infotainmen Tesla ini yang dikenal luas dengan sebutan TBone merupakan salah satu contoh paling representatif dari eksploitasi nirkabel pada kendaraan modern. Serangan ini berupa eksploitasi eksekusi kode jarak jauh tanpa interaksi pengguna (*zero-click*) yang menargetkan sistem infotainmen berbasis Linux pada kendaraan Tesla melalui jaringan Wi-Fi. Dengan merangkaikan beberapa kerentanan kerusakan memori (*memory corruption*) pada ConnMan (pengelola koneksi jaringan), para peneliti berhasil mendapatkan kendali penuh atas domain infotainmen tanpa perlu memiliki akses fisik ke kendaraan tersebut.

Hal yang membuat demonstrasi ini sangat menarik bukan hanya kecanggihan teknis dari rangkaian eksploitasi tersebut, melainkan juga metode penyampaiannya. Para peneliti menggunakan drone yang dilengkapi dengan perangkat Wi-Fi (*dongle*) dan antena untuk melancarkan serangan dari udara, sehingga berhasil menyusupi kendaraan dari jarak sekitar 100 meter. Meskipun eksploitasi ini tidak memberikan kendali langsung atas fungsi-fungsi berkendara yang krusial bagi keselamatan, kejadian ini menunjukkan dengan jelas bagaimana

kelemahan pada sistem periferal tetap dapat berujung pada konsekuensi keamanan yang serius. Gambar 5.6 menyajikan alur langkah-langkah serangan untuk studi kasus ini:



Gambar 5.5 Tesla TBone: Alur eksekusi kode jarak jauh dari Wi-Fi ke sistem infotainment

Penemuan dan Pengintaian

Serangan ini dimulai dengan studi mendalam terhadap ekosistem elektronik dan perangkat lunak Tesla, dengan perhatian khusus pada sistem infotainment dan modul *Vehicle Control Security* (VCSEC). Kedua komponen ini dipilih karena secara bersama-sama membentuk batasan krusial antara antarmuka yang terpapar ke luar dengan fungsi-fungsi internal kendaraan. Sistem infotainment merupakan salah satu domain yang paling kompleks dan dapat diakses melalui jaringan di dalam kendaraan, sementara VCSEC berfungsi sebagai gerbang keamanan yang bertanggung jawab untuk mengautentikasi serta mengotorisasi berbagai tindakan terkait bodi kendaraan, termasuk penguncian pintu dan fitur-fitur lain yang memerlukan kontrol akses.

Fokus utama dari pengintaian ini adalah memahami bagaimana sistem infotainment berkomunikasi dengan VCSEC. Meskipun unit infotainment tidak dapat mengendalikan fungsi-fungsi berkendara yang krusial bagi keselamatan secara langsung, unit ini mengirimkan permintaan terautentikasi ke VCSEC untuk melakukan tindakan seperti akses pintu dan fitur kenyamanan. Hal ini menjadikan sistem infotainment sebagai target yang menarik, karena dengan menguasainya, penyerang dapat mengeluarkan perintah yang sah kepada *Electronic Control Unit* (ECU) di dalam kendaraan melalui VCSEC.

Untuk memfasilitasi analisis mendalam, para peneliti memperoleh perangkat keras Tesla dari kendaraan yang telah rusak (*salvaged*), yang memungkinkan mereka untuk mempelajari kedua sistem tersebut di luar batasan kendaraan yang sebenarnya. Para peneliti menggunakan JTAG untuk mengambil (*dump*) *firmware* dari komponen-komponen yang terkait dengan VCSEC, kemudian melakukan analisis statis secara ekstensif terhadap *firmware* tersebut guna memahami hubungan kepercayaan (*trust relationships*), alur pesan, dan

batasan hak akses (*privilege boundaries*). *Firmware* tersebut berhasil diemulasi menggunakan KunnaEmu, sebuah emulator khusus yang dikembangkan oleh para peneliti. Selanjutnya, para peneliti melakukan *fuzzing* terhadap *firmware* yang berjalan di dalam emulator tersebut.

Secara paralel, para peneliti juga menganalisis sistem *infotainment* yang merupakan sistem Linux standar untuk mengidentifikasi potensi vektor serangan dan memahami mekanisme perlindungan yang ada. Sistem *infotainment* ini diemulasi menggunakan QEMU yang dijalankan pada perangkat Intel NUC. Melalui proses ini, para peneliti memetakan *attack surface* (permukaan serangan) sistem tersebut dan mengidentifikasi ConnMan sebagai target yang sangat menarik. ConnMan merupakan aplikasi pihak ketiga yang bertanggung jawab untuk mengelola konektivitas jaringan termasuk Wi-Fi, *Dynamic Host Configuration Protocol* (DHCP), dan *Domain Name System* (DNS) serta ditulis menggunakan bahasa pemrograman C yang tidak aman terhadap memori (*memory-unsafe*). Yang terpenting, aplikasi ini dapat diakses melalui Wi-Fi dan menangani data jaringan kompleks yang bisa dikendalikan oleh penyerang. Para peneliti kemudian melakukan *fuzzing* terhadap ConnMan menggunakan AFL.

Selama tahap pengintaian, tim peneliti keamanan juga menemukan kelemahan konfigurasi yang kritis: kendaraan Tesla di Eropa diprogram untuk terhubung secara otomatis ke jaringan Wi-Fi tertentu bernama "*Tesla Service*", yang ditujukan untuk keperluan diagnostik dan servis. Kunci prabagikan (*pre-shared key*) untuk SSID ini tertanam langsung di dalam *firmware*, dan penyelidikan lebih lanjut mengungkap bahwa kunci tersebut telah tersebar secara publik di internet. Kombinasi antara kepercayaan otomatis dan kredensial yang terekspos ini membuka jalan bagi akses jarak jauh yang dapat diandalkan.

Pada akhir fase pengintaian, para peneliti tidak hanya mengidentifikasi layanan yang rentan dalam ConnMan, tetapi juga menemukan jalur serangan yang jelas mulai dari akses nirkabel hingga kompromi pada sistem *infotainment* dengan potensi dampak lanjutan terhadap fungsi-fungsi kendaraan yang dimediasi oleh VCSEC. Perpaduan antara kepercayaan nirkabel yang terekspos, perangkat lunak yang tidak aman secara memori (*memory-unsafe*), serta interaksi lintas domain, menjadi landasan bagi eksploitasi jarak jauh penuh yang diperagakan pada tahap-tahap serangan berikutnya.

Akses Awal

Akses awal tidak bergantung pada eksploitasi kerentanan secara terpisah, melainkan pada penyalahgunaan perilaku jaringan normal kendaraan. Dengan mengonfigurasi titik akses (*access point*) Wi-Fi berbahaya yang menyiarkan SSID "*Tesla Service*" dan menggunakan kunci prabagikan (*pre-shared key*) yang sudah diketahui, para peneliti dapat membuat kendaraan Tesla di dekatnya terhubung ke jaringan mereka. Dalam demonstrasi TBone, titik akses palsu ini dipasang pada drone yang dilengkapi dengan perangkat Wi-Fi (*dongle*) dan antena, sehingga memungkinkan para peneliti untuk mendekati kendaraan secara diam-diam dan memulai koneksi dari udara. Begitu mobil terhubung ke jaringan palsu tersebut, para peneliti dapat berkomunikasi langsung dengan sistem infotainment.

Penting untuk dicatat bahwa pijakan awal ini tidak serta-merta memberikan kendali atas fungsi-fungsi kendaraan atau akses langsung ke bus internal kendaraan. Namun, hal ini memungkinkan para peneliti untuk memasuki zona kepercayaan (*trust zone*) sistem

infotainment, tempat mereka dapat berinteraksi dengan layanan sistem yang secara sah berkomunikasi dengan komponen kendaraan lainnya, termasuk VCSEC. Dari perspektif arsitektur, ini merupakan langkah krusial: sistem infotainment berfungsi sebagai jembatan antara antarmuka eksternal dan mekanisme otorisasi internal, sehingga kompromi pada sistem ini menjadi sangat berbahaya. Setelah terhubung melalui Wi-Fi, para peneliti menggunakan komunikasi jaringan nirkabel awal tersebut untuk mengidentifikasi versi *firmware*. Langkah ini sangat penting, karena pengetahuan yang tepat mengenai versi *firmware* memungkinkan mereka menghitung offset memori secara akurat sebuah prasyarat untuk melewati mitigasi eksploitasi modern dalam rangkaian serangan tersebut.

Eksploitasi

Tahap eksploitasi dibangun berdasarkan rangkaian kerentanan multi-tahap yang dirancang dengan cermat untuk melewati mitigasi eksploitasi Linux modern, seperti *Address Space Layout Randomization* (ASLR) dan *stack canaries*. Akibat adanya pertahanan ini, teknik *buffer overflow* sederhana tidak cukup untuk menjalankan kode arbitrer. Sebagai gantinya, para peneliti menggunakan *Return-Oriented Programming* (ROP) sebuah teknik yang merangkai urutan instruksi kecil yang sudah ada di dalam memori untuk melakukan tindakan berbahaya tanpa perlu menyuntikkan kode baru. Para peneliti menyusun rangkaian serangan multi-tahap yang kompleks untuk mengeksploitasi kelemahan dalam ConnMan guna mencapai eksekusi kode jarak jauh (*remote code execution*) yang andal.

Istilah:

- *Address Space Layout Randomization* (ASLR) adalah teknik perlindungan memori yang mengacak lokasi kode, stack, heap, dan pustaka (*library*) dalam ruang alamat suatu proses, sehingga menyulitkan penyerang untuk memprediksi alamat memori selama proses eksploitasi.
- *Stack canary* adalah nilai pelindung khusus yang ditempatkan pada stack sebelum alamat kembali (*return address*) untuk mendeteksi buffer overflow berbasis stack; jika nilai tersebut berubah, program dapat mendeteksi adanya kerusakan dan menghentikan eksekusi.
- *Return-Oriented Programming* (ROP) adalah teknik eksploitasi yang menggunakan kembali rangkaian instruksi pendek yang sudah ada di dalam memori program untuk menjalankan logika berbahaya; teknik ini sering digunakan untuk melewati mekanisme perlindungan modern seperti ASLR dan *stack canary*.

Tahap pertama memanfaatkan kebocoran informasi *stack* dalam implementasi klien DHCP pada ConnMan (CVE-2021-26676). Dengan menyusun respons DHCP berbahaya, para peneliti berhasil membocorkan informasi memori sensitif, termasuk alamat *stack* dan alamat dasar pustaka bersama (*shared libraries*) seperti *libc*. Informasi ini digunakan untuk menembus mekanisme ASLR dan memberikan peta yang tepat mengenai tata letak memori proses, yang diperlukan untuk tahap serangan berikutnya.

Berbekal pengetahuan tersebut, para peneliti memicu kerentanan kedua: *stack overflow* dalam logika penguraian (*parsing*) respons DNS pada ConnMan, khususnya di dalam fungsi yang menangani respons DNS tanpa kompresi (CVE-2021-26675). Para peneliti menggunakan *scapy* untuk menyusun dan mengirimkan paket DNS serta DHCP berbahaya.

Karena telah mengetahui tata letak *stack* secara akurat, mereka dapat melompati *stack canary* dan melakukan penulisan parsial yang terkendali untuk mengalihkan alur eksekusi ke dalam *ROP chain* (rangkaian instruksi *Return-Oriented Programming*) yang telah disusun dengan cermat. Hal ini memungkinkan eksekusi kode yang andal di dalam proses sistem *infotainment*, yang dilakukan sepenuhnya melalui Wi-Fi tanpa memerlukan interaksi pengguna sebuah metode yang dikenal sebagai *zero-click remote code execution*.

Setelah eksekusi kode berhasil dilakukan, eksploitasi tersebut memicu koneksi TCP balik (*reverse TCP connection*) ke server yang dikendalikan oleh peneliti, yang kemudian mengirimkan *payload* tahap kedua ke dalam memori. Pada titik ini, para peneliti telah menguasai sistem *infotainment* dan dapat berinteraksi dengan layanan kendaraan yang berada pada tingkat lebih tinggi.

Persistensi

Meskipun para peneliti tidak menjabarkan mekanisme persistensi apa pun yang dapat bertahan setelah perangkat dimulai ulang (*reboot*) atau pembaruan *firmware*, perlu dicatat bahwa satu detail operasional sangat membantu proses eksploitasi: ConnMan secara otomatis dimulai ulang setiap kali mengalami kerusakan (*crash*). Perilaku ini memungkinkan para peneliti untuk berulang kali mencoba memicu kerentanan tersebut hingga eksploitasi berhasil, sehingga serangan menjadi sangat andal terlepas dari kompleksitas rantai eksploitasi yang digunakan. Setelah payload tahap kedua berjalan, para peneliti mempertahankan kendali melalui saluran komando-dan-kendali (*command-and-control*) yang aktif selama sesi berlangsung. Walaupun persistensi ini bersifat sementara, hal tersebut sudah cukup untuk menunjukkan secara menyeluruh dampak dari kompromi keamanan yang terjadi.

Dampak

Meskipun sistem infotainmen dipisahkan dari sistem penggerak yang krusial bagi keselamatan melalui gerbang internal (*internal gateway*), konsekuensi dari keberhasilan peretasan tetaplah signifikan. Para peneliti menunjukkan bahwa mereka mampu mematikan firewall, membuka port pengisian daya, serta mengoperasikan pintu model *falcon wing* pada kendaraan dari jarak jauh. Para peneliti juga menyatakan bahwa tingkat akses ini memungkinkan mereka untuk mengendalikan berbagai fungsi kendaraan lainnya, seperti menyesuaikan posisi kursi, mengatur sistem kendali iklim, mengubah volume musik, dan fungsi-fungsi lain yang dapat diakses melalui konsol infotainmen.

Banyak dari tindakan tersebut dilakukan melalui VCSEC, yang berfungsi menegakkan otorisasi untuk fungsi-fungsi yang berkaitan dengan bodi kendaraan. Penting untuk dicatat bahwa serangan ini tidak melangkahi model keamanan VCSEC; sebaliknya, serangan tersebut memanfaatkan fakta bahwa sistem infotainmen merupakan klien tepercaya bagi VCSEC. Begitu sistem infotainmen berhasil diretas, sistem tersebut dapat mengirimkan permintaan sah yang memang dirancang untuk diterima oleh VCSEC. Hal ini menunjukkan bahwa keamanan modul otorisasi pada dasarnya sangat bergantung pada integritas klien yang berada di tingkat sebelumnya (*upstream clients*).

Selain kendali fungsional, serangan ini menimbulkan kekhawatiran serius terkait privasi. Akses ke domain sistem hiburan (*infotainment*) dapat mengekspos riwayat lokasi, akun

pengguna, perangkat seluler yang terhubung, serta kredensial atau token yang mungkin digunakan oleh layanan kendaraan lainnya. Demonstrasi ini juga menyoroti risiko arsitektural yang lebih luas di seluruh industri otomotif: meskipun sistem yang krusial bagi keselamatan telah diisolasi, kerentanan pada komponen pihak ketiga yang terpapar koneksi nirkabel seperti ConnMan tetap dapat berdampak nyata terhadap keamanan, privasi, dan keselamatan.

Pada akhirnya, serangan TBone menjadi pengingat kuat bahwa dalam kendaraan yang terhubung (*connected vehicles*), tidak ada sistem yang benar-benar bisa dianggap "tidak krusial." Perangkat periferal yang menghadap ke luar, seperti sistem hiburan, serta komponen perangkat lunak pihak ketiga seperti ConnMan (pengelola koneksi jaringan), tidak dapat dianggap sebagai risiko rendah. Terlebih lagi, ketika sistem-sistem ini dipercaya oleh modul yang krusial bagi keamanan seperti VCSEC maka kompromi pada sistem tersebut dapat menjadi batu loncatan yang mengubah kerentanan nirkabel jarak jauh menjadi kendali serius atas perilaku kendaraan. Setelah membahas studi kasus nyata tersebut, kini kita akan meninjau praktik-praktik terbaik yang dapat diadopsi oleh organisasi otomotif untuk mengurangi risiko munculnya kerentanan dalam sistem nirkabel kendaraan mereka.

5.6 PRAKTIK TERBAIK DALAM PENGAMANAN SISTEM NIRKABEL KENDARAAN

Mengamankan sistem nirkabel kendaraan memerlukan lebih dari sekadar menangani kerentanan individual saat ditemukan. Kompleksitas dan masa pakai kendaraan modern menuntut pendekatan berbasis prinsip yang tetap efektif seiring dengan berkembangnya teknologi, taktik penyerang, dan pola penggunaan. Bagian ini menyajikan praktik-praktik terbaik untuk mengamankan sistem nirkabel kendaraan dengan mempertimbangkan kelemahan umum serta contoh-contoh dari studi kasus yang telah dibahas. Alih-alih membahas perbaikan yang bersifat parsial, kami berfokus pada praktik arsitektural dan organisasional yang secara sistematis mengurangi kelemahan-kelemahan berulang yang telah disebutkan sebelumnya seperti perluasan permukaan serangan, isolasi yang lemah, layanan yang terekspos, jalur kendali yang tidak aman, kredensial yang tertanam secara permanen (*hardcoded*), keterbatasan protokol, serta perangkat lunak yang rentan pada antarmuka nirkabel. Tujuannya adalah mengalihkan perspektif pembaca dari mitigasi kerentanan yang bersifat reaktif menuju rekayasa keamanan yang proaktif. Jika diterapkan secara konsisten, prinsip-prinsip ini akan membuat platform kendaraan tetap tangguh dalam lingkungan nirkabel yang penuh ancaman sepanjang masa operasionalnya.

Memperlakukan Semua Antarmuka Nirkabel Sebagai Titik Masuk yang Tidak Tepercaya

Setiap antarmuka nirkabel pada kendaraan baik itu teknologi berbasis IP dengan *bandwidth* tinggi seperti Wi-Fi dan seluler, komunikasi jarak pendek seperti Bluetooth dan NFC, sistem RF tingkat rendah seperti RKE dan UWB, maupun teknologi siaran seperti GNSS dan V2X harus diperlakukan sebagai sumber input yang tidak tepercaya. Perlu dicatat bahwa kedekatan fisik, jangkauan terbatas, atau penggunaan perangkat keras khusus tidak boleh dianggap sebagai jaminan keamanan yang memadai.

Dalam praktiknya, ini berarti setiap antarmuka nirkabel harus memiliki pemilik yang jelas, model ancaman, inventaris layanan yang terbuka, serta batasan kepercayaan (*trust*

boundary) yang terdokumentasi. Antarmuka yang hanya diaktifkan untuk keperluan manufaktur, diagnostik, atau operasional servis tidak boleh tetap dapat diakses dalam mode operasional normal. Prinsip ini sangat penting bagi antarmuka yang secara tradisional dianggap berisiko rendah, seperti NFC atau sistem kunci berbasis frekuensi di bawah 1 GHz (sub-GHz).

Antarmuka semacam ini sering kali berfungsi sebagai landasan kepercayaan (*trust anchor*) untuk operasi dengan hak akses lebih tinggi, termasuk akses kendaraan, penyandingan (*pairing*), dan penyediaan kunci digital. Begitu kepercayaan diberikan melampaui pertukaran sinyal radio awal, kelemahan dapat merambat ke dalam jaringan internal kendaraan maupun sistem *backend*. Oleh karena itu, arsitektur keamanan harus mengasumsikan bahwa setiap sinyal nirkabel dapat diamati, diputar ulang (*replay*), dimodifikasi, ditunda, diganggu (*jamming*), atau ditiru oleh penyerang yang memiliki motivasi dan peralatan yang memadai.

Menegakkan Isolasi yang Kuat Antara Domain Nirkabel dan Sistem Kendaraan

Antarmuka nirkabel tidak boleh memberikan akses implisit ke jaringan internal kendaraan atau domain kendali. Setiap teknologi nirkabel harus diisolasi menggunakan batasan kepercayaan yang tegas, gerbang protokol (*protocol gateway*), dan lapisan validasi pesan. Hal ini berlaku baik untuk antarmuka berbasis IP seperti Wi-Fi dan jaringan seluler maupun sistem non-IP seperti RKE, UWB, dan V2X. Selain itu, isolasi jaringan yang tepat termasuk segmentasi ke dalam zona aman dan penggunaan firewall sangat krusial untuk membatasi paparan risiko, terutama bagi perangkat yang dapat diakses melalui jaringan publik atau APN privat yang dikonfigurasi dengan buruk. Isolasi ini harus ditegakkan secara teknis, bukan sekadar diasumsikan dalam rancangan arsitektur. Sebagai contoh, jika titik akses Wi-Fi, layanan Bluetooth, modem seluler, atau penerima V2X mengalami kompromi, hal tersebut tidak boleh secara otomatis memberikan akses ke jaringan Ethernet internal, bus CAN, layanan diagnostik, atau domain kendali yang berkaitan dengan keselamatan.

Isolasi menjadi sangat krusial ketika berbagai teknologi nirkabel saling berinteraksi. Sebagai contoh, hasil pengukuran jarak menggunakan UWB dapat digabungkan dengan logika RKE, peristiwa NFC dapat memicu penyediaan kunci digital, dan data GNSS dapat memengaruhi navigasi atau keputusan keamanan yang bergantung pada waktu. Dalam setiap kasus, data yang berpindah antar-domain harus divalidasi, dibatasi, dan ditempatkan dalam konteks yang tepat, alih-alih dipercaya secara implisit. Dengan demikian, subsistem nirkabel tidak boleh dipercaya hanya karena subsistem tersebut tertanam di dalam kendaraan. Desain sistem harus menerapkan isolasi yang kuat antara domain nirkabel dan jaringan internal kendaraan, di mana setiap pesan yang melintasi domain harus mendapatkan otorisasi dan validasi secara eksplisit.

Merancang Autentikasi dan Otorisasi Untuk Lingkungan yang Rentan Terhadap Serangan

Mekanisme autentikasi pada seluruh antarmuka nirkabel harus dirancang dengan mempertimbangkan keberadaan penyerang aktif. Penggunaan kredensial statis, rahasia bersama (*shared secrets*), dan hubungan kepercayaan implisit pada dasarnya menimbulkan risiko signifikan yang berpotensi berdampak pada seluruh kendaraan dalam satu armada. Hal ini berlaku tidak hanya untuk layanan berbasis IP, tetapi juga untuk protokol frekuensi radio

(RF), sistem kunci digital, dan skema autentikasi siaran (*broadcast*). Untuk mengatasi kelemahan yang berkaitan dengan rahasia yang tertanam secara permanen (*hardcoded*) serta penggunaan kembali kredensial yang sama untuk seluruh armada, sistem autentikasi harus mengandalkan identitas yang unik untuk setiap kendaraan dan setiap komponen, alih-alih menggunakan kata sandi bersama atau kunci global.

Identitas per kendaraan, penyimpanan kunci yang didukung perangkat keras, dan mekanisme tantangan-respons yang kuat secara kriptografis sangat penting. Untuk sistem seperti RKE dan UWB, desain kriptografis harus tahan terhadap serangan pemutaran ulang, relai, dan penurunan versi. Untuk sistem V2X dan yang bergantung pada GNSS, mekanisme otentikasi dan integritas harus memperhitungkan ketiadaan kepercayaan titik-ke-titik yang andal. Otorisasi harus sama ketatnya. Otorisasi menentukan tindakan yang diizinkan dan sumber daya yang dapat diakses setelah otentikasi. Otentikasi yang berhasil hanya boleh memberikan hak istimewa minimum yang diperlukan, dengan kepercayaan yang dibatasi oleh waktu, konteks, dan kondisi operasional. Kredensial juga harus dikelola sebagai aset siklus hidup. Ini termasuk penyediaan yang aman, penyimpanan yang didukung perangkat keras, rotasi, pencabutan, dan prosedur pemulihan yang jelas ketika kunci atau sertifikat diduga telah dikompromikan.

Melindungi Kredensial, Kunci, dan Alur Kerja Penyediaan (*Provisioning*)

Kredensial dan kunci kriptografis yang digunakan oleh sistem kendaraan nirkabel tidak boleh dibagikan secara luas di antara berbagai kendaraan, layanan, maupun pemasok. Kata sandi bawaan, rahasia yang tertanam dalam kode (*hardcoded*), dan kunci yang digunakan kembali secara global dapat menyebabkan satu insiden kompromi berdampak pada seluruh armada kendaraan.

Kunci harus dibuat secara unik, disimpan dalam media penyimpanan berbasis perangkat keras (*hardware-backed storage*) jika memungkinkan, dan dibatasi hanya untuk fungsi-fungsi yang benar-benar diperlukan. Alur kerja penyediaan untuk kunci digital, kredensial seluler, sertifikat *backend*, dan kredensial pembaruan harus melalui proses autentikasi, dapat diaudit, serta dirancang untuk mendukung mekanisme pencabutan (*revocation*). Praktik ini secara langsung mengurangi dampak dari ekstraksi rahasia, analisis *firmware*, kebocoran dokumentasi, dan penyalahgunaan kredensial *backend* dengan membatasi sejauh mana kredensial yang berhasil diperoleh pihak lain dapat digunakan kembali.

Menghilangkan Layanan yang Tidak Diperlukan, Jalur Kendali Warisan (*Legacy*), dan Mekanisme Cadangan (*Fallback*) yang Tidak Aman

Banyak kerentanan nirkabel muncul bukan semata-mata karena protokol modern, melainkan karena mekanisme warisan yang dipertahankan demi kompatibilitas mundur, kebutuhan manufaktur, diagnostik, atau kemudahan layanan. Sebagai contoh, antarmuka perintah AT (*AT command*), jalur kendali berbasis SMS, mode pemasangan (*pairing*) yang tidak aman, serta perilaku penurunan versi protokol (*protocol downgrade*) dapat menciptakan kelemahan dalam sistem yang sebenarnya dirancang dengan kokoh. Selain itu, layanan warisan atau layanan pemeliharaan yang tidak lagi diperlukan seperti Telnet, SSH, atau ADB sebaiknya

dihapus atau dinonaktifkan pada sistem produksi guna memperkecil permukaan serangan. Layanan-layanan ini sering kali aktif secara bawaan dan dapat memberikan titik masuk yang mudah bagi penyerang untuk mengakses sistem nirkabel kendaraan. Oleh karena itu, sistem dalam tahap produksi harus diperiksa untuk memastikan tidak ada antarmuka administrasi (seperti Telnet, SSH, atau HTTP), ADB, titik akhir teknis (*engineering endpoints*), kredensial pengujian, maupun layanan diagnostik yang terbuka dan terhubung ke antarmuka eksternal.

Perlu dicatat bahwa mekanisme cadangan (*fallback*) patut mendapatkan perhatian khusus. Sistem UWB yang secara diam-diam beralih kembali ke perilaku RKE (*Remote Keyless Entry*) lama, tumpukan protokol Bluetooth yang menerima akses profil tanpa autentikasi, atau sistem Wi-Fi yang mengekspos layanan diagnostik saat terjadi kesalahan semuanya menciptakan kondisi yang dapat dieksploitasi. Jika memungkinkan, jalur cadangan harus diamankan dengan tingkat keamanan yang setara dengan mekanisme utama; jika tidak, jalur tersebut sebaiknya tidak disertakan. Perilaku cadangan juga harus dapat diamati dan diaudit. Sistem harus dalam kondisi terkunci (*fail-closed*) ketika jalur nirkabel yang aman tidak tersedia, terutama jika jalur cadangan tersebut memengaruhi akses kendaraan, fungsi kunci digital, diagnostik, atau pembaruan sistem. Penting untuk diingat bahwa layanan yang tidak diperlukan serta kompatibilitas dengan sistem lama (*backward compatibility*) tidak boleh mengesampingkan persyaratan keamanan, khususnya pada sistem yang mengendalikan akses kendaraan atau fungsi yang berkaitan dengan keselamatan.

Mengantisipasi Kompromi Parsial dan Merancang Sistem Dengan Prinsip Pembatasan (*Containment*)

Arsitektur keamanan nirkabel harus mengasumsikan bahwa beberapa komponen pada akhirnya akan mengalami kompromi. Oleh karena itu, dengan menerapkan pendekatan pertahanan berlapis (*defense-in-depth*) serta melakukan segmentasi, pemisahan hak akses (*privilege separation*), dan penggunaan antarmuka komunikasi yang terbatas, sebuah serangan yang berhasil terhadap tumpukan protokol Wi-Fi, layanan Bluetooth, modem seluler, penerima RF, atau komponen V2X tidak seharusnya memberikan akses tak terbatas ke sistem lain yang terhubung ataupun ke fungsi-fungsi kendaraan yang krusial bagi keselamatan.

Penerapan pembatasan semacam ini memerlukan isolasi proses yang kuat, pemisahan hak akses, dan segmentasi jaringan. Lebih lanjut, subsistem nirkabel harus beroperasi dengan hak akses seminimal mungkin dan berkomunikasi dengan sistem kendaraan melalui antarmuka yang terbatas dan terdefinisi dengan jelas. Prinsip ini sangat penting untuk membatasi dampak serangan terhadap sistem yang berinteraksi dengan pihak luar. Pembatasan juga harus diterapkan di dalam subsistem nirkabel itu sendiri. *Daemon* jaringan, pengurai protokol (*protocol parsers*), klien pembaruan, dan penangan pesan (*message handlers*) tidak boleh dijalankan dengan hak akses yang tidak diperlukan; hal ini bertujuan untuk mencegah situasi di mana kompromi pada satu proses memberikan kendali penuh yang tidak terbatas atas keseluruhan modul. Dengan mempertimbangkan kemungkinan terjadinya serangan yang berhasil serta merancang sistem yang mampu membatasi dampak serangan tersebut, alih-alih sekadar berupaya mencegah serangan secara sempurna sistem dapat tetap tangguh (*resilient*) bahkan ketika pertahanan individual mengalami kegagalan.

Melakukan Validasi Semantik Terhadap Semua Input Nirkabel, Bukan Sekadar Validasi Sintaksis

Validasi input merupakan pertahanan krusial terhadap injeksi perintah (*command injection*) dan berbagai serangan lainnya. Semua input, termasuk yang berasal dari antarmuka seperti perintah AT (*AT commands*) dan SMS, harus divalidasi serta dibersihkan (*sanitasi*) secara menyeluruh sebelum diproses. Sanitasi yang tepat memastikan bahwa perintah atau data berbahaya tidak membahayakan perangkat maupun operasionalnya. Hal ini sangat penting untuk antarmuka perintah AT, penangan SMS, profil Bluetooth, pesan diagnostik, serta jalur kendali RF atau seluler yang bersifat eksklusif (*proprietary*), di mana input yang secara sintaksis sah bisa saja tetap merepresentasikan operasi yang tidak aman.

Selain itu, setiap data masukan yang diterima melalui antarmuka nirkabel harus divalidasi lebih lanjut melebihi kebenaran protokol dasar. Misalnya, pemeriksaan masuk akal, pembatasan kecepatan, validasi temporal, dan korelasi lintas sensor sangat penting, terutama untuk sistem siaran dan sistem yang bergantung pada sensor seperti GNSS dan V2X. Misalnya, data GNSS harus dievaluasi berdasarkan sensor inersia dan batasan peta, dan pesan V2X harus dinilai konsistensinya dengan observasi lokal. Selain itu, sistem akses berbasis RF harus memvalidasi waktu, urutan, dan konteks daripada hanya mengandalkan kebenaran kriptografi. Dengan kata lain, validasi semantik mengurangi efektivitas serangan di mana musuh dapat memberikan masukan yang valid secara teknis namun tidak aman secara operasional ke sistem target.

Memperkuat Perangkat Lunak yang Berinteraksi Dengan Jaringan Nirkabel Serta Komponen Pihak Ketiga

Perangkat lunak yang berinteraksi dengan jaringan nirkabel harus diperlakukan sebagai kode berisiko tinggi karena memproses input yang diterima dari jaringan eksternal dan antarmuka radio. Komponen seperti *stack* Bluetooth, layanan Wi-Fi, *daemon* modem seluler, layanan DNS dan DHCP, penangan pesan V2X, serta klien pembaruan harus dirancang dan diuji dengan asumsi bahwa penyerang dapat memengaruhi data yang diproses oleh komponen tersebut secara langsung. Jika memungkinkan, bahasa pemrograman yang aman terhadap memori (*memory-safe languages*) sebaiknya diutamakan untuk komponen baru yang berinteraksi dengan jaringan nirkabel. Jika bahasa yang tidak aman terhadap memori seperti C atau C++ harus digunakan, tim pengembang wajib menerapkan penguatan kompiler (*compiler hardening*) yang ketat, analisis statis, *fuzz testing*, pemindaian dependensi, serta tinjauan kode yang berfokus pada logika pengurai (*parser*) dan penanganan pesan.

Komponen pihak ketiga dan sumber terbuka (*open-source*) harus diinventarisasi, dipantau, dan diperbarui sepanjang siklus hidup kendaraan. Hal ini mencakup pemeliharaan *Software Bill of Materials* (SBOM), pelacakan kerentanan yang diketahui, serta memastikan bahwa dependensi yang rentan khususnya yang berinteraksi dengan jaringan nirkabel dapat diperbaiki (*patched*) tanpa melemahkan autentisitas pembaruan atau menciptakan asumsi kepercayaan (*trust assumptions*) baru.

Membangun Keamanan ke Dalam Siklus Hidup Pengembangan

Keamanan kendaraan nirkabel bukanlah tugas rekayasa yang hanya dilakukan sekali. Perangkat radio, protokol, layanan *backend*, komponen pihak ketiga, serta perangkat yang digunakan oleh penyerang akan terus berkembang seiring waktu. Oleh karena itu, desain yang aman harus didukung oleh berbagai aktivitas keamanan sejak tahap awal siklus hidup pengembangan, yang mencakup pemodelan ancaman, implementasi yang aman, pemantauan keamanan siber, mekanisme pembaruan yang aman, pengujian berkelanjutan, serta fleksibilitas kriptografis jangka panjang.

Tim pengembang harus menerapkan praktik pengembangan perangkat lunak yang aman, seperti penggunaan bahasa pemrograman yang aman terhadap memori (*memory-safe languages*), pelaksanaan tinjauan kode (*code review*), dan pemindaian kerentanan. Semua perangkat lunak yang terhubung secara nirkabel harus menjalani pengujian secara berkala meliputi *fuzzing*, pengujian penetrasi (*penetration testing*), dan pengujian regresi terutama setelah dilakukan pembaruan atau perubahan konfigurasi. Penting juga untuk melakukan pengujian keamanan secara menyeluruh terhadap perangkat lunak pihak ketiga dan komponen OSS sebelum integrasi.

Aktivitas-aktivitas ini sangat penting untuk mengidentifikasi dan menangani kelemahan keamanan selama siklus hidup pengembangan. Mengingat banyak komponen yang terpapar ke jaringan nirkabel memproses input yang dikendalikan oleh penyerang, *fuzzing* harus diperlakukan sebagai aktivitas dasar untuk *parser*, penangan protokol, *daemon* jaringan, klien pembaruan, dan layanan pemrosesan pesan. Selain itu, kendaraan harus mampu menerima pembaruan keamanan untuk komponen nirkabel sepanjang masa pakainya tanpa menimbulkan asumsi kepercayaan (*trust assumptions*) yang baru. Penggunaan tanda tangan kriptografis memastikan bahwa hanya pembaruan yang terautentikasi yang dipasang, sementara protokol aman seperti HTTPS melindungi pembaruan selama proses distribusi.

Di samping itu, pembatasan akses ke server *firmware* serta pemantauan terhadap server tersebut untuk mendeteksi perubahan yang tidak sah sangatlah penting guna mencegah manipulasi dan menjamin keaslian pembaruan. Kolaborasi di seluruh industri otomotif juga akan menjadi krusial, khususnya dalam pemantauan keamanan siber melalui peningkatan pertukaran informasi mengenai kerentanan. Keamanan sepanjang siklus hidup kendaraan sangatlah penting, terutama bagi kendaraan yang memiliki masa pakai lama dan beroperasi dalam ekosistem nirkabel yang terus berubah dengan cepat, di mana ancaman serta serangan baru senantiasa berkembang.

5.7 PENGUJIAN PENETRASI DAN PENGAMANAN SISTEM NIRKABEL KENDARAAN

Sistem nirkabel pada kendaraan secara signifikan memperluas cakupan kerentanan (*attack surface*) kendaraan modern dengan menghadirkan berbagai antarmuka komunikasi eksternal. Selama bertahun-tahun, banyak kerentanan umum telah ditemukan dalam sistem nirkabel kendaraan. Kerentanan tersebut mencakup kredensial yang tertanam secara permanen (*hard-coded*), kunci kriptografi yang digunakan bersama atau bersifat statis, autentikasi yang lemah atau tidak ada sama sekali, layanan debug atau diagnostik yang

terbuka, serta layanan jaringan yang tidak diperlukan namun tetap aktif pada sistem produksi. Beberapa kelemahan juga muncul akibat keterbatasan atau cacat desain dalam protokol nirkabel itu sendiri, yang dapat dieksploitasi meskipun implementasinya telah sesuai dengan spesifikasi protokol tersebut.

Berdasarkan pengamatan ini, bab ini memperkenalkan teknik-teknik pengujian penetrasi (*penetration testing*) yang umum digunakan untuk menilai keamanan sistem nirkabel kendaraan. Kami membahas pendekatan praktis untuk menemukan antarmuka yang terbuka, melakukan enumerasi layanan, serta mengidentifikasi kesalahan konfigurasi atau kelemahan pada tingkat protokol. Beberapa perangkat (*tools*) juga dipaparkan untuk menunjukkan bagaimana penguji dapat memindai kerentanan, menganalisis lalu lintas nirkabel, serta menguji berbagai teknologi komunikasi secara aktif, terukur, dan sistematis. Selain itu, untuk memberikan pemahaman yang lebih baik mengenai penerapan teknik-teknik tersebut dalam situasi nyata, bab ini juga mengulas beberapa studi kasus di mana sistem nirkabel kendaraan berhasil disusupi. Studi kasus ini menggambarkan bagaimana kelemahan yang tampak sepele dapat digabungkan, sehingga memungkinkan penyerang untuk memperoleh akses tanpa izin, meningkatkan hak akses (*privilege escalation*), atau mengganggu fungsi kendaraan.

Terakhir, bab ini menguraikan praktik-praktik terbaik untuk memitigasi masalah keamanan nirkabel yang umum terjadi. Praktik tersebut mencakup pengurangan permukaan serangan dengan menonaktifkan layanan yang tidak diperlukan, menerapkan autentikasi dan otorisasi yang kuat, menghindari penggunaan kredensial yang tertanam secara permanen (*hard-coded*), serta mengikuti pendekatan *secure-by-design* (keamanan sejak tahap perancangan) di sepanjang siklus hidup pengembangan. Dengan menerapkan prinsip-prinsip ini sejak awal dan secara konsisten, produsen serta pengembang dapat meningkatkan ketahanan sistem nirkabel pada kendaraan terhadap serangan di dunia nyata secara signifikan. Setelah menyelesaikan pembahasan mengenai antarmuka nirkabel pada kendaraan, pada bab berikutnya kita akan beralih untuk menelaah keamanan komunikasi di seluruh jaringan internal kendaraan.

BAB 6
KEAMANAN DAN PENGUJIAN JARINGAN INTERNAL KENDARAAN

Dahulu, orang menganggap jaringan di dalam kendaraan hampir mustahil dijangkau oleh penyerang. Namun, seiring dengan berkembangnya berbagai antarmuka konektivitas, peluang untuk menyusup ke dalam jaringan kendaraan pun menjadi semakin terbuka lebar. Dalam skenario yang paling umum, penyerang hanya perlu mendapatkan pijakan pada *Electronic Control Unit* (ECU) yang dapat diakses dari jarak jauh. Dengan begitu, mereka akan berada tepat di tepi jaringan kendaraan dan dapat mulai mengganggu operasional kendaraan tersebut. Akses fisik langsung ke port *On-Board Diagnostics* (OBD)-II merupakan vektor serangan lain yang memungkinkan penyerang memperoleh akses langsung ke jaringan di dalam kendaraan. Risiko keamanan mendasar pada *CAN bus* di dalam kendaraan pertama kali ditunjukkan secara sistematis pada tahun 2010; saat itu, penyerang yang memiliki akses fisik ke port OBD-II mampu menyuntikkan pesan *CAN* arbitrer untuk mengendalikan mesin, rem, serta fungsi kendaraan lainnya melalui berbagai jenis ECU.

Bab ini bertujuan untuk mempelajari protokol komunikasi yang umum digunakan di dalam kendaraan seperti *Controller Area Network* (CAN), *Local Interconnect Network* (LIN), *FlexRay*, dan *Automotive Ethernet* serta menunjukkan bagaimana protokol-protokol tersebut dapat diserang. Untuk itu, kita akan menelaah berbagai kelemahan dari setiap protokol komunikasi sebelum membahas teknik pengujian penetrasi (*penetration testing*) yang dirancang untuk mengeksploitasi kelemahan-kelemahan tersebut. Selanjutnya, kita akan mengeksplorasi metode serangan seperti *sniffing*, *spoofing*, dan intersepsi komunikasi jaringan, sekaligus menyoroti berbagai perangkat yang dapat digunakan untuk melancarkan serangan-serangan tersebut.

Guna menunjukkan betapa nyata risiko terhadap jaringan semacam ini, kami akan menyajikan beberapa studi kasus mengenai serangan nyata yang berhasil menyusup ke dalam protokol jaringan kendaraan. Terakhir, kita akan menarik pelajaran mengenai praktik terbaik keamanan untuk memperkuat protokol jaringan dan meningkatkan tingkat kesulitan dalam melakukan penetrasi. Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

- Gambaran umum komunikasi di dalam kendaraan
- Memahami protokol jaringan di dalam kendaraan
- Kelemahan umum pada komunikasi di dalam kendaraan
- Teknik pengujian penetrasi (*penetration testing*) untuk jaringan di dalam kendaraan
- Studi kasus serangan nyata terhadap komunikasi di dalam kendaraan
- Praktik terbaik untuk mengamankan komunikasi di dalam kendaraan

6.1 PROTOKOL KOMUNIKASI PADA JARINGAN INTERNAL KENDARAAN MODERN

Seiring dengan pergeseran arsitektur kelistrikan/elektronik (E/E) kendaraan dari segmentasi jaringan berbasis domain menuju arsitektur berbasis zona atau komputasi terpusat terjadi peralihan bertahap dari protokol jaringan lama (*legacy*) seperti CAN ke protokol yang lebih baru, contohnya *Automotive Ethernet*. Meskipun jumlah unit kendali elektronik (ECU) berbasis CAN terus berkurang, tampaknya belum ada arsitektur di masa depan yang akan sepenuhnya menghilangkan jaringan CAN. Salah satu alasan utamanya adalah biaya jaringan CAN yang rendah serta keandalannya yang tinggi. Selain itu, karakteristik *real-time* yang dimilikinya menjadikan CAN pilihan ideal bagi ECU dengan sumber daya terbatas yang memerlukan komunikasi yang andal, efisien, dan memiliki latensi rendah melalui bus jaringan.

Pentingnya CAN terletak pada perannya yang terus menangani pesan kontrol dan status yang krusial untuk mesin, transmisi, sistem pengereman, dan sebagainya. Protokol LIN tetap digunakan karena merupakan opsi jaringan berbiaya rendah yang cocok untuk sensor serta aktuator cerdas dengan sumber daya terbatas. FlexRay diperkenalkan untuk memenuhi kebutuhan komunikasi yang deterministik dan tahan terhadap kegagalan (*fault-tolerant*) pada sistem yang sangat kritikal bagi keselamatan seperti suspensi aktif dan sistem kemudi elektronik (*steer-by-wire*) di mana mekanisme berbasis peristiwa (*event-driven*) pada CAN tidak lagi memadai. Dalam beberapa tahun terakhir, *Automotive Ethernet* telah menjadi teknologi jaringan pilihan untuk menangani kebutuhan *bandwidth* yang besar pada kendaraan berbasis perangkat lunak (*software-defined vehicles*).

Baik untuk mentransfer paket pembaruan perangkat lunak berukuran besar ke berbagai ECU, mengambil data diagnostik dalam jumlah besar dari ECU tertentu, maupun mengirimkan data sensor yang kompleks, Ethernet telah membuktikan diri sebagai pilihan jaringan yang tangguh untuk digunakan di dalam kendaraan. Selain itu, kematangan protokol lapisan atas (*higher-layer protocols*) serta ketersediaan perangkat infrastruktur pendukung membuat teknologi ini lebih mudah untuk diadopsi. Protokol seperti *Audio Video Bridging* (AVB) dan *Time-Sensitive Networking* (TSN) menjadikannya pilihan ideal untuk komunikasi yang sensitif terhadap waktu serta untuk alokasi *bandwidth*. Selain bertindak sebagai tulang punggung pusat untuk menghubungkan domain berkecepatan tinggi (seperti unit infotainment, unit telematika, dan *gateway* pusat), Ethernet juga digunakan untuk menghubungkan sensor bandwidth tinggi seperti *Light Detection and Ranging* (LiDA™) dan Kamera. Tabel 6.1 membandingkan protokol jaringan utama dalam kendaraan dan menyoroti peran mereka:

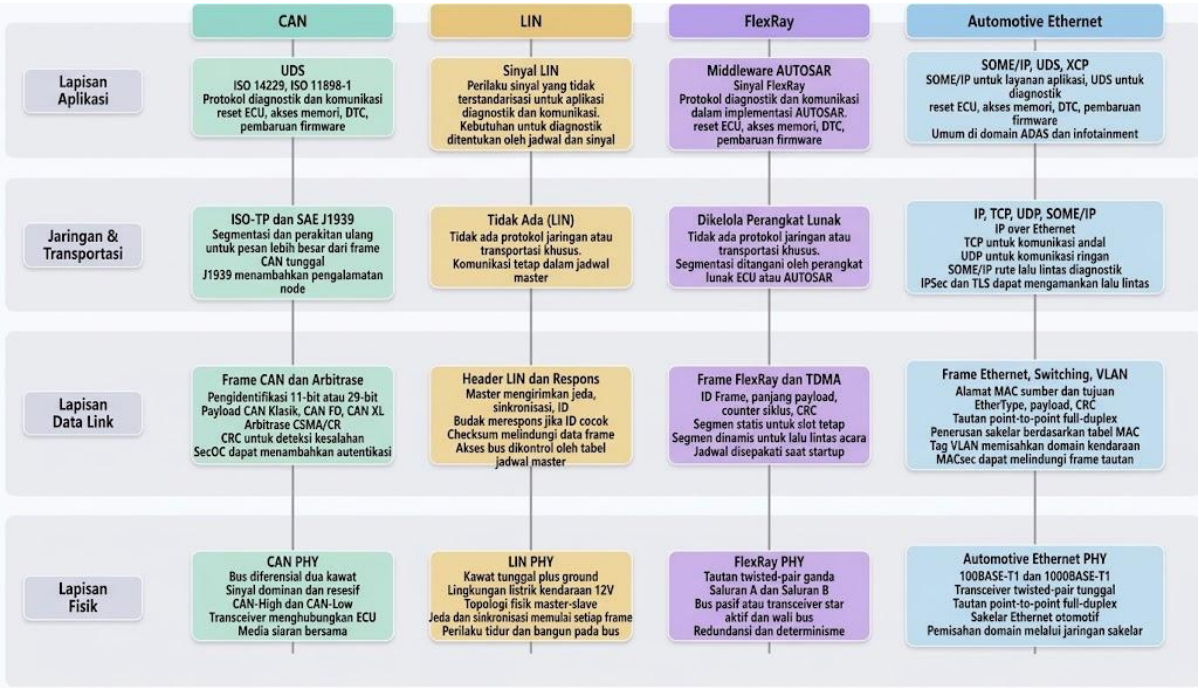
Tabel 6.1 Perbandingan protokol jaringan dalam kendaraan yang umum digunakan

Protokol	**Topologi (Kecepatan)**	**Peran tipikal dalam arsitektur E/E (elektrik/elektronik)**
CAN (Klasik | FD | XL)	Bus (Hingga 1 Mbps untuk versi Klasik; hingga 8 Mbps untuk CAN-FD; hingga 20 Mbps untuk CAN-XL)	Powertrain (sistem penggerak), sasis, kontrol bodi (jaringan kendali waktu nyata/*real-time*)

LIN	Bus kabel tunggal (umumnya 19,2 kbps)	Elektronika bodi, fitur kenyamanan (jaringan berkecepatan rendah dengan biaya murah)
Flex™ay	Bus dua saluran atau topologi bintang (10 Mbps)	Sistem yang kritis terhadap waktu dan keselamatan (contoh: sistem *drive-by-wire* pada desain tertentu)
Ethernet Otomotif	Topologi bintang/pohon dengan switch (100 Mbps, 1 Gbps, 5/10 Gbps)	Tulang punggung (*backbone*) kendaraan dan komputer kendaraan yang terhubung ke sensor-sensor dengan *throughput* (kapasitas transmisi data) tinggi

6.2 ARSITEKTUR DAN KEAMANAN PROTOKOL JARINGAN KENDARAAN

Sebagaimana jaringan pada umumnya, jaringan dalam kendaraan terdiri dari beberapa protokol dan lapisan, mulai dari kabel fisik hingga protokol aplikasi. Memahami setiap lapisan protokol jaringan dalam kendaraan merupakan prasyarat untuk mengidentifikasi kelemahan keamanan yang dapat dieksploitasi selama fase penetrasi. Pada bagian-bagian selanjutnya, kami menyajikan tinjauan singkat mengenai setiap protokol di berbagai lapisan jaringan, yang juga diilustrasikan dalam Gambar 6.1. Anda disarankan untuk mempelajari lebih lanjut mengenai protokol-protokol jaringan yang ingin Anda dalami. Mari kita lihat lapisan-lapisan ini secara detail:



Gambar 6.1 Gambaran umum lapisan dan teknologi jaringan dalam kendaraan

Lapisan Fisik

Pada dasar semua lapisan jaringan terdapat lapisan fisik. Setiap protokol jaringan harus mendefinisikan bagaimana sinyal listrik direpresentasikan pada media fisik, dan komponen apa

yang dibutuhkan untuk memungkinkan komunikasi yang andal. Mari kita lihat bagaimana setiap protokol dalam kendaraan mendefinisikan lapisan fisiknya.

Lapisan Fisik Jaringan Area Pengontrol (CAN PHY)

CAN menggunakan bus diferensial dua kawat (*CAN-High* dan *CAN-Low*) dengan pensinyalan dominan dan resesif. Setiap ECU terhubung ke media bersama melalui transceiver CAN, yang berfungsi sebagai penggerak dan penerima saluran untuk lapisan fisik, biasanya menggabungkan ketahanan listrik dan dalam beberapa implementasi mendukung sinyal wake-on-bus.

Lapisan Fisik Jaringan Interkoneksi Lokal (LIN PHY)

LIN mengimplementasikan topologi satu kawat plus ground dengan arsitektur *master/slave*, di mana satu node master yang ditunjuk mengontrol semua komunikasi. Setiap node terhubung ke bus melalui transceiver LIN, yang menangani konversi tegangan antara mikrokontroler dan sistem kelistrikan kendaraan 12V. Master memulai setiap frame dengan mengirimkan sinyal break diikuti oleh byte sinkronisasi, yang digunakan *node slave* untuk menyinkronkan waktu mereka sebelum merespons. Manajemen daya juga ditangani pada lapisan fisik; ketika bus dalam keadaan idle, semua node memasuki status tidur berdaya rendah dan terbangun secara otomatis saat perangkat induk kembali beraktivitas.

FlexRay PHY

FlexRay menggunakan topologi bus dua saluran, dengan Saluran A (Channel A) dan Saluran B (Channel B) masing-masing membawa sinyal melalui pasangan kabel berpilin (*twisted pair*) khusus. Hal ini memberikan redundansi sekaligus lebar pita (*bandwidth*) yang lebih tinggi dibandingkan protokol saluran tunggal. Setiap node terhubung ke bus melalui transceiver FlexRay, yang menangani konversi sinyal dan dapat mencakup fungsi *bus guardian*. Jika diimplementasikan, *bus guardian* membantu memastikan bahwa suatu node hanya melakukan transmisi selama slot waktu yang telah ditetapkan untuknya, sehingga mencegah satu node mengganggu komunikasi node lainnya. FlexRay mendukung topologi bus pasif maupun topologi bintang aktif (*active star*).

Automotive Ethernet PHY

Automotive Ethernet (berdasarkan standar seperti IEEE 100BASE-T1 dan 1000BASE-T1) menggunakan transceiver khusus untuk satu pasang kabel, yang dirancang agar dapat beroperasi secara andal dalam lingkungan elektromagnetik kendaraan, serta mendukung panjang kabel hingga beberapa meter antar-node. Berbeda dengan Ethernet multi-pasang kabel tradisional yang digunakan dalam jaringan perusahaan, transceiver Automotive Ethernet menjalankan komunikasi *full-duplex* melalui satu pasangan kabel berpilin. ECU terhubung ke jaringan dengan metode *point-to-point* melalui *switch* Ethernet kelas otomotif, yang mengarahkan lalu lintas data ke port tertentu alih-alih menyiarkannya ke seluruh media bersama. Jika dikonfigurasi dengan tepat, *switch* ini dapat mendukung pemisahan domain antar-jaringan kendaraan yang berbeda misalnya, menjaga agar lalu lintas data sistem hiburan (*infotainment*) dan sistem penggerak (*powertrain*) tetap terisolasi satu sama lain.

Data Link Layer

Selanjutnya, di atas lapisan fisik (*physical layer*) terdapat lapisan tautan data (*data link layer*). Lapisan ini mendefinisikan format bingkai (*frame*) dan aturan akses media untuk setiap protokol. Biasanya, lapisan ini diimplementasikan sebagai intellectual property (IP) perangkat keras khusus di dalam unit mikrokontroler (MCU) atau *System on Chip* (SoC) untuk memenuhi kebutuhan pemrosesan waktu nyata (*real-time*) dalam komunikasi di dalam kendaraan.

Data Link CAN

Karena semua node yang terhubung ke bus CAN yang sama berbagi media fisik yang sama, setiap ECU dapat mengirimkan bingkai data ke bus tersebut, dan semua node lain pada segmen yang sama dapat melihatnya. Frame CAN terdiri dari pengidentifikasi (berukuran 11-bit dalam format standar atau 29-bit dalam format diperluas), bit kontrol, muatan data (hingga 8 byte pada CAN klasik, hingga 64 byte pada CAN FD, dan hingga 2048 byte pada CAN XL), serta kolom *Cyclic Redundancy Check* (CRC) untuk deteksi kesalahan. CAN menggunakan metode *carrier-sense multiple access with collision resolution* (CSMA/CR), di mana beberapa node dapat mulai mengirimkan data secara bersamaan; node dengan nilai pengidentifikasi terendah akan memenangkan arbitrase, sementara node lainnya akan berhenti mengirimkan data dan mencoba kembali saat bus dalam keadaan tidak aktif (*idle*).

Tautan Data LIN

Frame LIN terdiri dari header yang dikirimkan oleh node master, yang mencakup field break, byte sinkronisasi (*sync byte*), dan field identifier, diikuti oleh respons yang berisi byte-byte data serta checksum.

Tautan Data FlexRay

Bingkai (*frame*) FlexRay terdiri dari bagian kepala (*header*) yang memuat pengidentifikasi bingkai, panjang muatan (payload), dan penghitung siklus, diikuti oleh muatan data hingga 254 byte serta CRC untuk deteksi kesalahan. Akses bus dikendalikan oleh skema *Time Division Multiple Access* (TDMA) yang dibagi menjadi segmen statis di mana setiap simpul (*node*) telah diberi slot komunikasi tetap dan segmen dinamis yang memungkinkan transmisi berbasis peristiwa dengan prinsip siapa cepat dia dapat (*first-come, first-served*). Jadwal ini dikonfigurasi secara konsisten di seluruh simpul yang berpartisipasi dan bersifat tetap selama operasi berlangsung, sehingga menjamin latensi komunikasi yang dapat diprediksi dan bersifat deterministik.

Tautan Data Ethernet

Ethernet otomotif menggunakan pembingkaian (*framing*) Ethernet standar, dengan setiap bingkai berisi alamat *media access control* (MAC) sumber dan tujuan, kolom *EtherType* yang mengidentifikasi protokol lapisan yang lebih tinggi, muatan data (*data payload*), serta CRC untuk deteksi kesalahan. Karena setiap tautan beroperasi dalam mode *point-to-point full-duplex* antara ECU dan *port* sakelar (*switch*), tidak ada media bersama sehingga tidak ada risiko tabrakan (*collision*) antar-node.

Ethernet mendukung berbagai protokol lapisan yang lebih tinggi secara bersamaan, termasuk IP, *Address Resolution Protocol* (ARP), dan lalu lintas dengan penandaan *Virtual Local Area Network* (VLAN), yang menyediakan landasan fleksibel untuk beragam layanan di dalam

kendaraan. Keputusan penerusan lalu lintas dibuat oleh sakelar berdasarkan tabel alamat MAC, dan penandaan VLAN memungkinkan lalu lintas dari domain kendaraan yang berbeda untuk dipisahkan secara logis meskipun berbagi jaringan Ethernet fisik yang sama. Dari sisi fitur keamanan, MACsec dapat diterapkan untuk menyediakan enkripsi pada tingkat bingkai (*frame-level encryption*) dan autentikasi pesan di antara node Ethernet yang terhubung langsung.

Lapisan Jaringan dan Transport

Di atas lapisan tautan data (*data link layer*) terdapat lapisan jaringan (*network layer*) dan lapisan transport (*transport layer*), yang menangani pengalamatan, perutean (jika diperlukan), serta segmentasi dan penyusunan kembali pesan-pesan yang berukuran lebih besar. CAN mengandalkan lapisan transport tambahan (seperti ISO-TP atau SAE J1939), sementara LIN dan FlexRay tidak mendefinisikan protokol khusus pada lapisan-lapisan ini; adapun Ethernet menggunakan jaringan berbasis IP standar.

Jaringan dan Transport Pada CAN

ISO-TP (ISO 15765-2) adalah protokol lapisan transport di atas CAN yang mendukung transmisi pesan yang melebihi kapasitas muatan (*payload*) maksimum bingkai (*frame*) CAN, dengan cara membagi muatan yang besar tersebut ke dalam beberapa bingkai CAN. Banyak protokol diagnostik dibangun di atas ISO-TP. SAE J1939 adalah rangkaian protokol yang dibangun di atas CAN dan digunakan secara luas pada kendaraan berat. Protokol ini mendefinisikan fungsionalitas lapisan jaringan (*network layer*), lapisan transpor (*transport layer*), dan lapisan aplikasi (*application layer*) tersendiri. SAE J1939 memperkenalkan skema pengalamatan node, di mana setiap node diberi alamat sumber unik untuk mengidentifikasinya di sepanjang bus, serta menyediakan mekanisme lapisan transpor untuk transfer siaran (*broadcast*) yang melibatkan banyak paket maupun koneksi antar-node (*peer-to-peer*).

Lapisan Jaringan dan Transpor Pada LIN

LIN tidak mendefinisikan protokol lapisan jaringan atau lapisan transpor secara khusus. Sebaliknya, seluruh komunikasi diatur dalam jadwal master/slave. LIN biasanya digunakan untuk aplikasi sederhana berkecepatan rendah yang tidak memerlukan pengalamatan atau segmentasi pesan.

Lapisan Jaringan dan Transpor Pada FlexRay

Mirip dengan LIN, FlexRay juga tidak mendefinisikan protokol lapisan jaringan atau lapisan transpor secara khusus. Identifikasi node didasarkan pada penempatan slot dalam jadwal komunikasi statis, bukan menggunakan alamat jaringan. Untuk aplikasi yang memerlukan segmentasi muatan (*payload*) yang lebih besar daripada satu frame FlexRay, proses ini ditangani oleh tumpukan perangkat lunak (*software stack*) pada ECU biasanya melalui middleware komunikasi AUTOSAR yang mengelola segmentasi dan penyusunan kembali data di atas lapisan tautan data (*data link layer*).

Lapisan Jaringan dan Transpor Pada Ethernet

ECU yang berkomunikasi melalui Ethernet menggunakan jaringan berbasis IP standar, dengan alamat IP yang ditetapkan baik secara statis maupun dinamis. Pada lapisan transpor,

digunakan *Transmission Control Protocol* (TCP) atau *User Datagram Protocol* (UDP) tergantung pada kebutuhan aplikasi: TCP untuk pengiriman yang andal dan berorientasi koneksi, sedangkan UDP untuk komunikasi yang ringan dan tanpa koneksi (*connectionless*). Selain itu, *Diagnostics over Internet Protocol* (DoIP) (ISO 13400) mendefinisikan bagaimana pesan diagnostik dirutekan melalui TCP dan UDP untuk mencapai ECU target. Terkait keamanan, pada lapisan jaringan, *Internet Protocol Security* (IPsec) dapat digunakan untuk melindungi komunikasi lapisan IP antar-titik akhir (*endpoint*), sementara pada lapisan transpor, *Transport Layer Security* (TLS) dapat digunakan untuk mengamankan sesi aplikasi.

Lapisan Aplikasi

Terakhir, lapisan aplikasi berada di tingkat paling atas dalam tumpukan jaringan kendaraan dan mengatur konten serta perilaku komunikasi antar-ECU. Berbeda dengan lapisan-lapisan di bawahnya, protokol lapisan aplikasi dalam jaringan otomotif tidak selalu terikat pada satu teknologi bus tertentu. artinya, beberapa protokol dapat beroperasi melalui berbagai mekanisme transpor. Perlu dicatat bahwa LIN dan FlexRay tidak menetapkan protokol lapisan aplikasi yang terstandarisasi; perilaku aplikasi dalam jaringan-jaringan ini dikelola melalui definisi sinyal dan bingkai (*frame*) yang masing-masing ditentukan dalam berkas konfigurasi serta *middleware* AUTOSAR. Oleh karena itu, protokol yang kita bahas dalam bagian ini berkaitan dengan CAN dan Ethernet.

Unified Diagnostic Services (UDS)

UDS (ISO 14229) merupakan protokol permintaan/respons diagnostik standar yang digunakan di seluruh industri otomotif. Protokol ini beroperasi melalui CAN dengan menggunakan ISO-TP, serta melalui Ethernet dengan menggunakan DoIP. UDS menyediakan berbagai layanan, termasuk pembacaan kode masalah diagnostik (*diagnostic trouble codes*), pengaturan ulang ECU (*reset ECU*), operasi baca/tulis memori, kontrol rutin, dan alur kerja pembaruan *firmware*. Protokol ini ditujukan untuk digunakan oleh klien diagnostik resmi di bengkel maupun lingkungan produksi.

Scalable Service-Oriented Middleware over IP (SOME/IP)

SOME/IP adalah protokol komunikasi berorientasi layanan yang digunakan dalam lingkungan AUTOSAR Adaptive melalui Ethernet. Protokol ini menyediakan fitur penemuan layanan (*service discovery*) dan serialisasi pesan antar-ECU. SOME/IP umumnya digunakan untuk komunikasi antar-layanan aplikasi dalam domain seperti sistem hiburan (*infotainment*) dan *Advanced Driver Assistance Systems* (ADAS).

Universal Measurement and Calibration Protocol (XCP)

XCP digunakan melalui CAN maupun Ethernet, terutama selama tahap pengembangan kendaraan untuk keperluan pengukuran dan kalibrasi secara langsung (live). XCP menawarkan berbagai fitur bermanfaat, seperti kemampuan membaca dan menulis pada alamat memori tertentu, serta mendukung pemrograman *flash*. Hal ini memungkinkan para insinyur untuk menyesuaikan parameter kontrol seperti waktu injeksi bahan bakar secara presisi saat sistem sedang berjalan (*runtime*).

Secure Onboard Communication (SecOC)

SecOC merupakan mekanisme keamanan yang ditetapkan oleh AUTOSAR (*AUTomotive Open System Architecture*). Mekanisme ini menjamin keaslian pesan dan perlindungan terhadap serangan pengulangan (*replay attack*) untuk *Protocol Data Unit* (PDU) yang bersifat kritis, terlepas dari protokol bus yang mendasarinya. Cara kerjanya adalah dengan menambahkan nilai kesegaran (*freshness value*) dan *Message Authentication Code* (MAC) pada PDU yang dikirim, kemudian memverifikasinya saat pesan tersebut diterima. SecOC dapat diterapkan pada CAN, CAN-FD, maupun Ethernet, menjadikannya mekanisme yang tidak bergantung pada protokol tertentu (*protocol-agnostic*) dalam tumpukan komunikasi AUTOSAR. Setelah meninjau tumpukan protokol jaringan dalam kendaraan yang mencakup lapisan fisik (*physical*), tautan data (*data link*), jaringan (*network*), transport, dan aplikasi langkah selanjutnya adalah memeriksa kelemahan umum yang terdapat dalam komunikasi tersebut serta bagaimana penyerang dapat memanfaatkannya.

6.3 KELEMAHAN UMUM DALAM KOMUNIKASI DI DALAM KENDARAAN

Untuk melaksanakan pengujian penetrasi (*penetration testing*) yang efektif, sangat penting untuk mengidentifikasi berbagai kelemahan yang ada pada jaringan target maupun protokol komunikasinya. Bahkan saat ini, masih banyak insinyur otomotif yang meyakini bahwa jaringan di dalam kendaraan merupakan lingkungan yang tertutup dan tepercaya. Sayangnya, pandangan ini menciptakan anggapan yang berlebihan mengenai betapa sulitnya bagi penyerang untuk mengakses jaringan di dalam kendaraan tersebut. Protokol lama seperti CAN dan LIN dirancang sebelum era konektivitas berkembang, sehingga secara inheren tidak aman. FlexRay memang lebih modern dibandingkan CAN dan LIN, namun mengikuti pola yang sama:

Dirancang agar bersifat deterministik dan toleran terhadap kesalahan (*fault-tolerant*), namun tanpa disertai keamanan bawaan. Dengan hadirnya Automotive Ethernet, dukungan keamanan bawaan seperti autentikasi, segmentasi, dan enkripsi telah meningkatkan aspek keamanan; meskipun demikian, kelemahan tetap dapat muncul akibat masalah konfigurasi dan implementasi. Seiring dengan evolusi kendaraan dari sistem yang terisolasi secara mekanis menjadi platform berbasis perangkat lunak yang saling terhubung dan memiliki konektivitas eksternal, kelemahan-kelemahan ini telah berubah menjadi permukaan serangan (*attack surface*) yang dapat dieksploitasi oleh pihak penyerang.

Bagian ini mengkaji kelemahan-kelemahan tersebut secara sistematis, dengan fokus mendalam pada dua protokol yang paling umum digunakan dan memiliki risiko tinggi, yaitu CAN dan Automotive Ethernet, diikuti oleh LIN dan FlexRay. Untuk setiap protokol, analisis mencakup keterbatasan desain yang melekat, kelemahan pada perangkat lunak komunikasi lapisan atas (*upper-layer*), serta pertimbangan topologi yang memengaruhi permukaan serangan. Gambaran umum mengenai kelemahan yang lazim ditemukan pada jaringan di dalam kendaraan (*in-vehicle network*) disajikan dalam Gambar 6.2.



Gambar 6.2 Kelemahan umum pada jaringan di dalam kendaraan

CAN

CAN dirancang untuk keandalan dan determinisme di lingkungan dengan gangguan elektrik tinggi, bukan untuk keamanan. Keputusan desain yang membuat CAN tangguh untuk tujuan awalnya justru menciptakan kerentanan bagi penyerang yang memiliki akses ke bus tersebut.

Arsitektur Siaran dan Ketiadaan Autentikasi Sumber

CAN beroperasi menggunakan topologi bus bersama (*shared bus*), di mana setiap pesan yang dikirimkan oleh satu node akan diterima oleh semua node lain dalam segmen bus yang sama. Tidak ada konsep pengalamatan sumber pada lapisan tautan data (*data link layer*) CAN. Pengidentifikasi bingkai (*frame identifier*) dalam CAN hanya menentukan jenis pesan, bukan identitas pengirimnya. Dengan demikian, setiap node yang terhubung dapat mengirimkan pesan dengan pengidentifikasi apa pun, sementara node penerima tidak memiliki mekanisme di tingkat protokol untuk memverifikasi keabsahan sumber pesan tersebut.

Ketiadaan Dukungan Enkripsi dan Autentikasi

CAN tidak menyediakan dukungan enkripsi maupun autentikasi bawaan. Semua bidang data dikirimkan dalam bentuk teks polos (*plaintext*) melalui bus. Penyerang yang terhubung ke bus fisik atau memiliki akses ke node yang telah disusupi dapat melakukan pemantauan pasif untuk mendapatkan gambaran lengkap mengenai seluruh komunikasi di dalam kendaraan yang melalui bus tersebut. Selain itu, karena tidak adanya dukungan autentikasi pesan, node yang disusupi dapat mengirimkan pesan palsu (*spoofed*), pesan yang telah diubah, atau pesan yang dikirim ulang (*replayed*), tanpa kemampuan bagi penerima untuk mendeteksi bahwa pesan tersebut tidak sah. Dalam domain yang krusial bagi keselamatan seperti sistem penggerak (*powertrain*) dan sasis, hal ini menimbulkan risiko operasional bagi ECU yang

mengandalkan data jaringan, seperti kecepatan mesin, posisi katup gas (*throttle*), tekanan rem, sudut kemudi, dan perintah pengereman.

Tidak Adanya Kontrol Akses

CAN tidak memiliki mekanisme kontrol akses bawaan. Artinya, tidak ada daftar putih (*whitelist*) untuk pengirim yang diizinkan bagi setiap pengidentifikasi pesan, tidak ada penyaringan berbasis peran, dan tidak ada mekanisme untuk mencegah suatu node mengirimkan pesan dengan pengidentifikasi di luar lingkup fungsional yang seharusnya. Ini berarti bahwa ECU yang telah disusupi, atau node berbahaya yang dipasang oleh penyerang ke dalam bus, dapat mengirimkan pesan apa pun tanpa adanya pertahanan di tingkat protokol untuk menghentikannya.

Mekanisme Bingkai Kesalahan (Error Frame) dan Kondisi Bus-Off

Mekanisme penanganan kesalahan pada CAN, meskipun dirancang untuk menjamin keandalan, justru menciptakan kerentanan terhadap ketersediaan sistem (*availability*). Setiap node CAN memiliki *Transmit Error Counter* (TEC) dan *Receive Error Counter* (REC). Ketika penghitung ini melampaui ambang batas tertentu, node tersebut akan beralih ke status *Warning* (peringatan) dan *Error Passive*. Lebih lanjut, jika nilai TEC melebihi 255, node tersebut akan beralih ke status *Bus-Off*; pada tahap ini, node tersebut akan diputus dari bus dan seluruh komunikasinya akan terhenti. Seorang penyerang dapat dengan sengaja memicu kondisi ini dengan membuat node target berulang kali menganggap transmisinya sendiri sebagai transmisi yang keliru, sehingga penghitung kesalahan TEC (*Transmit Error Counter*) pada node tersebut terus bertambah hingga akhirnya mencapai status *Bus-Off*. Akibatnya, serangan *Bus-Off* terhadap ECU yang krusial bagi keselamatan seperti pengontrol sistem pengereman anti-terkunci (*Anti-lock Braking System*/ABS) atau modul kendali kantung udara (*airbag*) dapat berujung pada serangan penolakan layanan (*denial-of-service*) yang berdampak pada aspek keselamatan.

Kelemahan Lapisan Atas ISO-TP

ISO-TP adalah lapisan protokol transpor yang digunakan pada CAN untuk mendukung pesan yang melebihi batas muatan (*payload*) satu bingkai (*frame*) CAN (8 byte pada CAN 2.0, dan hingga 64 byte pada CAN FD). ISO-TP menghadirkan serangkaian kelemahannya sendiri pada lapisan transpor komunikasi. ISO-TP mendefinisikan empat jenis bingkai:

- ***Single frame* (bingkai tunggal)**: memuat data hingga 7 byte saat dikirim dalam bingkai CAN klasik; hingga 62 byte saat dikirim dalam bingkai CAN FD.
- ***First frame* (bingkai pertama)**: memulai transmisi multi-bingkai, termasuk informasi mengenai total panjang pesan.
- ***Consecutive frame* (bingkai lanjutan)**: membawa segmen-segmen berikutnya disertai dengan nomor urut (*sequence number*).
- ***Flow control* (kontrol aliran)**: dikirim oleh penerima untuk mengatur laju dan ritme transmisi pengirim.

Kelemahan-kelemahan berikut relevan dari perspektif keamanan:

- **Kehabisan buffer (*buffer exhaustion*) melalui manipulasi *First Frame***: Penyerang dapat mengirimkan *First Frame* yang menyatakan panjang pesan yang besar (hingga 4095

byte dalam format panjang *First Frame* 12-bit klasik, atau lebih besar pada versi ISO 15765-2 yang lebih baru) tanpa diikuti oleh *Consecutive Frame*. ECU penerima akan mengalokasikan buffer penyusunan ulang (*reassembly buffer*) dan membiarkannya tetap terbuka sembari menunggu segmen-segmen selanjutnya. Penyerang dapat mengirimkan beberapa transmisi tidak lengkap semacam itu ke ECU yang memiliki sumber daya terbatas untuk menghabiskan memori yang tersedia.

- **Manipulasi nomor urut**: *Consecutive Frame* membawa nomor urut 4-bit (0x0 hingga 0xF). Penyerang dapat menyisipkan *Consecutive Frame* dengan nomor urut yang salah atau tidak terduga selama transmisi multi-bingkai berlangsung. Hal ini dapat menyebabkan penerima membatalkan sesi, merusak buffer penyusunan ulang, atau pada implementasi perangkat lunak (*stack*) yang buruk memicu perilaku yang tidak terdefinisi.
- **Penyisipan bingkai *Flow Control***: Penyerang dapat menyisipkan bingkai *Flow Control* palsu untuk mengganggu sesi diagnostik yang sedang berlangsung. Sebagai contoh, penyerang dapat memalsukan bingkai Flow Control dengan nilai *Separation Time Minimum* (STmin) sebesar 0x7F (127 ms). Akibatnya, pengirim yang sah akan dipaksa untuk menyisipkan jeda 127 ms di antara setiap *Consecutive Frame*, yang menurunkan *throughput* hingga mendekati nol dan secara efektif menghentikan sesi tersebut.

Kelemahan Lapisan Atas SAE J1939

SAE J1939 menyediakan protokol terstandarisasi untuk komunikasi kendaraan komersial, termasuk lapisan protokol transpor yang dirancang untuk menangani pesan multi-paket yang melebihi delapan byte. Akan tetapi, terdapat beberapa kelemahan utama pada lapisan-lapisan protokol bagian atas. Beberapa contohnya meliputi hal-hal berikut:

- **Kehabisan buffer (*buffer exhaustion*) melalui pesan TP.CM_BAM yang cacat**: Pesan broadcast announce (TP.CM_BAM) ini menginstruksikan ECU penerima untuk mengalokasikan buffer terlebih dahulu bagi transmisi multi-paket yang akan datang. Penyerang yang menyuntikkan bingkai (*frame*) TP.CM_BAM yang telah dimanipulasi dengan menyatakan ukuran pesan yang sangat besar dapat memaksa ECU target untuk mengalokasikan memori secara berlebihan. Hal ini menyebabkan kehabisan sumber daya memori dan berpotensi mengakibatkan sistem mengalami crash, tanpa memerlukan autentikasi apa pun sebelumnya.
- **Manipulasi jumlah paket di luar rentang melalui TP.CM_RTS**: ECU yang menerima pesan *request-to-send* (TP.CM_RTS) mungkin memproses jumlah paket yang dideklarasikan tanpa pemeriksaan batas (*bounds checking*) yang memadai. Dengan demikian, penyerang yang mengirimkan bingkai TP.CM_RTS yang telah dimanipulasi dengan jumlah total paket di luar rentang yang diizinkan dapat menyebabkan ECU mengalokasikan atau mengakses memori di luar batas yang seharusnya. Akibatnya, terjadi kerusakan memori (*memory corruption*) atau crash pada ECU target.
- **Kurangnya autentikasi pengirim**: Protokol J1939 mewarisi sifat broadcast dari CAN dan tidak menyediakan mekanisme untuk mengautentikasi alamat sumber suatu bingkai. Oleh karena itu, penyerang yang memiliki akses ke bus dapat mengirimkan bingkai

kendali ke ECU target mana pun dengan menyamar sebagai simpul (*node*) jaringan yang sah.

Kelemahan Pada Lapisan Aplikasi UDS

Meskipun UDS menyediakan berbagai layanan diagnostik yang berguna untuk pemeriksaan dan identifikasi masalah pada ECU, terdapat beberapa kelemahan utama. Beberapa contohnya meliputi:

- ***Brute-force* pada akses keamanan (*security access*)**: Layanan akses keamanan (*Security Access* – Layanan 0x27) menggunakan mekanisme tantangan-respons (*challenge-response*) berbasis seed-key. Jika implementasinya menggunakan algoritma derivasi kunci yang statis, lemah, atau deterministik, penyerang dapat melakukan *brute-force* atau memprediksi respons kunci yang tepat. Selain itu, banyak ECU produksi yang tidak menerapkan pembatasan percobaan (*attempt-limiting*) secara memadai, sehingga penyerang dapat terus-menerus mengulangi proses tantangan-respons seed-key tanpa adanya penguncian sistem (*lockout*).
- **Pintasan autentikasi (*authentication bypass*) dan kegagalan validasi sertifikat**: Layanan autentikasi (*Authentication* – Layanan 0x29), jika diimplementasikan, menggunakan pertukaran sertifikat berbasis *Public Key Infrastructure* (PKI) untuk mengidentifikasi klien dan memberikan hak akses. Jika validasi tanda tangan dan rantai sertifikat, pemeriksaan masa berlaku atau pencabutan sertifikat, verifikasi bukti kepemilikan, ataupun pengaitan peran yang terautentikasi dengan layanan yang diizinkan tergolong lemah, penyerang dapat melakukan autentikasi menggunakan sertifikat yang tidak sah atau memperoleh akses yang lebih luas daripada yang seharusnya.
- **Eskalasi sesi tanpa autentikasi atau dengan autentikasi yang tidak memadai**: *DiagnosticSessionControl* (Layanan 0x10) memungkinkan transisi ke *Extended Diagnostic Session* (Sesi Diagnostik Lanjutan) atau *Programming Session* (Sesi Pemrograman). Pada banyak ECU, sesi-sesi ini dapat diakses tanpa autentikasi awal yang cukup kuat, sehingga memberikan akses ke layanan sensitif seperti pembacaan/penulisan pengidentifikasi data (*Data Identifier* — Layanan 0x22/0x2E), pembacaan/penulisan memori berdasarkan alamat (Layanan 0x23/0x3D), pengendalian rutin (*Routine Control* — Layanan 0x31) untuk memicu prosedur yang telah ditetapkan sebelumnya (yang dapat memengaruhi aspek keselamatan), serta pengaturan ulang ECU (ECU *Reset* — Layanan 0x11). Dengan akses ke layanan-layanan tersebut, penyerang dapat membaca dan menimpa data sensitif, menjalankan prosedur yang ditetapkan oleh produsen termasuk penghapusan memori *flash*, transisi ke mode boot, atau pengujian actuator atau melakukan pengaturan ulang (*reboot*) pada ECU secara berulang untuk menciptakan kondisi penolakan layanan (*denial-of-service*), tergantung pada layanan spesifik mana yang dibiarkan tanpa perlindungan.
- **Jalur pengunduhan dan pemrograman yang tidak aman**: *RequestDownload*, *TransferData*, dan *RequestTransferExit* (Layanan 0x34, 0x36, dan 0x37) menyediakan mekanisme standar untuk menulis data ke dalam ECU. Jika proses masuk ke sesi

pemrograman, autentikasi, validasi rentang alamat, pemeriksaan integritas, atau verifikasi tanda tangan lemah, penyerang dapat mengunggah *firmware*, data kalibrasi, atau data konfigurasi yang berbahaya. Pada implementasi yang menggabungkan urutan pengunduhan dengan rutinitas aktivasi yang ditetapkan oleh produsen, hal ini dapat mengakibatkan eksekusi kode yang bersifat persisten (menetap).

- P**enekanan komunikasi dan penyembunyian kesalahan (*fault hiding*)**: *CommunicationControl* (Layanan 0x28) dapat menonaktifkan komunikasi ECU, *ControlDTCSetting* (Layanan 0x85) dapat menekan pembaruan kode masalah diagnostik (DTC), dan *TesterPresent* (Layanan 0x3E) dapat menjaga sesi dengan hak akses khusus tetap aktif tanpa batas waktu. Jika layanan-layanan ini tidak dilindungi dengan memadai, penyerang dapat mengurangi visibilitas jaringan, menghambat pencatatan kesalahan, dan memperpanjang durasi waktu untuk melakukan tindakan berbahaya lebih lanjut.

Kelemahan Komunikasi XCP

XCP dirancang sebagai protokol untuk keperluan rekayasa (*engineering*) dan seharusnya dinonaktifkan pada kendaraan produksi. Jika XCP secara tidak sengaja tertinggal tanpa autentikasi yang tepat, protokol ini dapat digunakan untuk melakukan operasi invasif, seperti mengubah konten memori secara langsung (*on-the-fly*), yang dapat berujung pada eksekusi kode arbitrer (sewenang-wenang). XCP beroperasi melalui berbagai lapisan transport, termasuk XCP pada CAN (XCPoC) dan XCP pada Ethernet (XCPoE), sehingga relevan untuk kedua jenis jaringan tersebut. Kelemahan utama dalam XCP meliputi hal-hal berikut:

- **Koneksi dan akses memori tanpa autentikasi**: Perintah *CONNECT* pada XCP memulai sesi tanpa persyaratan autentikasi dalam spesifikasi dasarnya. Setelah terhubung, pihak pengontrol (*master*) dapat mengeluarkan perintah SHORT_UPLOAD untuk membaca wilayah memori ECU mana pun, serta SHORT_DOWNLOAD atau DOWNLOAD untuk menulis data ke wilayah tersebut. Dengan demikian, jika XCP tidak dinonaktifkan pada ECU kendaraan produksi, penyerang dapat memiliki kemampuan untuk membaca dan menulis memori secara tidak terbatas melalui bus CAN.
- **Perlindungan *seed-key* yang tidak ada atau diimplementasikan dengan lemah**: XCP menetapkan mekanisme perlindungan *seed-key* opsional untuk sumber daya yang memerlukan hak akses khusus. Namun, fitur ini sering kali dinonaktifkan selama tahap pengembangan dan pengujian, sehingga secara tidak sengaja tetap dalam kondisi nonaktif saat masuk ke tahap produksi. Bahkan jika diaktifkan, dalam banyak implementasi, fitur ini menggunakan algoritma kunci yang lemah atau sudah diketahui publik. Penyerang yang berhasil memperoleh algoritma kunci (yang sering kali dapat diekstraksi dari *firmware* ECU) dapat membuka akses ke semua sumber daya yang dilindungi.
- **Eksfiltrasi data pengukuran**: Mode Akuisisi Data (DAQ) pada XCP memungkinkan pengaliran (*streaming*) variabel internal ECU secara terus-menerus. Jika didukung, hal ini memberikan penyerang saluran akses baca yang persisten dan berfrekuensi tinggi terhadap kondisi internal ECU.

Kelemahan Implementasi SecOC

SecOC merupakan mekanisme utama untuk menambahkan autentikasi pesan pada komunikasi CAN, dengan cara melampirkan MAC (*Message Authentication Code*) dan nilai kesegaran (*freshness value*) pada setiap PDU yang diamankan. Untuk frame CAN 2.0 maupun CAN FD, MAC dan nilai kesegaran biasanya dipotong (*truncated*) agar tersedia ruang bagi data payload. Sekumpulan kunci yang dibagikan sebelumnya (*pre-shared keys*) di antara ECU pengirim dan penerima digunakan untuk menghasilkan serta memverifikasi MAC tersebut. Meskipun SecOC berfungsi sebagai mitigasi terhadap komunikasi tanpa autentikasi bukan merupakan kelemahan bawaan protokol itu sendiri implementasi SecOC dapat menciptakan celah serangan baru jika terdapat kesalahan dalam penerapannya.

Contoh kelemahan implementasi tersebut antara lain:

- **MAC yang dipotong dengan panjang bit yang tidak memadai**: SecOC memungkinkan pemotongan MAC hingga hanya 24 bit untuk menyesuaikan dengan batasan *payload* pada CAN. Spesifikasi AUTOSAR mengakui bahwa pemotongan MAC akan menurunkan tingkat keamanan dan merekomendasikan penggunaan setidaknya 64 bit untuk memberikan perlindungan yang memadai terhadap serangan tebakan (*guessing attacks*). MAC 24-bit hanya menyediakan 16 juta kemungkinan nilai; artinya, keamanan setiap pesan yang diautentikasi sangat bergantung pada implementasi nilai kesegaran yang tepat guna mencegah upaya tebakan berulang. Dalam penerapan di mana pengelolaan nilai kesegaran lemah atau tidak ada, ruang MAC yang terbatas tersebut menjadi celah serangan yang nyata.
- ***Rollover* (pengulangan kembali) nilai kesegaran dan kehabisan penghitung (*counter exhaustion*)**: Nilai kesegaran dalam SecOC mencegah serangan *replay* (pengiriman ulang pesan) dengan memastikan setiap pesan yang diautentikasi membawa nilai penghitung yang unik selama masa pakai kunci MAC. Karena SecOC mengandalkan kunci MAC jangka panjang yang dibagikan sebelumnya, penghitung kesegaran tidak boleh mengalami *rollover* selama kunci tersebut masih digunakan. Jika terdapat kelemahan protokol atau cacat implementasi yang memungkinkan penyerang untuk mempercepat penghitung misalnya, memaksa penghitung maju dengan cepat tanpa adanya lalu lintas pesan yang sah maka penghitung tersebut dapat habis sebelum waktunya. Begitu rollover terjadi, penyerang dapat mengirimkan ulang (*replay*) *frame* yang telah diautentikasi sebelumnya yang memiliki nilai penghitung dan kunci yang sama sehingga melumpuhkan perlindungan anti-replay yang dirancang oleh SecOC.
- **Pintasan verifikasi (*verification bypass*) pada implementasi penerima**: Dalam beberapa implementasi produksi, kegagalan verifikasi SecOC tidak menyebabkan PDU dibuang, melainkan tetap diteruskan ke lapisan aplikasi (*application layer*) dengan menyertakan penanda (*flag*) bahwa verifikasi telah gagal. Jika aplikasi tidak memeriksa tanda kegagalan (*failure flag*) ini dan menanganinya dengan tepat, suatu kelalaian yang sering terjadi pada integrasi awal SecOC penyerang dapat menyisipkan pesan yang tidak terautentikasi, yang kemudian tetap diproses oleh aplikasi target seolah-olah verifikasinya berhasil.

- **Manajemen kunci dan paparan kunci simetris**: SecOC bergantung pada kunci simetris yang digunakan bersama oleh ECU-ECU yang saling berkomunikasi. Jika, misalnya, kunci yang sama digunakan kembali di berbagai ECU, varian kendaraan, atau tahun model, maka kompromi terhadap materi kunci pada satu ECU sebagai contoh, melalui ekstraksi *firmware* atau pemeriksaan saluran samping (*side-channel analysis*) akan berdampak pada semua ECU yang menggunakan kunci tersebut. Hal ini merupakan kelemahan dalam manajemen kunci, bukan kelemahan pada SecOC itu sendiri; namun, sangatlah penting bagi organisasi untuk menerapkan langkah-langkah yang tepat dalam pengelolaan kunci.

Ethernet Otomotif

Ethernet Otomotif telah menjadi jaringan tulang punggung (*backbone*) untuk domain yang membutuhkan bandwidth besar pada kendaraan modern. Berbeda dengan CAN, Ethernet menyediakan bandwidth yang memadai untuk mendukung protokol keamanan TI standar, seperti TLS. Namun, transisi dari arsitektur yang berpusat pada CAN menuju arsitektur berbasis zona atau domain dengan Ethernet telah menghadirkan permukaan serangan (*attack surface*) yang baru dan lebih luas, terutama jika aspek keamanan tidak dipertimbangkan sejak awal.

Salah Konfigurasi atau Kurangnya Segmentasi Jaringan

Jaringan Ethernet Otomotif diatur secara logis menggunakan VLAN (IEEE 802.1Q) untuk memisahkan lalu lintas antar domain fungsional; misalnya, mengisolasi lalu lintas sensor ADAS dari sistem infotainment atau telematika. Jika segmentasi VLAN dikonfigurasi dengan salah atau tidak diterapkan, jaringan tersebut dapat beroperasi layaknya domain siaran (*broadcast domain*) datar, mirip dengan CAN. Dengan demikian, penyerang yang berhasil menyusupi satu node dalam jaringan dapat memperoleh visibilitas terhadap lalu lintas dari semua domain fungsional, serta kemampuan untuk menyuntikkan paket yang menargetkan ECU mana pun yang terhubung. Contoh kesalahan konfigurasi yang dapat menimbulkan kerentanan antara lain:

- **Ketidakcocokan VLAN asli (*native* VLAN) pada port trunk yang memungkinkan VLAN hopping**: Dalam trunking 802.1Q, VLAN asli membawa lalu lintas yang tidak diberi tag (*untagged traffic*). Jika dua port trunk yang terhubung dikonfigurasi dengan VLAN asli yang berbeda, frame tanpa tag yang dikirim dari satu sisi akan ditetapkan ke VLAN yang salah di sisi penerima. Akibatnya, *frame* tersebut melintasi batas VLAN tanpa melalui router atau *gateway*, sehingga secara efektif memintas kebijakan kontrol akses antar-VLAN.
- **Double-tagging yang memungkinkan VLAN hopping**: Dalam trunking 802.1Q, penyerang dapat menyusun *frame* dengan dua tag VLAN, di mana tag luar sesuai dengan VLAN asli, sementara tag dalam menargetkan VLAN yang berbeda. Switch pertama akan menghapus tag luar dan meneruskan frame tersebut sebagai lalu lintas VLAN asli, sedangkan switch kedua akan memproses tag dalam dan mengirimkannya ke VLAN tujuan sehingga memintas kontrol akses antar-VLAN. Hal ini memungkinkan node yang telah disusupi di domain dengan tingkat kepercayaan lebih rendah (seperti

sistem infotainment) untuk menyuntikkan lalu lintas ke dalam domain dengan tingkat kepercayaan lebih tinggi (seperti ADAS atau sistem penggerak/*powertrain*).

- **Antarmuka manajemen yang dapat diakses dari VLAN tamu atau VLAN yang tidak tepercaya**: Antarmuka manajemen pada switch dan *gateway* yang biasanya digunakan untuk konfigurasi, pembaruan *firmware*, dan pemantauan seharusnya hanya dapat diakses melalui VLAN manajemen khusus yang dibatasi. Namun, jika antarmuka ini dapat diakses dari VLAN dengan tingkat kepercayaan lebih rendah, seperti domain sistem hiburan (*infotainment*) atau telematika, maka node yang telah disusupi di domain tersebut dapat berinteraksi langsung dengan antarmuka manajemen.
- **Tidak adanya atau longgarnya penyaringan lalu lintas keluar (*egress filtering*) antar segmen domain**: Tanpa aturan penyaringan lalu lintas keluar yang tegas di batas-batas domain, lalu lintas yang berasal dari segmen dengan tingkat kepercayaan lebih rendah dapat mencapai segmen dengan tingkat kepercayaan lebih tinggi tanpa pemeriksaan. Sebagai contoh, ECU sistem hiburan yang telah disusupi dapat mengirimkan paket secara langsung ke domain ADAS atau powertrain jika *gateway* tidak menerapkan penyaringan lalu lintas keluar pada segmen sistem hiburan tersebut.
- **Tidak diterapkannya kontrol akses berbasis port (IEEE 802.1X) pada port switch Ethernet**: Standar 802.1X mengharuskan perangkat untuk melakukan autentikasi sebelum diberikan akses jaringan di tingkat port. Jika fitur ini tidak diaktifkan, perangkat apa pun yang terhubung secara fisik ke port Ethernet akan langsung mendapatkan akses jaringan tanpa verifikasi identitas. Penyerang yang memiliki akses fisik ke bagian dalam kendaraan misalnya melalui konektor OBD-II atau dengan menyambungkan perangkat secara langsung ke jalur kabel Ethernet dapat menghubungkan perangkat kendali mereka dan memperoleh akses jaringan penuh, serupa dengan serangan penyadapan pada bus CAN (CAN *bus clipping attack*).

Istilah:

- *Trunking* 802.1Q memungkinkan satu tautan Ethernet untuk membawa lalu lintas dari berbagai VLAN secara bersamaan dengan menambahkan tag 4-byte pada setiap *frame*. Tag ini mengidentifikasi VLAN asal *frame* tersebut, sehingga *switch* jaringan dapat memisahkan dan meneruskan lalu lintas ke VLAN yang tepat.
- *Native* VLAN adalah VLAN yang ditetapkan untuk setiap *frame* tanpa tag (*untagged frame*) yang tiba di *port trunk*. Saat *switch* menerima *frame* yang tidak memiliki tag 802.1Q, switch tersebut secara otomatis mengaitkan *frame* tersebut dengan native VLAN.
- *Port trunk* adalah *port switch* yang dikonfigurasi untuk membawa lalu lintas dengan tag dari berbagai VLAN secara bersamaan. *Port* ini biasanya digunakan untuk menghubungkan antar-switch atau menghubungkan *switch* ke *router* maupun *gateway*.
- Lalu lintas tanpa tag (*untagged traffic*) adalah *frame Ethernet* yang tidak membawa tag VLAN. Pada *port trunk*, *frame* tanpa tag secara otomatis ditetapkan ke *native* VLAN.
- *Double-tagging* adalah teknik di mana sebuah *frame Ethernet* dienkapsulasi dengan dua tag VLAN 802.1Q—tag luar dan tag dalam. Tag luar akan dihapus oleh *switch* pertama, sementara tag dalam

kemudian diproses oleh switch kedua. Hal ini memungkinkan frame dikirim ke VLAN yang berbeda dari VLAN yang diizinkan untuk diakses oleh pengirim.
- VLAN *hopping* adalah teknik serangan di mana penyerang mengirimkan *frame* yang dirancang khusus untuk berpindah dari satu VLAN ke VLAN lain tanpa melalui *router* atau *gateway*. Teknik ini memintas kontrol akses antar-VLAN yang seharusnya diterapkan pada batas tersebut.
- IEEE 802.1X adalah standar kontrol akses jaringan berbasis *port* yang mengharuskan perangkat untuk melakukan autentikasi sebelum diberikan akses ke jaringan. Sebelum autentikasi berhasil, *port* akan diblokir dan lalu lintas jaringan tidak diizinkan.
- Konektor OBD-II adalah port standar yang menyediakan akses fisik ke antarmuka diagnostik kendaraan.

Kelemahan Lapisan Atas DoIP

DoIP memungkinkan sesi diagnostik UDS melalui Ethernet, menggantikan ISO-TP berbasis CAN untuk ECU yang terhubung melalui Ethernet. DoIP biasanya dapat diakses pada VLAN diagnostik khusus dan dapat dijangkau melalui konektor OBD-II atau dari jarak jauh melalui unit telematika. Kelemahan yang relevan meliputi hal-hal berikut:

- **Respons identifikasi kendaraan tanpa autentikasi**: Respons Identifikasi Kendaraan (*Vehicle Identification Response*) melalui UDP mengungkapkan Nomor Identifikasi Kendaraan (VIN), Pengidentifikasi Entitas (EID), dan alamat logis dari ECU yang dapat dijangkau kepada host mana pun di jaringan tanpa memerlukan autentikasi sebelumnya. Penyerang dapat memanfaatkan hal ini untuk melakukan tahap pengintaian (*reconnaissance*) guna memperoleh peta lengkap sistem kendaraan target yang dapat diakses.
- **Enumerasi alamat logis:** Ruang alamat logis 16-bit (0x0000–0xFFFF) yang digunakan untuk mengalamati ECU individual atau fungsi diagnostik tergolong cukup kecil sehingga dapat dienumerasi secara menyeluruh. Artinya, penyerang dapat melakukan serangan *brute-force* dengan menelusuri seluruh ruang alamat tersebut, mengirimkan permintaan *TesterPresent* (Layanan 0x3E) ke setiap alamat, lalu mengamati respons yang diterima untuk memetakan seluruh ECU yang dapat diakses.
- **Aktivasi perutean (*Routing Activation*) tanpa autentikasi kriptografis**: Permintaan Aktivasi Perutean (*Routing Activation Request*) mendukung kolom byte autentikasi khusus dari produsen peralatan asli (*Original Equipment Manufacturer* atau OEM), namun implementasinya bersifat opsional dan sering kali tidak ada dalam sistem produksi. Akibatnya, jika fitur ini tidak dikonfigurasi, host mana pun yang dapat menjangkau port TCP 13400 dapat mengaktifkan koneksi perutean dan memulai sesi UDS.

Istilah:

- *Entity Identifier* (EID) adalah pengidentifikasi unik yang diberikan kepada entitas DoIP biasanya berupa simpul jaringan seperti *gateway* atau ECU yang digunakan untuk membedakannya dari entitas lain dalam jaringan kendaraan. EID umumnya diturunkan dari alamat MAC antarmuka Ethernet.

- *Logical address* (alamat logis) adalah pengidentifikasi 16-bit yang diberikan kepada setiap ECU dalam jaringan DoIP, yang digunakan untuk mengarahkan pesan diagnostik ke target yang tepat. Berbeda dengan alamat IP yang mengidentifikasi antarmuka jaringan, alamat logis mengidentifikasi ECU tertentu atau fungsi diagnostik di dalam arsitektur diagnostik kendaraan.
- *Routing Activation Request* (Permintaan Aktivasi Perutean) adalah pesan DoIP yang dikirim oleh klien diagnostik ke *gateway* DoIP untuk membangun koneksi perutean diagnostik melalui sesi TCP yang telah terbentuk sebelumnya. Setelah *gateway* mengonfirmasi aktivasi perutean dengan respons positif, klien diizinkan untuk mengirim pesan diagnostik yang ditujukan kepada ECU tertentu di balik *gateway* tersebut.

Kelemahan Middleware SOME/IP

SOME/IP mendukung komunikasi berbasis layanan (*service-oriented*) dalam arsitektur kendaraan yang terhubung melalui Ethernet. Beberapa kelemahan utamanya meliputi:

- **Penemuan layanan (*service discovery*) SOME/IP-SD tanpa kontrol akses**: SOME/IP *Service Discovery* (SOME/IP-SD) menggunakan multicast UDP untuk mengumumkan dan menemukan layanan. Dengan demikian, setiap node di dalam jaringan dapat mengirim pesan `*FindService*` dan menerima respons `*OfferService*` dari semua layanan yang terlihat. Spesifikasi SOME/IP-SD tidak mencantumkan mekanisme autentikasi maupun otorisasi apa pun. Akibatnya, penyerang yang memiliki akses ke jaringan dapat mendata semua layanan yang tersedia, antarmukanya, serta port yang digunakan, tanpa memerlukan kredensial apa pun.
- **Tidak adanya autentikasi pesan:** Pesan SOME/IP tidak memiliki perlindungan autentikasi bawaan. Konsekuensinya, penyerang dapat menyisipkan pesan permintaan (*request*) atau pesan pemberitahuan (*notification*) palsu yang ditujukan ke layanan mana pun yang telah ditemukan. Karena layanan penerima tidak dapat memverifikasi asal pesan, ia akan memproses pesan tersebut seolah-olah berasal dari pengirim yang sah, yang berpotensi memicu perilaku yang tidak diinginkan pada aplikasi penerima.
- **Pembajakan langganan (*subscription hijacking*)**: Mekanisme *publish-subscribe* tidak melakukan autentikasi terhadap pelanggan (*subscriber*). Oleh karena itu, penyerang dapat mendaftarkan diri sebagai pelanggan untuk aliran data sensor berfrekuensi tinggi seperti umpan kamera atau data LiDAR sehingga mendapatkan akses ke aliran data yang berkelanjutan.
- **Penanganan payload yang tidak sesuai format (*malformed payload*)**: Proses serialisasi pada SOME/IP sering menjadi sumber kerentanan implementasi. Sebagai contoh, kolom panjang (*length field*) yang terlalu besar, kerancuan tipe (*type confusion*) pada kolom varian, serta penyandian *Type-Length-Value* (TLV) yang tidak sesuai format terbukti dapat memicu buffer overflow dan kondisi *denial-of-service* (penolakan layanan) pada stack SOME/IP.

Istilah:

- Pesan Permintaan (*Request message* – Tipe Pesan 0x00) dikirim oleh klien SOME/IP untuk memanggil metode pada layanan jarak jauh (*remote service*) dengan harapan mendapatkan

respons. Ini merupakan mekanisme utama untuk pemanggilan prosedur jarak jauh (*remote procedure call*) dalam komunikasi berbasis layanan SOME/IP.

- Pesan Pemberitahuan (*Notification message* – Tipe Pesan 0x02) dikirim oleh layanan SOME/IP kepada semua pelanggan (*subscriber*) yang terdaftar saat suatu peristiwa terjadi atau nilai data berubah. Pesan ini digunakan dalam pola komunikasi publish-subscribe dan tidak memerlukan permintaan sebelumnya dari penerima.
- *Type confusion* (kerancuan tipe) adalah kerentanan yang terjadi ketika sebuah program memproses data sebagai tipe yang berbeda dari yang seharusnya. Penyerang dapat mendeklarasikan satu tipe data pada pengidentifikasi tipe (*type identifier*), namun menyajikan data dengan struktur tipe yang berbeda. Hal ini menyebabkan implementasi penerima salah menafsirkan data tersebut, yang berpotensi mengakibatkan kerusakan memori (*memory corruption*) atau perilaku program yang tidak terduga.
- *Type-Length-Value* (TLV) adalah format serialisasi data di mana setiap kolom dalam pesan dijelaskan oleh tiga komponen: pengidentifikasi Tipe (*Type*) yang menunjukkan tipe data, nilai Panjang (*Length*) yang menentukan jumlah byte data yang disertakan, serta Nilai (*Value*) yang berisi data aktualnya. Penyerang dapat mengeksploitasi proses penguraian (*parsing*) TLV dengan mendeklarasikan nilai Panjang yang tidak sesuai dengan jumlah byte yang sebenarnya dikirim misalnya, mendeklarasikan 65.536 byte padahal hanya mengirimkan 8 byte yang dapat memicu perilaku yang tidak diinginkan.

Ketiadaan atau Kesalahan Konfigurasi TLS

Berbeda dengan CAN, Ethernet otomotif memiliki lebar pita (*bandwidth*) yang memadai untuk mendukung TLS. Namun, sering kali TLS tidak digunakan, dikonfigurasi dengan cipher suite yang sudah usang, atau diimplementasikan dengan logika validasi sertifikat yang tidak tepat. Beberapa kesalahan konfigurasi yang umum meliputi:

- **Layanan yang dapat diakses melalui HTTP teks biasa (*plaintext*) atau TCP mentah (raw TCP), padahal HTTPS/TLS dapat diterapkan:** Menjalankan layanan diagnostik, konfigurasi, atau pembaruan *Over-the-Air* (OTA) melalui HTTP yang tidak dienkripsi atau TCP mentah akan mengekspos seluruh komunikasi kepada setiap node yang berada dalam segmen jaringan kendaraan yang sama.
- **Diterimanya TLS 1.0 atau 1.1 demi kompatibilitas dengan klien lama (*legacy*)**: TLS 1.0 dan 1.1 adalah versi yang sudah usang dan memiliki kelemahan kriptografis yang diketahui. Jika versi-versi ini tetap didukung karena alasan kompatibilitas sistem lama, penyerang dapat memaksa penurunan versi (*downgrade*) dari TLS 1.2 atau 1.3 ke versi protokol yang lebih lemah tersebut, lalu melancarkan serangan.
- **Penggunaan sertifikat yang ditandatangani sendiri (*self-signed certificates*) tanpa root of trust (akar kepercayaan) atau pemeriksaan pencabutan sertifikat**: Dalam infrastruktur kunci publik (*Public Key Infrastructure* atau PKI) kendaraan, sertifikat untuk *Electronic Control Unit* (ECU) seharusnya diterbitkan oleh *Certificate Authority* (CA) tepercaya, yang memungkinkan pembentukan rantai kepercayaan (*chain of trust*) yang dapat diverifikasi. Sebaliknya, sertifikat yang ditandatangani sendiri tidak memberikan jaminan keamanan semacam itu; selain itu, tanpa adanya pemeriksaan

pencabutan, sertifikat milik ECU yang telah disusupi akan tetap dianggap sah untuk selamanya.

- **Tidak diterapkannya *certificate pinning*, yang membuka celah bagi serangan *man-in-the-middle* dari *node* gerbang (*gateway*) yang telah disusupi**: *Certificate pinning* mengikat suatu ECU pada sertifikat spesifik yang diharapkan dari mitra komunikasinya, sehingga menolak sertifikat lain meskipun sertifikat tersebut secara teknis sah. Tanpa *certificate pinning*, *gateway* yang telah disusupi dapat membangun sesi TLS terpisah dengan dua ECU yang berkomunikasi dengan cara menyajikan sertifikat sah miliknya kepada masing-masing pihak lalu mendekripsi, membaca, dan bahkan memodifikasi seluruh lalu lintas data di antara keduanya, tanpa bisa dideteksi oleh ECU mana pun.
- **Tidak diterapkannya mutual TLS (mTLS), yang memungkinkan klien mana pun untuk terhubung ke titik akhir (*endpoint*) layanan yang sensitif**: TLS standar hanya mengautentikasi server terhadap klien, bukan sebaliknya. Dalam konteks Ethernet otomotif, di mana komunikasi antar-ECU seharusnya dibatasi hanya untuk *node* yang berwenang, ketiadaan mutual TLS (mTLS) berarti penyerang yang berhasil mengakses jaringan kendaraan dapat terhubung ke titik akhir layanan yang sensitif tanpa perlu melakukan autentikasi.

Istilah:

- Sertifikat yang ditandatangani sendiri (*self-signed certificate*) adalah sertifikat digital yang ditandatangani oleh entitas yang sama dengan penerbitnya, bukan oleh Otoritas Sertifikat (*Certificate Authority*) tepercaya. Karena sertifikat tersebut tidak diverifikasi oleh Otoritas Sertifikat tepercaya, pihak penerima tidak memiliki cara untuk memastikan apakah mereka benar-benar berkomunikasi dengan entitas yang dituju.
- Pemeriksaan pencabutan (*revocation checking*) adalah proses untuk memastikan bahwa suatu sertifikat belum dicabut sebelum sertifikat tersebut dipercaya. Sebagai contoh, sebuah sertifikat dapat dicabut jika kunci privat yang terkait telah disusupi, atau jika Otoritas Sertifikat penerbitnya memutuskan bahwa sertifikat tersebut tidak lagi layak dipercaya.

Kelemahan XCP Melalui Ethernet (XCPoE)

Kelemahan umum XCP yang dijelaskan pada bagian CAN juga berlaku untuk XCPoE. Perlu dicatat bahwa jangkauan jaringan Ethernet semakin memperluas permukaan serangan (*attack surface*). Oleh karena itu, terdapat kelemahan tambahan khusus Ethernet untuk XCPoE, antara lain:

- Slave XCP tanpa autentikasi yang dapat diakses melalui jaringan: Slave XCPoE yang dapat diakses pada tulang punggung (*backbone*) Ethernet kendaraan atau dapat dijangkau melalui *gateway* yang salah konfigurasi dari antarmuka telematika maupun OBD memberikan akses penuh untuk membaca/menulis memori ECU kepada host mana pun yang dapat menjangkau port UDP atau TCP tersebut. Dalam konfigurasi seperti ini, prasyarat akses fisik ke bus yang biasanya membatasi eksploitasi pada XCP melalui CAN (XCPoC) tidak lagi berlaku, sehingga eksploitasi jarak jauh menjadi lebih mungkin dilakukan.

- Tidak adanya autentikasi lapisan transpor (*transport-layer*) dalam spesifikasi dasar: Spesifikasi XCPoE tidak mewajibkan penggunaan TLS atau bentuk keamanan lapisan transpor lainnya. Dalam praktiknya, koneksi XCPoE sering kali berjalan melalui UDP atau TCP dalam bentuk teks polos (*plaintext*) tanpa enkripsi maupun autentikasi sesi. Hal ini membuat koneksi tersebut rentan terhadap penyadapan pasif (*eavesdropping*) atas data pengukuran, serta injeksi perintah aktif oleh penyerang yang terhubung ke segmen jaringan yang sama.
- Penemuan slave berbasis siaran (*broadcast*): XCPoE mendukung penemuan slave melalui siaran UDP (UDP *broadcast*), yang memungkinkan penyerang untuk mendata semua ECU yang mendukung XCP di dalam jaringan tanpa perlu mengetahui alamat IP-nya terlebih dahulu. Informasi yang terkumpul memberikan gambaran yang jelas kepada penyerang mengenai target yang tersedia, sebelum mereka melanjutkan upaya untuk melakukan koneksi dan mengakses memori.

Ketentuan:

Dalam XCP, komunikasi mengikuti model *master-slave*. Slave XCP adalah ECU yang mengekspos memori internal serta data kalibrasinya melalui antarmuka XCP. Master XCP adalah klien biasanya berupa perangkat kalibrasi atau bisa juga berupa node yang dikendalikan oleh penyerang yang memulai sesi dan mengeluarkan perintah untuk membaca dari atau menulis ke memori slave tersebut.

Kelemahan Ethernet Umum Lainnya

Selain kelemahan khusus otomotif yang dijelaskan di atas, terdapat beberapa kelemahan protokol Ethernet secara umum yang juga relevan dengan jaringan Ethernet di dalam kendaraan dan patut diperhatikan:

- **ARP *spoofing***: ARP, yang memetakan alamat IP ke alamat MAC dalam jaringan lokal, tidak melakukan autentikasi terhadap balasan yang diterimanya. Perangkat apa pun di dalam jaringan dapat mengirimkan balasan ARP palsu yang mengaitkan alamat MAC miliknya sendiri dengan alamat IP perangkat lain sehingga menyebabkan node lain memperbarui *cache* ARP mereka dengan pemetaan yang salah. Dalam jaringan Ethernet otomotif, hal ini memungkinkan penyerang yang memiliki akses ke jaringan untuk memosisikan diri di antara dua ECU yang sedang berkomunikasi, atau di antara ECU dan sistem diagnostik, sehingga mereka dapat mencegat atau memodifikasi paket data yang sedang dikirim tanpa disadari oleh pihak-pihak yang berkomunikasi.
- **MAC *spoofing***: Jaringan Ethernet mengidentifikasi perangkat berdasarkan alamat MAC-nya, dan mekanisme kontrol akses seperti pemfilteran MAC (MAC *filtering*) bergantung pada pengidentifikasi ini. Karena alamat MAC dapat dikonfigurasi melalui perangkat lunak pada sebagian besar antarmuka jaringan, perangkat apa pun dapat menampilkan alamat MAC sembarang. Dalam jaringan Ethernet otomotif, hal ini memungkinkan perangkat yang tidak sah untuk menyamar sebagai ECU atau perangkat diagnostik yang resmi, yang berpotensi melewati kontrol akses dan mendapatkan akses

ke subsistem kendaraan atau ECU yang krusial bagi keselamatan yang seharusnya dibatasi aksesnya.

- **MAC *table overflow***: *Switch Ethernet* mengelola tabel *Content Addressable Memory* (CAM) yang menyimpan pemetaan antara alamat MAC dan port fisik switch untuk memastikan lalu lintas data hanya diteruskan ke tujuan yang dituju. Ketika tabel tersebut penuh, sebagian besar *switch* akan beralih ke *mode fail-open*, di mana semua *frame* yang masuk akan disiarkan (*broadcast*) ke setiap port. Dalam jaringan Ethernet otomotif, hilangnya isolasi lalu lintas ini berarti komunikasi antar-ECU yang krusial bagi keselamatan yang biasanya terbatas pada segmen jaringan tertentu menjadi terlihat oleh semua perangkat yang terhubung ke switch tersebut, sehingga pesan kontrol dan lalu lintas diagnostik dapat terpapar kepada pihak yang tidak berwenang.

LIN

LIN terutama digunakan untuk fungsi-fungsi dalam domain bodi kendaraan, seperti pengaturan posisi kursi, penyesuaian kaca spion, dan kontrol jendela. Meskipun kecepatan yang terbatas dan area serangan fisik yang sempit membuatnya kurang menonjol dibandingkan CAN atau Ethernet dari perspektif keamanan, LIN menjadi relevan begitu seorang penyerang berhasil mengakses jaringan kendaraan dan beralih menargetkan perangkat yang terhubung melalui LIN.

Kepercayaan Implisit Terhadap Master dan Ketiadaan Autentikasi

Model *master-slave* pada LIN mengasumsikan bahwa master dapat dipercaya secara inheren. Artinya, *node slave* akan merespons header frame yang dikeluarkan oleh master tanpa melakukan verifikasi kriptografis apa pun terhadap identitas master tersebut. Dengan demikian, penyerang yang berhasil mengakses bus LIN baik dengan menyusupi ECU master (yang biasanya terhubung ke jaringan CAN) maupun dengan melakukan penyadapan fisik pada bus kabel Tunggal dapat menyamar sebagai master dan mengirimkan *header frame* arbitrer yang akan direspons oleh *node slave*. Akibatnya, penyerang dapat membaca data status sensor dan aktuator dari perangkat serta periferal LIN yang terhubung.

FlexRay

Meskipun FlexRay merupakan protokol jaringan dalam kendaraan yang bersifat deterministik dan toleran terhadap kesalahan dirancang untuk aplikasi yang krusial bagi keselamatan (*safety-critical*) dan memerlukan ketepatan waktu yang terjamin protokol ini tidak diadopsi secara luas, sehingga memiliki area serangan yang terbatas. Walaupun dirancang untuk aplikasi yang krusial bagi keselamatan, FlexRay memiliki keterbatasan keamanan yang serupa dengan CAN.

Ketiadaan Autentikasi Pesan

Frame FlexRay tidak memuat autentikasi maupun kolom identitas sumber. Oleh karena itu, protokol ini tidak memberikan jaminan bahwa isi pesan berasal dari pengirim yang seharusnya atau bahwa pesan tersebut belum dimanipulasi. Dengan demikian, penyerang yang telah mengakses bus FlexRay dapat memalsukan *frame* dengan muatan (*payload*) arbitrer, yang kemudian akan diproses begitu saja oleh ECU penerima.

Jadwal TDMA Sebagai Sarana Serangan

Jadwal TDMA pada FlexRay ditetapkan secara statis dan dapat diperoleh melalui berkas deskripsi jaringan seperti *Field Bus Exchange* (FIBEX) atau AUTOSAR XML (ARXML) yang diekstrak dari lingkungan pengembangan atau direkonstruksi melalui pemeriksaan lalu lintas pasif dengan mengamati penugasan slot di sepanjang siklus komunikasi yang berhasil ditangkap. Berbekal pengetahuan mengenai jadwal ini, penyerang dapat menentukan waktu penyisipan pesan secara tepat untuk menyasar slot komunikasi tertentu, dengan tujuan menimpa nilai yang sah menggunakan nilai pilihan penyerang, yang pada gilirannya dapat memengaruhi perilaku kendaraan. Selain itu, mengganggu mekanisme sinkronisasi jam FlexRay misalnya dengan mengirimkan bingkai (*frame*) yang memuat nilai penghitung siklus (*cycle counter*) yang tidak tepat dapat menyebabkan *node* dalam klaster menyatakan adanya kesalahan sinkronisasi dan beralih ke status aman (*safe state*). Hal ini mengakibatkan penolakan layanan (*denial of service*) yang mengganggu sistem-sistem krusial bagi keselamatan, seperti sistem kemudi elektronik (*steer-by-wire*) atau sistem pengereman elektronik (*brake-by-wire*).

Pertimbangan Topologi Jaringan

Kelemahan protokol yang dijelaskan pada bagian sebelumnya tidak berdiri sendiri. Kemungkinan eksploitasi secara praktis bergantung pada topologi jaringan tempat protokol tersebut diterapkan, serta batasan kepercayaan (*trust boundaries*) yang diberlakukan atau tidak diberlakukan di antara bagian-bagian jaringan tersebut.

Segmentasi Jaringan yang Lemah dan Kurangnya Penerapan Batasan

Pada arsitektur berbasis domain yang lebih lama dengan segmentasi minimal, beberapa bus CAN dan ECU yang terhubung mungkin berbagi *gateway* dengan kemampuan penyaringan (*filtering*) yang terbatas. Jika *gateway* tersebut tidak menerapkan perutean pesan berbasis daftar putih (*whitelist*) yaitu hanya mengizinkan ID pesan tertentu untuk melintasi batasan domain maka sebuah node yang telah disusupi di satu domain dapat mengirimkan pesan yang merambat melalui *gateway* menuju ECU target di domain lain. Selain itu, tulang punggung (*backbone*) Ethernet yang datar atau tersegmentasi dengan buruk akan memperparah masalah ini secara signifikan. Tanpa penerapan VLAN dan kebijakan perutean antar-VLAN pada *gateway* atau switch pusat, sistem infotainment yang telah disusupi, misalnya, dapat langsung mengirimkan pesan ke ECU ADAS atau pengontrol domain (*domain controller*) yang mengelola lalu lintas CAN pada sistem penggerak (*powertrain*).

Batasan Kepercayaan Pada Gateway dan Pengontrol Domain

Gateway dan pengontrol domain merupakan titik utama penerapan batasan kepercayaan dalam arsitektur jaringan kendaraan. Komponen-komponen ini sangat menentukan apakah kelemahan pada tingkat protokol di segmen bus tertentu dapat dirangkaikan menjadi serangan yang mampu merambah dan menyasar sistem lain di luar titik masuk awalnya. Beberapa hal yang perlu diperhatikan antara lain:

- **Daftar putih (*whitelist*) ID pesan versus perutean langsung (*pass-through*)**: *Gateway* yang meneruskan semua ID pesan CAN antar segmen jaringan tanpa penyaringan tidak memberikan isolasi apa pun antar domain. Untuk mengatasi masalah ini, penggunaan

daftar putih untuk ID pesan yakni hanya mengizinkan ID pesan tertentu untuk diteruskan ke domain tujuan yang dituju menjadi kontrol mendasar guna membatasi injeksi pesan lintas domain.

- **Pembatasan laju (*rate limiting*) dan penyaringan berbasis anomali**: Meskipun daftar putih ID pesan telah diaktifkan, penyerang tetap dapat melakukan tindakan seperti serangan penolakan layanan (*denial-of-service*) dengan mengirimkan ulang lalu lintas yang sah dalam volume dan frekuensi tinggi. Penerapan kemampuan pembatasan laju pada *gateway*, yang dikombinasikan dengan fitur deteksi anomali waktu (*timing anomaly detection*), dapat membantu mengatasi masalah ini.
- **Keamanan penerjemahan Ethernet-ke-CAN**: Dalam arsitektur berbasis zona (*zone architecture*), sebuah ECU zona berfungsi menerjemahkan komunikasi antara tulang punggung Ethernet dan segmen CAN atau LIN lokal. Fungsi translasi merupakan batasan keamanan yang krusial: unit kendali elektronik (ECU) zona yang telah disusupi dapat menyuntikkan lalu lintas ke dalam segmen CAN atau LIN yang terhubung untuk memanipulasi komunikasi di dalam zona tersebut, tanpa diketahui oleh gerbang pusat (*central gateway*). Oleh karena itu, ECU zona harus menerapkan standar keamanan yang setara dengan gerbang pusat.

Istilah:

- *Zone* ECU (ECU zona) adalah simpul jaringan kendaraan yang mengumpulkan komunikasi dari semua ECU di dalam wilayah fisik tertentu pada kendaraan seperti zona depan, belakang, kiri, atau kanan terlepas dari domain fungsionalnya. Zone ECU berfungsi sebagai gerbang lokal (*gateway*) antara segmen CAN atau LIN yang terhubung dengannya dan jaringan tulang punggung (*backbone*) Ethernet pusat.
- *Central gateway* (gerbang pusat) adalah simpul jaringan kendaraan sentral yang merutekan serta menyaring lalu lintas antar segmen atau domain jaringan yang berbeda, seperti *powertrain* (sistem penggerak), sasis, ADAS, dan infotainment. Komponen ini menegakkan batasan kepercayaan (*trust boundaries*) di seluruh arsitektur kendaraan, mengontrol pesan mana yang boleh melintasi batasan segmen, serta menjadi titik utama bagi antarmuka eksternal seperti OBD-II dan komunikasi telematika untuk berinteraksi dengan jaringan internal kendaraan.

Bagaimana Topologi Memperbesar Kelemahan Protokol

Penting untuk dipahami bahwa hubungan antara topologi dan kelemahan protokol bersifat multiplikatif, bukan aditif. Sebagai contoh, bus CAN tanpa autentikasi pesan memiliki risiko terbatas jika diisolasi secara fisik. Namun, bus yang sama dapat menjadi permukaan serangan (*attack surface*) bagi seluruh kendaraan jika dapat diakses dari antarmuka eksternal melalui *gateway* dengan kemampuan penyaringan (*filtering*) yang tidak memadai. Demikian pula, *backbone Ethernet* yang datar mengubah kerentanan pada masing-masing ECU menjadi peluang untuk melakukan pergerakan lateral (*lateral movement*). Penyerang dapat memanfaatkan celah ini untuk berpindah misalnya, dari domain sistem hiburan (*infotainment*) ke domain ADAS atau *powertrain* yang berpotensi menimbulkan risiko keselamatan.

Kelemahan yang ditemukan dalam protokol jaringan di dalam kendaraan mulai dari akses tanpa autentikasi, tidak adanya autentikasi pesan, hingga paparan akibat *gateway* yang salah konfigurasi menjadi landasan untuk memahami bagaimana seorang penguji penetrasi (*penetration tester*) dapat mendekati sistem-sistem tersebut dalam praktik nyata. Bagian selanjutnya akan memaparkan teknik-teknik yang dapat digunakan oleh penguji keamanan untuk mendeteksi kelemahan tersebut pada protokol CAN, Ethernet, LIN, dan FlexRay selama proses penilaian (*assessment*) jaringan di dalam kendaraan.

6.4 TEKNIK PENGUJIAN PENETRASI UNTUK JARINGAN DI DALAM KENDARAAN

Pengujian penetrasi pada jaringan di dalam kendaraan (*in-vehicle networks*) memiliki banyak konsep dasar yang serupa dengan pengujian penetrasi pada jaringan TI pada umumnya. Perbedaannya muncul karena adanya protokol lama (*legacy protocols*) yang mungkin tidak familier bagi penguji penetrasi TI biasa. Agar pengujian penetrasi yang dilakukan efektif, kita harus mengikuti pendekatan terstruktur yang dirancang khusus untuk jaringan di dalam kendaraan. Bagian ini mengelompokkan teknik pengujian berdasarkan protokol mengikuti struktur yang sama dengan bagian sebelumnya sehingga setiap teknik dapat dipahami dalam konteks kelemahan yang ditanganinya.

CAN

Arsitektur siaran (*broadcast*) pada CAN, ketiadaan autentikasi, dan tidak adanya kontrol akses menjadikannya salah satu jaringan di dalam kendaraan yang paling mudah diakses dari perspektif pengujian penetrasi. Begitu akses ke bus diperoleh, berbagai teknik dapat diterapkan. Alat-alat yang umum digunakan untuk pengujian penetrasi pada CAN meliputi python-can, cantools, SavvyCAN, candump, caringcaribou, BUSMASTER™, Scapy, dan Kayak.

Pemantauan Bus Secara Pasif dan Pemeriksaan Lalu Lintas

Pemantauan pasif merupakan langkah awal yang mendasar dalam setiap pengujian penetrasi CAN. Karena seluruh lalu lintas CAN disiarkan ke semua node pada segmen bus, perangkat yang terhubung ke bus baik melalui antarmuka OBD-II, koneksi fisik yang disambungkan langsung, maupun ECU yang telah disusupi akan menerima salinan lengkap dari semua lalu lintas yang ada di bus tersebut. Tujuan utama dari pemantauan pasif adalah enumerasi bus dan rekayasa balik (*reverse engineering*) terhadap muatan data (*payload*). Enumerasi bus mencakup penyambungan ke bus CAN dan pencatatan semua ID pesan yang teramati, frekuensi transmisi, serta *Data Length Code* (DLC), yang kemudian dibandingkan dengan berkas deskripsi jaringan seperti *Database* CAN (DBC) dan A2L/XML untuk mengidentifikasi pesan yang tidak terdokumentasi atau tidak terduga. Untuk ID pesan yang tidak memiliki definisi dalam berkas DBC, rekayasa balik terhadap muatan data dapat dilakukan dengan menerapkan berbagai kendali fisik seperti mengoperasikan kemudi, pedal gas, rem, dan sebagainya sembari memantau perubahan pada muatan data di dalam bus.

Injeksi Pesan dan Pemalsuan (Spoofing)

Karena CAN tidak memiliki mekanisme autentikasi sumber, setiap *node* pada bus dapat mengirimkan pesan dengan ID apa pun beserta muatan (*payload*) apa pun. ECU penerima tidak memiliki cara untuk membedakan antara frame yang sah dengan *frame* yang diinjeksikan

atau dipalsukan. Injeksi sinyal yang ditargetkan dilakukan dengan mengidentifikasi sinyal sasaran seperti sinyal kecepatan kendaraan atau perintah untuk membuka kunci pintu melalui pemantauan pasif atau pemeriksaan file CAN DBC. Setelah itu, penyerang menyusun dan mengirimkan *frame* dengan ID pesan yang sesuai serta nilai muatan yang diinginkan. Untuk memanipulasi status sistem secara berkelanjutan, penyerang dapat menginjeksikan pesan target dengan frekuensi yang lebih tinggi daripada frekuensi pengiriman oleh pengirim yang sah. Akibatnya, ECU penerima akan terus-menerus memproses muatan dari penyerang, alih-alih memproses data yang sah.

Basis data yang dikelola oleh komunitas, seperti opendbc, menyediakan berkas DBC yang telah didekode untuk berbagai model kendaraan produksi, sehingga secara signifikan mengurangi upaya yang diperlukan untuk memetakan lalu lintas CAN yang tertangkap menjadi sinyal-sinyal yang bermakna. Perlu dicatat bahwa korelasi sinyal dengan bantuan kecerdasan buatan (AI) yang diterapkan pada berbagai pesan yang ditangkap dari bus data yang sedang aktif dapat semakin mempercepat proses rekayasa balik (*reverse engineering*) terhadap sinyal-sinyal yang belum tercakup dalam basis data yang ada.

Serangan Replay (Pengulangan)

Serangan *replay* memanfaatkan ketiadaan perlindungan terhadap kebaruan (*freshness protection*) pada pesan-pesan CAN. Seorang penguji penetrasi (*pentester*) dapat menangkap urutan *frame* yang sah terkait dengan fungsi kendaraan tertentu seperti urutan pembukaan kunci kendaraan tanpa kunci fisik (*remote keyless entry*) atau inisiasi sesi diagnostic kemudian mengirimkannya kembali di lain waktu untuk memicu fungsi yang sama. Selain itu, *replay* parsial terhadap urutan yang terdiri dari beberapa pesan dapat menyebabkan ECU target berada dalam kondisi perantara yang tidak terdefinisi; hal ini terjadi karena *state machine* (mesin status) terus berjalan tanpa menyelesaikan transisi yang seharusnya, yang kemudian memicu perilaku tidak terduga atau potensi penolakan layanan (*denial of service*). Teknik ini sangat efektif untuk menyasar transisi sesi diagnostik UDS serta fungsi-fungsi kontrol bodi kendaraan yang bergantung pada urutan aktivasi yang terdiri dari beberapa langkah.

Fuzzing Pada Bus CAN

Fuzzing menyasar mekanisme validasi input dan penanganan kesalahan pada tumpukan (*stack*) CAN di ECU dengan cara mengirimkan *payload* yang tidak sesuai format secara sistematis. Tujuannya adalah untuk mengidentifikasi *frame* yang memicu perilaku tak terduga, seperti kerusakan sistem (*crash*), korupsi memori, atau perubahan status yang tidak diinginkan. Terdapat berbagai pendekatan dalam melakukan *fuzzing*. Sebagai contoh, random *fuzzing* mengirimkan *frame* dengan *byte payload* yang diacak; metode ini dapat mendeteksi ECU yang memproses frame tanpa melakukan validasi rentang nilai yang memadai. Teknik ini tergolong mudah, namun sering kali memiliki efisiensi yang rendah.

Di sisi lain, *mutation-based fuzzing* dimulai dari *payload* yang sudah diketahui valid, kemudian menerapkan perubahan sistematis seperti pembalikan *bit* (*bit-flip*) atau penggantian byte (*byte-substitution*). Pendekatan ini jauh lebih efisien karena *frame* hasil mutasi biasanya lolos dari pemeriksaan validasi awal dan dapat menjangkau logika pemrosesan ECU yang lebih dalam, tempat di mana cacat sistem sering kali ditemukan. Selain itu, *generation-based fuzzing*

menyusun *frame* berdasarkan model protokol target atau format pesan tertentu. Hal ini memungkinkan alat *fuzzing* untuk menghasilkan input yang secara sintaksis valid namun secara semantik keliru. Pendekatan ini memaksimalkan cakupan kode (*code coverage*) serta meningkatkan kemungkinan munculnya perilaku tak terduga akibat cacat pada tingkat implementasi di dalam tumpukan CAN atau logika aplikasi. Informasi lebih lanjut mengenai *fuzzing* pada bus CAN dapat ditemukan dalam.

Serangan Bus-Off

Tujuan dari pengujian serangan *bus-off* adalah untuk menentukan apakah ECU target menerapkan pemulihan kesalahan (*error recovery*) dan logika plausibilitas yang memadai agar tetap tangguh terhadap serangan penolakan layanan (*denial-of-service*) yang menyasar mekanisme *bus-off*. Serangan *bus-off* memanfaatkan mekanisme penanganan kesalahan pada CAN untuk memaksa ECU target masuk ke dalam *status bus-off*, yang secara efektif mengeluarkannya dari jaringan dan menyebabkan penolakan layanan (*denial of service*). Serangan ini dilakukan dengan mengirimkan bit dominan selama slot konfirmasi (*acknowledgment slot*) dari *frame* yang sedang dikirim oleh ECU target. Hal ini menyebabkan ECU tersebut menginterpretasikan interferensi tersebut sebagai kesalahan transmisi dan meningkatkan penghitung kesalahannya, yaitu TEC (*Transmit Error Counter*). Jika tindakan ini diulang dengan frekuensi yang cukup, nilai TEC akan melampaui 255, yang membuat node target beralih ke status *Bus-Off.* Keberhasilan serangan *Bus-Off* terhadap *node* yang krusial bagi keselamatan seperti pengontrol ABS atau kantung udara (*airbag*) dapat berdampak langsung pada aspek keselamatan. Oleh karena itu, serangan semacam ini harus dipertimbangkan sebagai bagian dari lingkup pengujian penetrasi (*penetration testing*).

Pengujian ISO-TP

Pengujian ISO-TP berfokus pada logika segmentasi dan penyusunan ulang (*reassembly*) pada lapisan transport. Tujuan dari pengujian ini adalah untuk menemukan kesalahan implementasi serta kerentanan terhadap input yang tidak sesuai format atau tidak terduga. Sebagai contoh, pengujian kehabisan buffer (*buffer exhaustion*) dapat dilakukan dengan mengirimkan *First Frame* yang menyatakan panjang pesan maksimum yang didukung, namun tidak diikuti dengan pengiriman *consecutive frame* (bingkai lanjutan). Penguji penetrasi (*pentester*) dapat mengulangi aktivitas ini dengan berbagai ID pesan sumber, lalu memantau ECU target untuk melihat apakah terjadi penurunan waktu respons atau kegagalan dalam menangani permintaan sah berikutnya.

Selain itu, penguji penetrasi dapat melakukan injeksi urutan *consecutive frame* yaitu menyisipkan bingkai dengan nilai penghitung (*counter*) yang tidak sesuai urutan saat transmisi multi-bingkai sedang berlangsung antara dua node yang sah. Penguji kemudian mengamati apakah ECU target membatalkan sesi, memunculkan pesan kesalahan, atau justru memproses bingkai yang disisipkan tersebut sebagai bagian dari penyusunan ulang pesan yang akan menyebabkan kerusakan pada keseluruhan pesan.

Pengujian Protokol Transport SAE J1939

Pengujian SAE J1939 berfokus pada logika segmentasi dan penyusunan ulang pada lapisan transport J1939. Pengujian kehabisan buffer dapat dilakukan dengan mengirimkan

bingkai TP.CM_BAM yang dirancang khusus untuk menyatakan ukuran pesan yang sangat besar, tanpa diikuti oleh paket data yang dijanjikan. Selanjutnya, ECU target dipantau untuk mendeteksi gejala kehabisan memori, seperti penurunan waktu respons atau hilangnya partisipasi pada bus komunikasi. Untuk sesi berbasis koneksi (*connection-mode*), penguji penetrasi dapat mengirimkan bingkai TP.CM_RTS yang berisi jumlah paket di luar rentang yang diizinkan ke ECU target, lalu mengamati apakah ECU tersebut melakukan pemeriksaan batas (*bounds checking*) atau justru mengalokasikan memori di luar batas yang seharusnya. Selain itu, karena J1939 tidak menyediakan autentikasi alamat sumber, penguji penetrasi dapat mengirimkan bingkai palsu (*spoofed frames*) dengan menggunakan alamat sumber milik *node* yang sah, lalu memverifikasi apakah ECU target menerima dan memproses bingkai tersebut.

Eksploitasi Diagnostik UDS

Tujuan bagi penguji penetrasi (*pentester*) adalah untuk meningkatkan hak akses sesi diagnostik, mengakses layanan yang dilindungi, serta menjalankan kemampuan penulisan atau pengendalian yang seharusnya dibatasi hanya untuk perangkat diagnostik resmi. Seorang penguji penetrasi dapat melakukan jenis pengujian berikut:

- **Enumerasi layanan**: Mengirimkan permintaan untuk setiap dari 256 kemungkinan ID layanan UDS (0x00–0xFF) dalam berbagai sesi diagnostik dan mencatat respons yang diterima. Respons positif (0x40 + ID layanan) mengonfirmasi ketersediaan layanan; respons negatif dengan *Negative Response Code* (NRC) 0x11 (*serviceNotSupported*) menunjukkan bahwa layanan tersebut tidak diimplementasikan; sedangkan NRC 0x7F (*requestCorrectlyReceivedResponsePending*) menunjukkan bahwa layanan tersebut ada, namun memerlukan status sesi tertentu. Berdasarkan respons yang diperoleh, matriks ketersediaan layanan dapat disusun untuk semua jenis sesi yang dapat diakses.
- **Peningkatan sesi (*session escalation*) tanpa autentikasi**: Mencoba melakukan transisi sesi menggunakan *DiagnosticSessionControl* (Layanan 0x10) ke *Extended Diagnostic Session* (byte sub-fungsi 0x03) dan *Programming Session* (byte sub-fungsi 0x02) tanpa melalui prosedur *Security Access* terlebih dahulu. Penguji harus mendokumentasikan jenis sesi apa saja yang dapat diakses tanpa autentikasi serta melakukan enumerasi terhadap layanan tambahan yang tersedia pada setiap sesi yang telah ditingkatkan tersebut.
- **Menyerang *Security Access* (Layanan 0x27)**: Meminta beberapa *seed* (nilai awal) dari ECU target menggunakan *SecurityAccess RequestSeed* (Layanan 0x27, byte sub-fungsi 0x01) dan memeriksa nilai *seed* yang terkumpul untuk mencari karakteristik pembuatan seed yang lemah. Jika *seed* bersifat tetap atau memiliki entropi rendah, respons kunci (*key response*) valid yang pernah ditangkap sebelumnya dapat digunakan kembali (*replay*) untuk melewati mekanisme *challenge-response*. Jika pembuatan seed tampak aman, target dapat dialihkan ke algoritme pembuatan kunci (*key generation algorithm*). Sebagai contoh, penguji penetrasi mungkin dapat menentukan hubungan antara *seed* dan kunci dengan memeriksa pasangan *seed*-kunci valid yang telah dikumpulkan, sehingga kunci yang tepat dapat dihitung untuk setiap *seed* yang diterima. Jika algoritme pembuatan kunci tidak dapat ditentukan hanya melalui

observasi, penguji penetrasi dapat mencoba mengekstraknya dari *firmware* ECU atau perangkat diagnostik, lalu melakukan rekayasa balik (*reverse engineering*) terhadap algoritme tersebut.

- **Enumerasi pengidentifikasi data (Layanan 0x22/0x2E)**: Lakukan enumerasi terhadap seluruh 65.536 kemungkinan Pengidentifikasi Data (*Data Identifier*/DID) menggunakan *ReadDataByIdentifier* (Layanan 0x22). Dokumentasikan DID mana saja yang mengembalikan data, panjang data tersebut, serta isinya. Selanjutnya, cobalah *WriteDataByIdentifier* (Layanan 0x2E) pada DID yang dapat ditulis dalam sesi dengan hak akses khusus (*privileged sessions*) untuk mengevaluasi kontrol akses penulisan dan mengidentifikasi parameter konfigurasi apa saja yang dapat diubah.
- **Pembacaan/penulisan memori berdasarkan alamat (Layanan 0x23/0x3D)**: Lakukan pengujian menggunakan ReadMemoryByAddress (Layanan 0x23) dengan berbagai alamat dasar (*base address*) dan panjang memori untuk mengidentifikasi wilayah memori yang dapat diakses, termasuk *Random Access Memory* (RAM), memori *flash*, dan register periferal yang dipetakan. Jika didukung, cobalah *WriteMemoryByAddress* (Layanan 0x3D) dalam sesi dengan hak akses khusus untuk menilai apakah penulisan ke memori secara arbitrer (sewenang-wenang) dapat dilakukan. Eksploitasi yang berhasil terhadap layanan-layanan ini dapat memungkinkan modifikasi kode saat program berjalan (*runtime*), manipulasi konfigurasi, atau ekstraksi data sensitif seperti kunci kriptografi dan parameter konfigurasi.
- **Enumerasi *RoutineControl* (Layanan 0x31)**: Lakukan enumerasi terhadap pengidentifikasi rutin (*routine identifier*) menggunakan *RoutineControl StartRoutine* (Layanan 0x31, byte sub-fungsi 0x01). Identifikasi rutin-rutin yang memicu perilaku kendaraan yang dapat diamati, memungkinkan penghapusan kode kesalahan (*fault code*), mengubah konfigurasi ECU, atau memengaruhi fungsi keselamatan.
- **Reset ECU (Layanan 0x11)**: Lakukan pengujian reset ECU menggunakan *soft reset* (byte sub-fungsi 0x01), *hard reset* (byte sub-fungsi 0x03), dan keyOffOnReset (byte sub-fungsi 0x02) pada semua jenis sesi yang dapat diakses. Amati hasilnya dan tentukan jenis reset mana yang dapat dijalankan tanpa memerlukan *Security Access* (akses keamanan). Jika memungkinkan, pengujian ini harus dilakukan dalam kondisi yang menyerupai operasional kendaraan seperti saat perangkat lunak sedang berjalan aktif atau dalam simulasi kondisi berkendara karena reset yang berhasil dalam situasi tersebut harus dianggap sebagai temuan yang relevan dengan aspek keselamatan. Perlu dicatat bahwa penyerang dapat mengeksploitasi kemampuan reset yang tidak memerlukan autentikasi sebagai bagian dari serangan *denial-of-service* (penolakan layanan) atau untuk menghapus status sesi diagnostik selama rangkaian serangan berlangsung.

Pengujian XCP di CAN

Pengujian XCP pada CAN menyasar antarmuka kalibrasi dan pengukuran pada ECU yang belum menonaktifkan XCP. Penguji penetrasi (*pentester*) dapat memulai dengan mengidentifikasi CAN ID dari slave XCP melalui pemindaian untuk mencari respons positif

terhadap perintah *CONNECT*. Setelah terhubung, perintah *SHORT_UPLOAD* dapat digunakan untuk membaca wilayah memori tertentu yang ditargetkan, termasuk parameter kalibrasi, wilayah stack, dan register periferal yang dipetakan ke dalam memori (*memory-mapped peripheral registers*). Selain itu, perintah SHORT_DOWNLOAD (untuk penulisan terarah dalam satu *frame*) atau DOWNLOAD (untuk penulisan blok data yang lebih besar) dapat digunakan untuk menulis nilai yang telah dimodifikasi ke dalam parameter kalibrasi.

Jika terdapat mekanisme perlindungan seed-key yang dicurigai diimplementasikan dengan lemah, penguji dapat mengirimkan beberapa permintaan GET_SEED, lalu memeriksa nilai seed yang terkumpul untuk mencari pola-pola yang lemah. Lebih lanjut, untuk menilai eksposur DAQ (*Data Acquisition*), penguji dapat mengeluarkan perintah GET_DAQ_LIST_INFO guna mencantumkan daftar DAQ yang telah dikonfigurasi beserta variabel pengukuran yang terkait. Setelah itu, penguji dapat mengaktifkan sesi DAQ menggunakan perintah START_STOP_DAQ_LIST untuk memastikan apakah aliran data berkelanjutan (*continuous streaming*) dari variabel internal ECU dapat diakses tanpa memerlukan autentikasi.

Pengujian Bypass SecOC

Pengujian bypass SecOC menilai apakah implementasi SecOC dalam sistem target memberikan perlindungan yang efektif terhadap pemalsuan dan pengulangan (*replay*) pesan. Pertama, melalui pemantauan pasif, seorang penguji penetrasi (*pentester*) dapat mengidentifikasi PDU yang dilindungi SecOC dengan mengamati data autentikasi berukuran tetap yang ditambahkan pada payload, serta menentukan panjang bit MAC yang digunakan. Untuk MAC berukuran pendek khususnya yang dipotong menjadi 24 bit penguji dapat menilai kelayakan pemalsuan melalui metode *brute-force* dengan mengestimasi jumlah upaya injeksi yang diperlukan untuk secara statistik menghasilkan tebakan MAC yang valid. Dengan ruang MAC 24-bit yang memiliki sekitar 16 juta kemungkinan nilai, laju transmisi pesan yang teramati akan menentukan seberapa cepat ambang batas ini dapat dicapai. Selain itu, perilaku nilai kesegaran (*freshness value*) dapat dinilai dengan memantau peningkatan penghitung (*counter*) pada bingkai (*frame*) yang berurutan maupun di seluruh siklus penyalaan (*ignition cycle*). Sebagai contoh, jika penghitung diatur ulang (*reset*) saat siklus daya dimulai, bingkai terautentikasi yang telah ditangkap sebelumnya mungkin dapat diulang (*replay*) pada siklus berikutnya, sebelum penghitung tersebut melampaui nilai yang telah ditangkap sebelumnya.

Selain itu, seorang pentester juga dapat memeriksa bagaimana ECU target menangani hilangnya sinkronisasi pada *freshness value* (nilai kesegaran). Sebagai contoh, sebuah ECU mungkin dikonfigurasi untuk hanya mengirimkan sebagian dari *freshness counter* misalnya 8 bit yang paling tidak signifikan (*least significant bits*) dari nilai 64-bit sehingga penerima harus merekonstruksi bit-bit sisanya berdasarkan *counter internal* yang lengkap. Dengan memutuskan koneksi ECU dari jaringan lalu menghubungkannya kembali, pentester dapat mengamati proses sinkronisasi ulang dan menentukan apakah implementasi tersebut menerapkan batasan ketat terhadap rentang *counter* yang dapat diterima, atau apakah pertukaran data tersebut dapat dimanipulasi untuk memungkinkan serangan *replay* (pengulangan pesan) maupun kehabisan *counter* sebelum waktunya. Lebih lanjut, untuk menguji kemungkinan *bypass* (pintasan) pada mekanisme verifikasi, pentester dapat

menyuntikkan PDU yang dilindungi oleh SecOC dengan *payload* yang *valid* namun memiliki MAC yang salah, lalu mengamati apakah aplikasi target tetap memproses pesan tersebut meskipun verifikasi gagal. Terakhir, pentester dapat memeriksa *firmware* ECU dan berpotensi mengekstrak kunci simetris yang digunakan untuk SecOC.

Ethernet Otomotif

Pengujian penetrasi (*penetration testing*) pada Ethernet otomotif mencakup lapisan jaringan, protokol transport dan diagnostik (DoIP, SOME/IP), serta lapisan keamanan (TLS). Berbeda dengan CAN, pengujian berbasis Ethernet dapat memanfaatkan perangkat keamanan TI yang lebih matang yang dapat diadaptasi untuk konteks otomotif, meskipun sifat eksklusif dari beberapa protokol Ethernet otomotif (khususnya SOME/IP) memerlukan perangkat khusus. Perangkat yang umum digunakan antara lain Wireshark, Nmap, tcpdump, Scapy, sslyze, mitmproxy, dan Ettercap.

Penemuan Jaringan dan Penangkapan Lalu Lintas Data

Langkah pertama dalam pengujian Ethernet otomotif adalah membangun gambaran lengkap mengenai topologi jaringan, yaitu menentukan ECU mana saja yang dapat diakses, pada VLAN apa, menggunakan alamat IP berapa, serta layanan apa saja yang terbuka. Penguji penetrasi (*pentester*) akan terhubung ke jaringan Ethernet otomotif melalui port monitor pada *switch* atau langsung melalui antarmuka Ethernet OBD-II, lalu menangkap seluruh lalu lintas data menggunakan Wireshark yang dilengkapi dengan dissector protokol otomotif untuk DoIP, SOME/IP, dan XCP. Hal ini bertujuan untuk mengidentifikasi semua alamat IP yang aktif, tag VLAN, serta jenis protokol yang ada di dalam jaringan. Selain itu, enumerasi VLAN dilakukan dengan memeriksa tag VLAN yang ditemukan dalam lalu lintas data yang tertangkap, guna mengidentifikasi VLAN apa saja yang digunakan di berbagai domain fungsional.

Untuk menguji ketidaksesuaian VLAN asli (*native* VLAN), seorang pentester dapat mengirimkan frame tanpa tag dari port yang terhubung, lalu mengamati apakah frame tersebut diterima pada VLAN yang tidak semestinya di sisi lain port trunk. Jika hal ini terjadi, maka terkonfirmasi adanya kesalahan konfigurasi yang memungkinkan VLAN *hopping* di mana lalu lintas melintasi batas VLAN tanpa melalui *gateway*. Selain itu, pentester dapat memeriksa apakah antarmuka manajemen dapat diakses dari VLAN dengan tingkat kepercayaan lebih rendah, dengan cara mencoba terhubung ke layanan manajemen switch atau *gateway* dari *domain infotainment* atau telematika. Jika akses fisik ke port Ethernet internal termasuk dalam lingkup pengujian, menghubungkan perangkat yang tidak sah ke port switch yang tidak terlindungi dapat mengonfirmasi tidak adanya kontrol akses berbasis port IEEE 802.1X; kondisi ini akan memberikan pentester akses jaringan secara langsung tanpa perlu autentikasi.

Enumerasi dan Eksploitasi DoIP

Seorang penguji penetrasi (*pentester*) dapat memulai dengan mengirimkan *Vehicle Identification Request* (Permintaan Identifikasi Kendaraan) DoIP ke alamat siaran (*broadcast address*) pada port UDP 13400, lalu mengumpulkan pesan *Vehicle Identification Response* (Respons Identifikasi Kendaraan) untuk memperoleh VIN, EID, dan alamat logis dari ECU yang dapat dijangkau. Proses ini tidak memerlukan autentikasi dan mampu memberikan peta lengkap mengenai target yang dapat diakses melalui DoIP. Jika *Vehicle Identification Response*

tidak diterima atau informasi alamat yang diberikan tidak lengkap, enumerasi alamat logis dapat dilakukan dengan metode *brute-force* melalui ruang alamat 16-bit. Caranya adalah dengan mengirimkan permintaan UDS TesterPresent ke setiap alamat, kemudian mengidentifikasi ECU yang aktif berdasarkan respons konfirmasi positif yang diterima.

Untuk menguji apakah aktivasi perutean (*routing activation*) dapat dilakukan tanpa autentikasi, penguji penetrasi dapat membangun koneksi ke port TCP 13400 pada *gateway* DoIP, mengirimkan *Routing Activation Request* (Permintaan Aktivasi Perutean) tanpa menyertakan data autentikasi khusus OEM, lalu memeriksa apakah *gateway* memberikan respons aktivasi yang sukses. Setelah perutean diaktifkan, teknik eksploitasi diagnostik UDS seperti yang telah dijelaskan sebelumnya untuk protokol CAN dapat diterapkan pula melalui DoIP.

Pengujian SOME/IP

Penguji penetrasi dapat mengirimkan pesan *FindService* (bagian dari *service discovery*) ke alamat multicast SOME/IP-SD pada port UDP 30490, lalu menangkap semua respons *OfferService* untuk menyusun inventaris lengkap layanan yang tersedia. Aktivitas ini tidak memerlukan autentikasi. Untuk menguji injeksi pesan, penguji penetrasi dapat mengirimkan pesan permintaan (*request*) palsu ke layanan yang telah ditemukan, atau mengirimkan pesan notifikasi (*notification*) palsu dengan menyamar sebagai penyedia layanan kepada para pelanggannya, kemudian mengamati apakah ECU penerima memproses pesan yang disuntikkan tersebut sebagai pesan yang sah. Selain itu, untuk menguji kemungkinan pembajakan langganan (*subscription hijacking*), penguji penetrasi dapat mengirimkan pesan *SubscribeEventgroup* yang menargetkan kelompok peristiwa (*event group*) penting seperti aliran data kamera atau data LiDAR dengan mencantumkan alamat IP penguji sebagai titik akhir pelanggan (*subscriber endpoint*).

Selanjutnya, penguji dapat memastikan apakah layanan target mulai mengirimkan notifikasi peristiwa tanpa memerlukan autentikasi apa pun. Terakhir, *fuzzing* layanan SOME/IP dapat dilakukan dengan mengirimkan pesan permintaan ke setiap layanan yang ditemukan, yang di dalamnya terdapat kolom *header* yang tidak sesuai format seperti nilai panjang (*length*) yang terlalu besar, jenis pesan yang tidak *valid*, atau penyandian TLV yang rusak kemudian memantau ECU target untuk melihat apakah terjadi *crash*, respons kesalahan, atau perilaku yang tidak terduga.

Pemeriksaan Konfigurasi TLS

Pengujian TLS pada *Automotive Ethernet* bertujuan untuk menilai apakah layanan yang seharusnya dilindungi oleh TLS benar-benar menerapkan perlindungan tersebut, serta apakah konfigurasi TLS yang digunakan telah memenuhi standar keamanan terkini. Melalui penangkapan data hasil penemuan jaringan (*network discovery*), seorang pentester dapat mengidentifikasi layanan TCP yang tidak menggunakan TLS, yang mungkin membawa informasi sensitif seperti data diagnostik, konfigurasi, atau pembaruan OTA dalam bentuk teks polos (*plaintext*). Bahkan jika TLS telah diaktifkan, versi protokol dan *cipher suite* yang didukung dapat diidentifikasi dengan mengirimkan pesan ClientHello. Langkah ini

memungkinkan deteksi terhadap versi protokol yang sudah usang, seperti TLS 1.0 atau 1.1, yang rentan terhadap serangan penurunan versi (*downgrade attack*).

Selain itu, untuk mengevaluasi validasi sertifikat, pentester dapat menyajikan sertifikat yang ditandatangani sendiri (*self-signed*) atau sertifikat yang telah kedaluwarsa kepada ECU target. Dengan mengamati apakah ECU target menolak koneksi tersebut atau justru menyelesaikan handshake, pentester dapat memastikan apakah ECU tersebut menerapkan validasi sertifikat dengan benar. Lebih lanjut, ketiadaan *certificate pinning* dapat diuji melalui *gateway node* yang telah dikompromikan dan berada dalam jalur lalu lintas data; *gateway* ini dapat membangun sesi TLS terpisah dengan setiap ECU yang berkomunikasi dengan cara menyajikan sertifikatnya sendiri kepada masing-masing pihak. Jika *handshake* berhasil diselesaikan tanpa ada ECU yang menolak sertifikat pengganti tersebut, maka dapat dipastikan bahwa *certificate pinning* tidak diterapkan. Hal ini memberikan akses penuh kepada pentester untuk melihat komunikasi yang dicegat dalam bentuk teks polos. Terakhir, ketiadaan mTLS (*mutual* TLS) dapat dikonfirmasi dengan mencoba terhubung ke ECU tanpa menyajikan sertifikat klien, lalu mengamati apakah koneksi tersebut diterima atau tidak.

Pengujian XCP Melalui Ethernet

Seorang pentester dapat mengirimkan siaran (*broadcast*) XCP `GET_SLAVE_ID` melalui *port* UDP 17525 untuk mendata semua ECU yang mendukung protokol XCP di dalam jaringan beserta detail *endpoint transport*-nya, tanpa memerlukan pengetahuan awal mengenai alamat IP target. Untuk ECU yang tidak merespons penemuan melalui broadcast, pemindaian *port* (*port scan*) yang ditargetkan pada port UDP dan TCP 17525 di rentang alamat IP yang ditemukan dapat mengungkap perangkat *slave* lainnya. Setelah perangkat slave teridentifikasi, sesi TCP atau UDP dapat dibangun menggunakan perintah `*CONNECT*` untuk mendata kapabilitas perangkat tersebut. Jika keamanan pada lapisan transport (*transport-layer security*) tidak diterapkan, komunikasi akan berlangsung dalam bentuk teks polos, sehingga memungkinkan pentester untuk melakukan penyadapan pasif (*passive eavesdropping*) guna menangkap data pengukuran. Penguji penetrasi (*pentester*) juga dapat menyuntikkan perintah untuk membaca dan menulis memori menggunakan SHO™T_UPLOAD, SHO™T_DOWNLOAD, dan DOWNLOAD, sebagaimana dijelaskan dalam bagian pengujian XCP berbasis CAN. Perlu dicatat bahwa melalui Ethernet, batasan payload CAN sebesar 8 byte tidak lagi menjadi kendala; dengan demikian, transfer data yang lebih besar dalam satu transaksi dapat dilakukan, sehingga proses pembacaan dan penulisan memori dalam jumlah besar menjadi jauh lebih cepat dan praktis dibandingkan jika dilakukan melalui CAN.

LIN

Pengujian penetrasi (*penetration testing*) pada LIN dibatasi oleh persyaratan akses fisik serta permukaan serangan (*attack surface*) protokol yang sempit dibandingkan dengan CAN dan Ethernet. Teknik-teknik di bawah ini mengasumsikan adanya akses fisik langsung ke bus LIN satu kabel (*single-wire*) melalui adaptor LIN-ke-USB atau LIN-ke-CAN, atau mengasumsikan bahwa ECU master LIN telah dikompromikan melalui pergerakan lateral (*lateral movement*) dari jaringan CAN atau Ethernet di mana keamanan *node* slave yang terhubung ke LIN kemudian dapat dinilai.

Pemantauan Pasif dan Pemeriksaan Frame

Seorang penguji penetrasi (*pentester*) dapat menangkap semua frame LIN untuk memeriksa jadwal *frame*, urutan *Protected Identifier* (PID) yang dikeluarkan oleh master, serta respons yang diberikan oleh slave. Hal ini memungkinkan kita untuk menyusun peta PID yang aktif, panjang data responsnya, dan waktu transmisinya, serta mengorelasikan PID yang teramati dengan perilaku kendaraan guna mengidentifikasi *frame* yang mengendalikan fungsi bodi kendaraan.

Penyamaran Sebagai Master dan Injeksi Frame

Karena slave pada LIN merespons setiap *header fram*e yang valid tanpa melakukan autentikasi terhadap identitas master, seorang pentester dapat menyamar sebagai ECU master dan menerima respons dari slave mana pun dengan mengirimkan header frame untuk PID target yang telah diidentifikasi selama pemantauan pasif. Selain itu, untuk PID di mana slave biasanya memberikan respons kepada master, pentester dapat menyuntikkan respons palsu (*spoofed response*) sebelum slave yang sah sempat merespons. Mengingat LIN menggunakan bus satu kabel tanpa mekanisme deteksi tabrakan (*collision detection*), respons pertama yang muncul di jalur bus akan dianggap sebagai respons yang sah; hal ini memungkinkan pentester untuk menekan (*suppress*) respons dari slave yang sah dengan menyisipkan *payload* yang dikendalikan oleh penyerang.

FlexRay

Pengujian penetrasi pada FlexRay merupakan yang paling khusus di antara keempat protokol yang dibahas dalam bab ini. Jadwal berbasis TDMA, konfigurasi kanal ganda (*dual-channel*), serta persyaratan untuk berpartisipasi dengan sinkronisasi siklus (*cycle-synchronized*) membuat pengujian aktif pada FlexRay jauh lebih kompleks dibandingkan dengan CAN, Ethernet, atau LIN.

Pemeriksaan Trafik Pasif dan Rekonstruksi Jadwal

Pengujian FlexRay dimulai dengan menangkap trafik secara pasif dengan menghubungkan perangkat yang mendukung FlexRay ke salah satu atau kedua kanal dalam klaster FlexRay, baik melalui penyadapan langsung (*direct tap*) pada bus maupun melalui *gateway* yang menyediakan akses pemantauan FlexRay. Menangkap lalu lintas data selama beberapa siklus komunikasi yang memadai memungkinkan penguji penetrasi (*pentester*) untuk merekonstruksi penempatan slot TDMA statis yaitu, menentukan node mana yang melakukan transmisi pada slot tertentu serta memetakan struktur muatan (*payload*) dari setiap slot. Jika tersedia berkas deskripsi jaringan dalam format FIBEX atau ARXML, berkas tersebut dapat dimuat ke dalam perangkat pemeriksaan untuk mendekode nilai sinyal secara langsung. Namun, apabila berkas deskripsi jaringan tidak tersedia, penguji penetrasi dapat mengorelasikan muatan slot dengan perilaku kendaraan yang dapat diamati, dengan menggunakan pendekatan rekayasa balik (*reverse engineering*) sinyal yang serupa dengan metode yang diterapkan pada protokol CAN.

Injeksi Frame dan Manipulasi Waktu

Injeksi *frame* FlexRay aktif memerlukan partisipasi dalam klaster sebagai *node* yang tersinkronisasi. Hal ini dapat dilakukan dengan mengintegrasikan antarmuka perangkat keras

yang mendukung FlexRay yang mampu bergabung dengan jadwal komunikasi klister atau dengan menyusupi *node* yang sudah ada di dalam klaster dan beroperasi melalui slot penjadwalannya. Setelah tersinkronisasi, seorang penguji penetrasi (*pentester*) dapat mengirimkan *frame* ke dalam slot statis yang diidentifikasi dari jadwal FIBEX atau ARXML. Targetnya adalah slot sinyal yang krusial bagi keselamatan, seperti perintah untuk kontrol sasis, sistem kemudi elektronik (*steer-by-wire*), atau sistem pengereman elektronik (*brake-by-wire*). Penyerang dapat menyisipkan *payload* (muatan data) yang dikendalikannya, yang kemudian akan diproses oleh *node* penerima sebagai data yang sah karena tidak adanya mekanisme autentikasi pesan.

Selain itu, karena sifat waktu yang deterministik pada FlexRay juga merupakan aspek keselamatan, seorang pentester dapat menguji ketahanan sistem terhadap masalah waktu. Hal ini dilakukan dengan mengirimkan *frame* yang memiliki nilai penghitung siklus (*cycle counter*) yang tidak tepat, atau dengan mengganggu mekanisme sinkronisasi jam. Tindakan tersebut dapat menyebabkan *node* dalam klaster melaporkan kesalahan sinkronisasi dan beralih ke kondisi aman (*safe state*), yang pada akhirnya mengakibatkan penolakan layanan (*denial of service*) terhadap fungsi-fungsi krusial yang didukung oleh klaster tersebut. Setelah meninjau berbagai teknik pengujian penetrasi yang umum digunakan untuk jaringan di dalam kendaraan, studi kasus berikut akan mengilustrasikan bagaimana beberapa teknik tersebut diterapkan dalam serangan nyata terhadap komunikasi di dalam kendaraan.

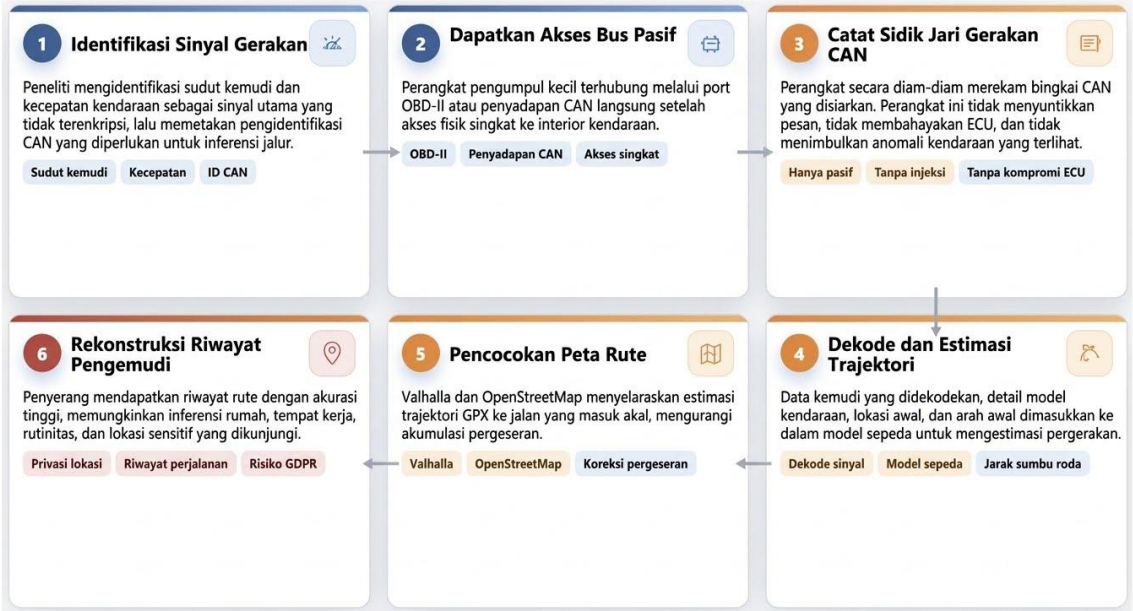
6.5 STUDI KASUS SERANGAN SIBER PADA JARINGAN INTERNAL KENDARAAN

Pada bagian ini, kita akan menelaah tiga serangan otomotif di dunia nyata, serta memeriksa bagaimana penyerang mengombinasikan berbagai teknik untuk menyerang jaringan di dalam kendaraan. Studi kasus ini menunjukkan bagaimana kelemahan dalam komunikasi jaringan di dalam kendaraan dapat ditemukan dan dieksploitasi oleh penyerang. Meskipun fokus utamanya adalah pada aspek jaringan di dalam kendaraan, konteks yang relevan tetap disertakan agar cakupan dan dampak serangan dapat dipahami secara menyeluruh. Ketiga studi kasus ini secara keseluruhan menggambarkan luasnya lanskap ancaman yang dihadapi oleh komunikasi di dalam kendaraan mulai dari pelanggaran privasi hingga pencurian kendaraan dan kendali jarak jauh atas fungsi-fungsi yang krusial bagi keselamatan.

Studi Kasus 1: Rekonstruksi Rute Kendaraan Melalui Penyadapan Lalu Lintas Pada Bus CAN

Studi kasus ini berfokus pada serangan terhadap privasi yang memanfaatkan ketiadaan perlindungan kerahasiaan (*confidentiality*) pada bus CAN. Hal ini menunjukkan bagaimana seorang penyerang dapat merekonstruksi rute persis yang dilalui oleh suatu kendaraan dengan cara membaca lalu lintas bus CAN secara pasif tanpa menyuntikkan pesan, tanpa meretas ECU, dan tanpa memicu anomali apa pun yang dapat teramati pada perilaku kendaraan tersebut. Studi ini menunjukkan bahwa meskipun bus CAN tidak memuat koordinat GPS, data sensor yang terkandung di dalamnya cukup untuk merekonstruksi riwayat perjalanan pengemudi dengan tingkat akurasi yang tinggi. Dengan menggunakan model kendaraan fisik dan algoritma pencocokan peta (*map-matching*), para peneliti berhasil mencapai akurasi rekonstruksi rute

rata-rata sebesar 95% pada empat kendaraan nyata di 41 lintasan uji dengan kondisi jalan dan lalu lintas yang beragam. Studi ini menyoroti bahwa akses terhadap jaringan di dalam kendaraan tidak hanya menimbulkan risiko keselamatan dan keamanan, tetapi juga menjadi ancaman serius yang selama ini kurang diperhatikan terhadap privasi pengemudi. Urutan langkah-langkah serangan untuk studi kasus ini disajikan dalam Gambar 6.3.



Gambar 6.3 Rekonstruksi rute kendaraan melalui penyadapan lalu lintas pada bus CAN

Penemuan dan Pengintaian

Tahap pertama melibatkan identifikasi sinyal CAN dan OBD-II yang diperlukan serta mencukupi untuk merekonstruksi jalur perjalanan suatu kendaraan. Para peneliti memulainya dengan meninjau sinyal-sinyal yang tersedia pada bus CAN di berbagai platform kendaraan. Meskipun bus CAN tidak mentransmisikan koordinat GPS, bus tersebut membawa beragam data sensor yang dapat digunakan sebagai proksi untuk menyimpulkan lokasi. Sinyal utama yang diidentifikasi adalah sudut kemudi (*steering wheel angle*) dan kecepatan kendaraan; keduanya disiarkan secara berkala pada bus CAN tanpa enkripsi maupun kontrol akses. Perlu dicatat bahwa sinyal-sinyal ini, jika dilihat secara terpisah, tidak mengungkapkan informasi lokasi spesifik apa pun. Akan tetapi, jika digabungkan dari waktu ke waktu, sinyal-sinyal tersebut membentuk "sidik jari pergerakan" kendaraan yang dapat dikorelasikan dengan peta jalan digital untuk merekonstruksi urutan jalan yang dilalui serta belokan yang dilakukan. Untuk mengidentifikasi pengenal (*identifier*) CAN dan pemetaan sinyal yang relevan pada setiap kendaraan yang diuji, para peneliti menggunakan pesan permintaan dan tanggapan OBD-II standar, yang didukung oleh semua kendaraan yang mematuhi standar tersebut dan tidak memerlukan otorisasi khusus untuk diakses.

Untuk menentukan pengenal CAN khusus (*proprietary*) bagi sensor sudut kemudi, para peneliti menggerakkan setir ke kiri dan ke kanan saat kendaraan dalam kondisi stabil, sembari mencatat paket data pada bus CAN. Mereka mengamati bahwa perubahan kecil pada sudut

kemudi menghasilkan perubahan yang kecil dan konsisten pada bidang data yang bersangkutan. Pengenal CAN untuk kecepatan diperoleh melalui pesan permintaan dan tanggapan OBD-II standar, yang telah diseragamkan untuk semua kendaraan yang mematuhi standar tersebut. Hal ini memungkinkan mereka untuk mengaitkan muatan (*payload*) bingkai CAN mentah dengan perilaku fisik kendaraan, serta mengekstrak nilai sinyal yang diperlukan untuk menyimpulkan lintasan kendaraan.

Akses Awal

Serangan ini memerlukan akses langsung ke bus CAN, yang dilakukan oleh para peneliti dengan menghubungkan perangkat pengumpul data ke *port* diagnostik OBD-II kendaraan atau dengan menyambungkan perangkat tersebut langsung ke jalur bus CAN. Kedua metode akses ini menuntut akses fisik singkat ke bagian dalam kendaraan target sebuah kondisi yang menurut para peneliti mungkin terjadi dalam berbagai skenario praktis. Sebagai contoh, seorang mekanik atau teknisi servis yang sedang melakukan pekerjaan rutin pada kendaraan dapat memasang perangkat pengumpul data berukuran kecil di lokasi tersembunyi di dekat port OBD-II dalam hitungan menit. Para peneliti juga menyebutkan adanya perangkat pencurian mobil komersial yang tersedia di pasar daring; perangkat ini memungkinkan pelaku untuk mengakses bagian dalam kendaraan dengan cepat guna memasang alat pengumpul data tanpa memicu alarm.

Setelah perangkat pengumpul data mendapatkan akses ke bus CAN, perangkat tersebut akan beroperasi secara diam-diam dalam mode pasif (menguping). Karena bus CAN merupakan media siaran (*broadcast*) tanpa autentikasi antar-*node*, perangkat tersebut dapat menerima semua bingkai data yang dikirimkan oleh setiap ECU di dalam jaringan tanpa perlu menyuntikkan lalu lintas data apa pun, sehingga keberadaannya tidak dapat dideteksi oleh ECU lain.

Eksploitasi

Inti dari serangan ini adalah algoritma inferensi lintasan deterministik yang menerjemahkan data sensor CAN mentah menjadi rekonstruksi jalur perjalanan (seperti lintasan GPS). Setelah data sudut kemudi dan kecepatan kendaraan terkumpul, para peneliti menerapkan model fisik "sepeda" (*bicycle model*) sebuah model kinematika yang mapan dalam studi dinamika kendaraan untuk menghitung posisi kendaraan secara bertahap seiring berjalannya waktu. Berdasarkan lokasi awal kendaraan dan arah tuju awal yang diketahui, model ini menghitung jarak serta arah perjalanan selama setiap rentang waktu, sehingga menghasilkan rangkaian estimasi koordinat geografis dalam format GPS *Exchange Format* (GPX). Input eksternal yang diperlukan mencakup lokasi awal kendaraan, arah tuju awal, serta informasi mengenai model kendaraan tersebut.

Model kendaraan ini menentukan pengidentifikasi sudut setir yang tepat, rumus dekode untuk sensor, dan nilai jarak sumbu roda (*wheelbase*) yang digunakan dalam model sepeda (*bicycle model*) untuk menghitung radius belok. Dalam praktiknya, informasi-informasi ini dapat diakses dengan mudah oleh pihak penyerang. Alamat rumah pemilik sering kali dapat diketahui melalui catatan publik, media sosial, atau pengamatan sebelumnya, sementara arah tuju awal kendaraan dapat diperkirakan berdasarkan orientasi jalan masuk (*driveway*) atau

tempat parkir. Model kendaraan dapat diamati secara langsung dari tampilan luar mobil atau diperoleh melalui catatan registrasi yang tersedia untuk umum.

Karena model kinematik mengakumulasi kesalahan posisi seiring berjalannya waktu, para peneliti menyertakan langkah pencocokan peta (*map-matching*) menggunakan mesin perutean sumber terbuka (*open-source*) Valhalla beserta data geografis dari *OpenStreetMap*. Algoritme pencocokan peta ini mengoreksi estimasi posisi pada setiap jendela waktu dengan menyesuaikan lintasan yang diperkirakan ke jalan terdekat yang masuk akal di peta. Langkah koreksi ini secara signifikan mengurangi penyimpangan akumulatif dan menjadi kunci untuk mencapai akurasi tinggi sebagaimana dilaporkan oleh para peneliti.

Eksploitasi teknis ini terdiri dari empat fase utama: pengambilan data CAN menggunakan perangkat pengumpul pasif; ekstraksi dan dekode sinyal menggunakan permintaan OBD-II serta pengidentifikasi CAN untuk sudut kemudi yang telah direkayasa balik (*reverse-engineered*); estimasi lintasan menggunakan model sepeda (*bicycle model*) dengan mempertimbangkan jarak sumbu roda kendaraan dan arah awal (*initial bearing*); serta koreksi jalur melalui pencocokan peta menggunakan Valhalla dan *OpenStreetMap*. Metode ini tidak memerlukan data pelatihan pembelajaran mesin (*machine learning*), basis data jalan khusus kendaraan, ataupun akses ke ponsel pintar pengemudi maupun perangkat pribadi lainnya. Satu-satunya prasyarat adalah kemampuan untuk mengumpulkan data bus CAN baik dengan memasang perangkat pengumpul data pada kendaraan maupun melalui ECU yang telah disusupi serta pengetahuan mengenai lokasi awal dan arah awal kendaraan, yang semuanya dapat diperoleh penyerang melalui pengamatan sebelumnya atau informasi yang tersedia untuk umum.

Persistensi

Serangan yang disebut *On Path Diagnostic - Intrusion and Inference* (OPD-II) ini tidak bergantung pada akses persisten dalam pengertian tradisional, karena serangan ini lebih bersifat pengumpulan data daripada penyusupan yang berkelanjutan. Jika perangkat pengumpul tetap terpasang secara fisik pada kendaraan, perangkat tersebut akan terus mencatat lalu lintas bus CAN pada setiap perjalanan berikutnya, sehingga memungkinkan penyerang untuk menyusun rekam jejak pergerakan pengemudi yang mendetail dan terus berkembang dari waktu ke waktu. Perlu dicatat bahwa dalam skenario nyata, penyerang dapat memperluas pendekatan ini dengan menggunakan perangkat pengumpul yang dilengkapi konektivitas seluler untuk mengirimkan data bus CAN keluar secara jarak jauh (*exfiltration*) hampir secara waktu nyata (*near-real time*).

Karena perangkat ini beroperasi secara senyap dan tidak menyuntikkan lalu lintas apa pun ke dalam bus CAN, perangkat tersebut tidak menimbulkan anomali yang dapat diamati pada perilaku kendaraan dan tidak dapat dideteksi oleh ECU lain di dalam jaringan. Satu-satunya hal yang berpotensi mengungkap serangan ini adalah keberadaan fisik perangkat kecil (seperti dongle OBD-II) atau perangkat lain yang disembunyikan di balik dasbor. Namun, kemungkinan untuk terdeteksi sangat kecil, kecuali jika pengemudi secara aktif memeriksa kendaraan tersebut.

Dampak

Dampak utama dari serangan melalui OBD-II adalah pelanggaran serius terhadap privasi lokasi pengemudi. Dengan merekonstruksi rute perjalanan kendaraan dengan tingkat akurasi koordinat rata-rata mencapai 95% penyerang dapat memperoleh catatan terperinci mengenai pergerakan pengemudi dari waktu ke waktu. Informasi ini dapat digunakan untuk menyimpulkan alamat rumah, tempat kerja, lokasi yang sering dikunjungi, rutinitas harian, serta hubungan sosial pengemudi; semua data tersebut dikategorikan sebagai data pribadi sensitif menurut peraturan privasi modern, termasuk *General Data Protection Regulation* (GDPR). Serangan ini juga signifikan dari perspektif sistemik. Karena sinyal pada CAN bus dan OBD-II telah distandarisasi di berbagai platform kendaraan, metodologi ini tidak terbatas pada merek atau model tertentu saja. Para peneliti telah memvalidasi serangan ini pada empat kendaraan berbeda, yang menunjukkan cakupan penerapannya yang luas. Penggunaan perangkat sumber terbuka (*open-source*), seperti Valhalla dan *OpenStreetMap*, berarti serangan ini tidak memerlukan infrastruktur khusus (*proprietary*) dan dapat direplikasi dengan biaya rendah.

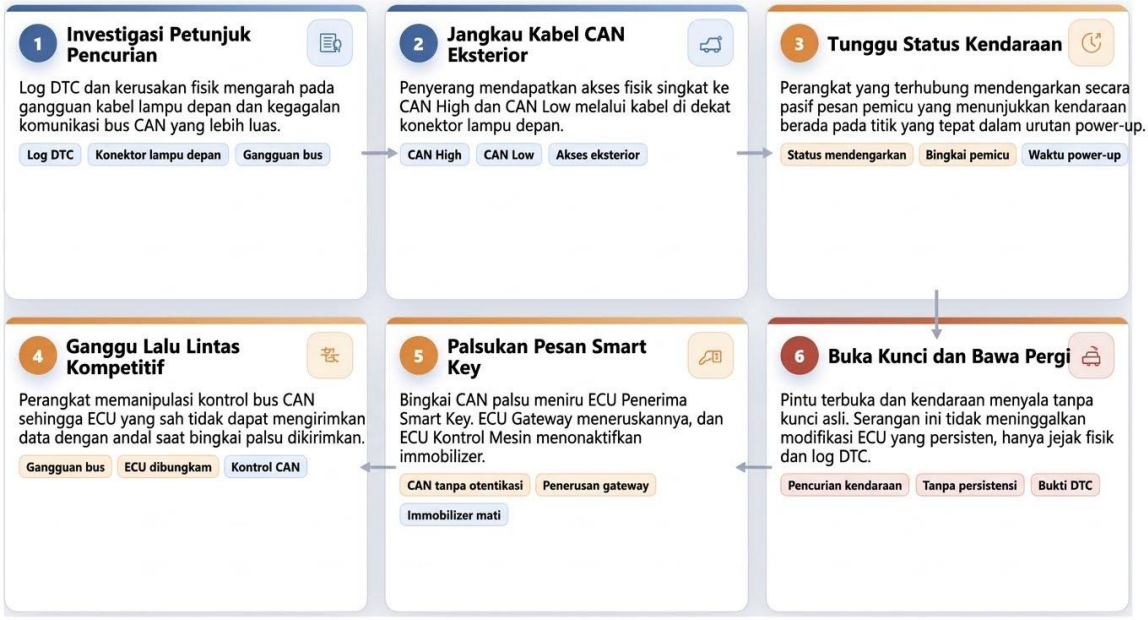
Dari sudut pandang keamanan siber otomotif, studi kasus ini menunjukkan bahwa kurangnya kerahasiaan (*confidentiality*) pada CAN bus bukan sekadar kekhawatiran teoretis. Ini merupakan risiko privasi nyata yang dapat dieksploitasi, yang memengaruhi setiap kendaraan yang mengandalkan komunikasi jaringan internal tanpa enkripsi. Pengemudi tidak memiliki indikasi apa pun bahwa pengumpulan data ini sedang terjadi, dan operasional kendaraan sama sekali tidak terganggu, menjadikannya salah satu kategori serangan yang paling sulit dideteksi.

Studi Kasus 2: Serangan Injeksi CAN Untuk Memintas Sistem Kunci Pintar (*Smart Key*) Guna Pencurian Kendaraan

Studi kasus ini menjabarkan teknik pencurian kendaraan di dunia nyata yang mengeksploitasi ketiadaan autentikasi pesan pada CAN bus untuk memintas sistem immobilizer kunci pintar (*smart key*) kendaraan. Berbeda dengan serangan *relay* kunci pintar tradisional yang mencegat dan mengirim ulang sinyal nirkabel dari kunci pintar serangan injeksi CAN memintas sistem kunci pintar sepenuhnya dengan cara menyuntikkan pesan CAN palsu langsung ke dalam jaringan internal kendaraan.

Serangan ini diselidiki dan didokumentasikan oleh para peneliti keamanan setelah salah satu dari mereka kehilangan mobil Toyota RAV4 miliknya akibat dicuri. Studi tersebut mengungkapkan bahwa ini bukanlah insiden terisolasi, melainkan teknik serangan yang tersebar luas dan digunakan dalam pencurian kendaraan terorganisir, yang menyasar berbagai model kendaraan dari berbagai produsen. Serangan ini sendiri tidak memerlukan pengetahuan kriptografi maupun interaksi apa pun dengan kunci pintar yang asli. Sebaliknya, metode ini mengeksploitasi salah satu kelemahan paling mendasar dari bus CAN: setiap *node* yang terhubung ke bus dapat mengirimkan pesan apa pun, dan semua node lainnya akan menerimanya sebagai pesan yang sah. Studi tersebut menunjukkan bahwa perangkat pencurian yang dirancang khusus dengan komponen yang menelan biaya sekitar Rp 100.000 dapat membuka kunci kendaraan dan menonaktifkan sistem imobilizernya. Rekaman CCTV

dari sebuah pencurian nyata mengonfirmasi bahwa seluruh proses tersebut memakan waktu sekitar 90 detik. Langkah-langkah alur serangan diilustrasikan pada Gambar 6.4.



Gambar 6.4 Alur eksploitasi pencurian kendaraan melalui serangan injeksi CAN untuk memintas sistem kunci pintar (*smart key*)

Penemuan dan Pengintaian

Investigasi dimulai setelah salah satu kendaraan milik peneliti keamanan mengalami gangguan dalam dua kesempatan terpisah; bumper depan ditarik lepas dan konektor kabel lampu depan dicabut. Awalnya, kejadian ini tampak seperti tindakan vandalisme, namun dua hari kemudian kendaraan tersebut dicuri. Dengan memanfaatkan sistem telematika bawaan kendaraan, para peneliti memperoleh catatan *Diagnostic Trouble Codes* (DTC) yang terekam di sekitar waktu pencurian. DTC tersebut menunjukkan adanya kegagalan komunikasi dengan *Electronic Control Unit* (ECU) pengatur sistem pencahayaan, yang selaras dengan kondisi konektor lampu depan yang terlepas secara fisik. Lebih jauh lagi, catatan tersebut menunjukkan bahwa beberapa sistem lain seperti kontrol kamera depan dan sistem kontrol mesin hibrida juga mengalami kegagalan komunikasi pada waktu yang bersamaan. Dengan demikian, hal ini mengindikasikan bahwa masalah yang terjadi bukanlah kegagalan pada satu ECU saja, melainkan masalah pada jalur komunikasi CAN (*CAN bus*) itu sendiri.

Para peneliti kemudian melakukan pengintaian dengan menelusuri forum, pasar di *dark web*, serta situs web yang dapat diakses publik yang menjual produk untuk membobol sistem keamanan mobil. Penelusuran ini mengungkap ekosistem komersial perangkat pencurian yang menyasar berbagai model kendaraan dari beragam produsen. Dalam contoh ini, perangkat pencurian tersebut tampak seperti pengeras suara (*speaker*) Bluetooth portabel merek JBL, yang memberikan dalih masuk akal jika pencuri dihentikan oleh aparat penegak hukum. Perangkat pencurian berbasis injeksi CAN yang menyamar sebagai produk JBL ini dijual dengan label "perangkat penyala darurat" (*emergency start device*).

Para peneliti membeli perangkat pencurian JBL CAN tersebut dan melakukan pemeriksaan menyeluruh. Pembongkaran perangkat kerasnya mengungkapkan adanya mikrokontroler PIC18F yang dilengkapi perangkat keras CAN dengan *firmware* yang telah diprogram sebelumnya, sebuah *transceiver* CAN untuk mengubah sinyal digital menjadi tegangan analog yang digunakan pada jalur kabel bus CAN, serta sirkuit tambahan yang terhubung ke transceiver CAN tersebut. Sirkuit tambahan ini berfungsi untuk memaksa bus CAN yang terhubung masuk ke dalam kondisi recessive, yang secara efektif mencegah unit kontrol elektronik (*Electronic Control Unit* atau ECU) lain untuk mengirimkan data. Total biaya komponen perangkat tersebut diperkirakan sekitar sepuluh dolar AS.

Akses Awal

Serangan ini memerlukan akses fisik ke jaringan kabel CAN bus kendaraan, namun penyerang dapat memperoleh akses ke CAN bus tanpa harus membobol masuk ke dalam kendaraan itu sendiri. Kendaraan dalam studi kasus ini memiliki beberapa CAN bus yang melayani domain fungsional berbeda, termasuk CAN bus kontrol yang menghubungkan ECU untuk sistem pencahayaan, kendali pintu, dan telematika; *CAN bus powertrain* yang menghubungkan ECU untuk kendali mesin dan motor hibrida; serta CAN bus otonom untuk sistem radar dan kamera. Para penyerang menargetkan CAN bus kontrol, yang terhubung dengan ECU penerima *Smart Key™.*

Dalam kasus ini, titik masuk ke CAN bus kontrol adalah melalui konektor lampu depan, yang diakses oleh penyerang dengan menarik bumper depan. Penyerang hanya membutuhkan waktu beberapa menit dan peralatan dasar untuk melakukan hal ini. Setelah konektor lampu depan dapat diakses, perangkat pencurian berbasis injeksi CAN dihubungkan langsung ke jalur CAN *High* dan CAN *Low* pada CAN bus kontrol, dengan daya yang bersumber dari baterai internal perangkat tersebut. Akses ke konektor lampu depan merupakan satu-satunya interaksi fisik yang diperlukan terhadap kendaraan target, karena seluruh tahapan serangan selanjutnya berlangsung melalui CAN bus.

Eksploitasi

Setelah dihubungkan dan dinyalakan, perangkat pencurian berbasis injeksi CAN akan masuk ke mode pemantauan (*listening state*), yakni mengawasi lalu lintas data pada bus untuk mendeteksi pesan CAN spesifik yang menandakan bahwa kendaraan berada dalam kondisi siap (*ready state*). Desain ini mencegah perangkat aktif terlalu dini dan memastikan rangkaian serangan dimulai pada momen yang tepat dalam siklus penyalaan kendaraan. Perlu dicatat mengapa serangan ini tetap efektif meskipun sistem *smart key* menggunakan protokol kriptografi *challenge-response* untuk mengautentikasi *key fob* (kunci pintar). Perlindungan kriptografi ini hanya mencakup komunikasi nirkabel antara key fob fisik dan ECU penerima Smart Key. Pesan-pesan CAN yang dikirimkan kemudian oleh ECU penerima *Smart Key* ke ECU kendali mesin (*Engine Control* ECU) untuk menonaktifkan immobilizer ternyata tidak diautentikasi. Artinya, perangkat apa pun yang terhubung ke CAN bus dapat menyamar sebagai ECU penerima *Smart Key* dengan menyuntikkan pesan yang sama, tanpa memerlukan pengetahuan kriptografi apa pun.

Setelah perangkat pencurian berbasis injeksi CAN menerima pesan pemicu yang menandakan kendaraan dalam kondisi siap, perangkat tersebut menjalankan dua tindakan secara bersamaan. Pertama, perangkat mengaktifkan sirkuit *transceiver* CAN yang telah dimodifikasi, yang secara aktif memaksa bus masuk ke dalam *status recessive* (pasif) yang tidak dapat diganggu oleh *node* lain. Tindakan ini mencegah semua *node* lain pada bus CAN kendali untuk melakukan transmisi sekaligus menonaktifkan mekanisme deteksi kesalahan CAN. Kedua, perangkat mulai mengirimkan *frame* CAN dengan kecepatan sekitar 20 frame per detik, yang berisi pesan palsu seolah-olah berasal dari ECU Penerima Kunci Pintar (*Smart Key Receiver* ECU) dengan instruksi: "Kunci valid, buka kunci immobilizer." Pada kendaraan yang tidak menerapkan autentikasi pesan di bus CAN, *frame* ini tidak dapat dibedakan dari pesan sah yang dikirimkan oleh ECU Penerima Kunci Pintar yang asli. Selain itu, karena ECU Penerima Kunci Pintar yang asli tidak dapat mengirimkan pesan, pesan palsu yang disuntikkan tersebut dapat diterima oleh ECU *Gateway* tanpa gangguan.

ECU *Gateway*, yang menghubungkan bus CAN kendali dengan bus CAN *powertrain* (sistem penggerak), meneruskan pesan CAN palsu tersebut ke bus *powertrain*. Di sana, ECU Kontrol Mesin (*Engine Control* ECU) menerima pesan tersebut dan menonaktifkan *immobilizer*. Pengiriman pesan secara berulang dengan kecepatan 20 frame per detik ini berfungsi untuk mengompensasi ketidakstabilan yang muncul akibat manipulasi bus (yang bertujuan membungkam ECU lain), sehingga memastikan ECU *Gateway* dapat menerima dan meneruskan pesan palsu tersebut dengan andal. Selanjutnya, untuk membuka kunci pintu kendaraan, pelaku menekan tombol *Play* pada pengeras suara JBL yang terhubung ke mikrokontroler PIC18F. Tindakan ini mengubah isi (*payload*) pesan CAN yang disuntikkan, sehingga memberikan instruksi kepada ECU kontrol pintu untuk membuka kunci pintu. Setelah itu, perangkat pencurian berbasis injeksi CAN dilepas, dan pelaku dapat masuk serta menyalakan mobil tanpa menggunakan kunci.

Persistensi

Berbeda dengan banyak serangan siber yang bertujuan untuk membangun akses jangka panjang ke sistem yang telah disusupi, serangan injeksi CAN tidak memiliki tujuan persistensi. Serangan ini tidak memodifikasi perangkat lunak atau *firmware* ECU apa pun, dan tidak ada perubahan permanen yang dilakukan terhadap konfigurasi atau perilaku kendaraan. Satu-satunya tujuan adalah pencurian kendaraan, yang dicapai dalam satu kali operasi. Setelah sistem *imobilizer* dinonaktifkan dan pintu dibuka, kendaraan dibawa pergi, dan perangkat pencurian pun dilepas. Tidak diperlukan interaksi lebih lanjut dengan jaringan di dalam kendaraan. Satu-satunya bukti yang tertinggal adalah kerusakan fisik pada bumper depan dan konektor lampu depan, serta kode masalah diagnostik (DTC) yang tercatat oleh ECU kendaraan akibat gangguan pada bus CAN selama serangan berlangsung.

Dampak

Dampak langsungnya adalah pencurian kendaraan target tanpa memerlukan *key fob* (kunci elektronik) dan tanpa memicu peringatan apa pun yang dapat dilihat oleh pemilik secara *real-time*. Skala dampaknya jauh melampaui satu kendaraan saja. Penelusuran yang dilakukan para peneliti mengungkap adanya pasar terorganisasi secara komersial untuk perangkat

pencurian berbasis injeksi CAN (CAN *injection*), yang menyasar puluhan model kendaraan dari berbagai produsen. Perangkat-perangkat ini diproduksi dengan biaya rendah, dijual dengan harga hingga lima ribu euro, dan merupakan bagian dari bisnis kriminal terstruktur yang dilengkapi dengan katalog produk serta daftar kompatibilitas model kendaraan. Karena kerentanan mendasarnya bersifat arsitektural berakar pada prinsip desain bus CAN di mana semua *node* (titik koneksi) memercayai setiap pesan yang diterima maka setiap *Electronic Control Unit* (ECU) dalam jaringan yang merespons pesan CAN tanpa autentikasi berpotensi dieksploitasi melalui teknik serangan yang sama.

Dari perspektif keamanan siber otomotif, studi kasus ini menunjukkan bagaimana kelemahan pada tingkat protokol yaitu tidak adanya autentikasi pesan pada bus CAN dapat langsung berubah menjadi serangan nyata yang berskala luas dan berbiaya rendah, dengan konsekuensi finansial yang signifikan bagi pemilik maupun produsen kendaraan. Hal ini juga menunjukkan bahwa jalur fisik kabel bus CAN seperti yang melewati konektor lampu depan di dekat bagian luar kendaraan harus dipertimbangkan sebagai bagian dari *attack surface* (permukaan serangan) dalam pemeriksaan ancaman.

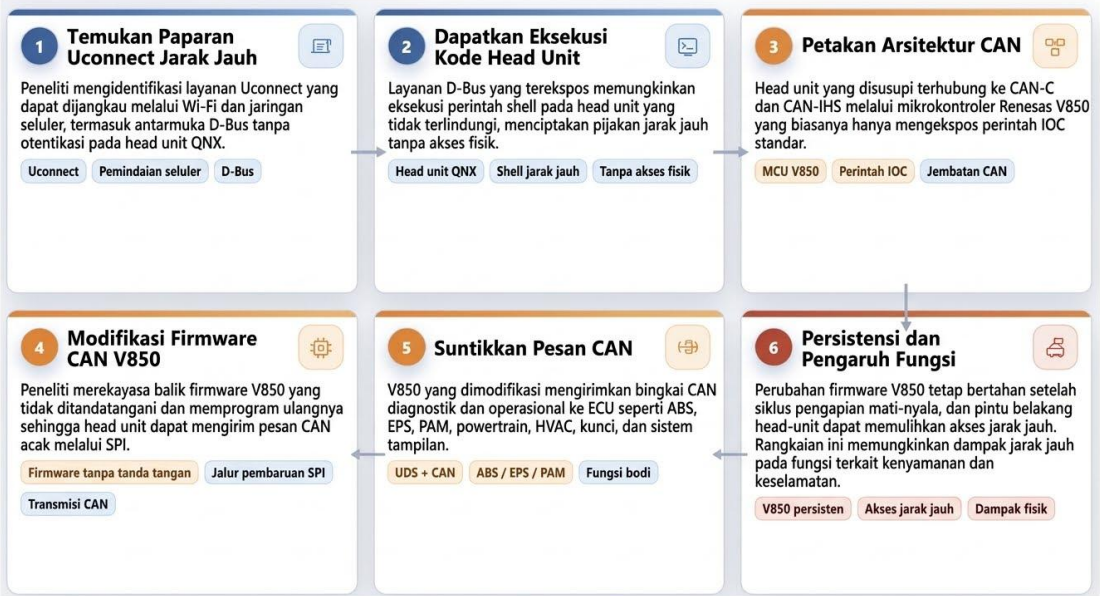
Studi Kasus 3: Eksploitasi Jarak Jauh Pada Kendaraan Penumpang – Dari Jaringan Seluler ke Bus CAN

Studi kasus ini menguraikan salah satu demonstrasi studi keamanan siber otomotif yang paling signifikan dan sering dikutip. Dalam studi tersebut, para peneliti keamanan berhasil mendapatkan akses jarak jauh penuh ke kendaraan produksi standar (tanpa modifikasi) melalui jaringan seluler, kemudian menyuntikkan pesan CAN untuk mengendalikan fungsi fisik yang krusial bagi keselamatan. Studi ini dilakukan pada Jeep Cherokee keluaran 2014 yang dilengkapi dengan sistem infotainment dan telematika *Uconnect*, serta didemonstrasikan kepada publik pada tahun 2015.

Berbeda dengan dua studi kasus sebelumnya yang memerlukan akses fisik singkat ke kendaraan target, serangan ini dilakukan sepenuhnya dari jarak jauh, tanpa interaksi fisik apa pun dengan kendaraan tersebut. Rangkaian serangan ini mencakup beberapa lapisan, dimulai dari jaringan seluler, berlanjut melalui sistem infotainment kendaraan, melewati chip perantara yang menangani komunikasi CAN, hingga akhirnya mencapai ECU yang krusial bagi keselamatan pada bus CAN di dalam kendaraan tersebut. Karena fokus bab ini adalah pada komunikasi jaringan di dalam kendaraan, pertama-tama kami menyajikan gambaran singkat mengenai bagaimana para peneliti memperoleh akses jarak jauh ke kendaraan tersebut, sebelum menjelaskan eksploitasi jaringan di dalam kendaraan yang dilakukan setelahnya.

Para peneliti menemukan bahwa sistem *Uconnect* mengekspos titik akses Wi-Fi dengan kata sandi yang dapat diprediksi, yang bisa diperoleh berdasarkan stempel waktu (*timestamp*) saat hotspot Wi-Fi tersebut diaktifkan. Dengan terhubung ke jaringan Wi-Fi ini dan memindai *port* yang terbuka, mereka menemukan layanan *Desktop* Bus (D-Bus) yang tidak memerlukan autentikasi yang berjalan pada unit kepala (*head unit*). Layanan ini kemudian mereka manfaatkan untuk mengeksekusi kode pada sistem *Uconnect* yang menjalankan sistem operasi QNX. Untuk menjangkau area di luar jangkauan Wi-Fi, para peneliti menggunakan *femtocell* sebuah stasiun pangkalan seluler berukuran kecil guna menghubungkan peralatan mereka ke

rentang IP jaringan seluler yang sama dengan yang digunakan oleh kendaraan-kendaraan rentan tersebut. Urutan langkah-langkah serangan ditunjukkan pada Gambar 6.5.



Gambar 6.5 Eksploitasi jarak jauh pada kendaraan penumpang - Alur eksploitasi dari jaringan seluler hingga ke bus CAN

Hal ini memungkinkan mereka untuk memindai rentang alamat IP operator seluler, mengidentifikasi kendaraan yang memiliki port terbuka, serta mengakses layanan D-Bus yang tidak memerlukan autentikasi secara langsung melalui jaringan seluler. Setelah target teridentifikasi, eksploitasi yang sama dijalankan dari jarak jauh, sehingga memungkinkan eksekusi kode pada head unit tanpa perlu adanya interaksi apa pun dengan kendaraan maupun pengemudinya. Dari titik akses inilah studi kasus ini dimulai. Langkah selanjutnya mencakup upaya untuk memperoleh akses ke bus CAN (*Controller Area Network*) di dalam kendaraan serta menyuntikkan pesan-pesan tertentu untuk memengaruhi fungsi fisik kendaraan.

Penemuan dan Pengintaian

Setelah berhasil mengeksekusi kode pada unit kepala (*head unit*) *Uconnect*, para peneliti mulai menjelajahi arsitektur internal kendaraan tersebut. Jeep Cherokee tahun 2014 menggunakan dua bus CAN: bus CAN-C, yang menghubungkan unit kontrol elektronik (ECU) yang krusial bagi keselamatan termasuk Sistem Pengereman *Anti-lock* (ABS), Kemudi Bertenaga Listrik (EPS), kontrol sistem penggerak (*powertrain*), kontrol transmisi, dan Modul Bantuan Parkir (*Park Assist Module*/PAM) serta bus CAN-IHS, yang menghubungkan ECU untuk kenyamanan dan bodi kendaraan, termasuk sistem Pemanas, Ventilasi, dan AC (HVAC), modul pintu, dan kursi dengan pemanas. Temuan kunci terkait arsitektur ini adalah bahwa unit kepala *Uconnect* yang menjalankan sistem operasi QNX pada prosesor Texas Instruments OMAP-DM3730 dan telah berhasil disusupi oleh para peneliti terhubung ke kedua bus CAN tersebut secara bersamaan. Hal ini menjadikannya titik tumpu (*pivot point*) potensial untuk mengakses seluruh ECU di dalam kendaraan.

Akan tetapi, unit kepala yang menjalankan QNX tersebut tidak memiliki kemampuan langsung untuk mengirim pesan CAN sembarang (*arbitrary CAN messages*). Para peneliti menemukan bahwa komunikasi CAN dari unit kepala ditangani oleh cip terpisah, yaitu mikrokontroler Renesas V850ES/FJ3, yang terletak di antara unit kepala dan bus CAN serta bertanggung jawab untuk menjalankan fungsi-fungsi spesifik terkait CAN. Komunikasi dengan V850 ini didasarkan pada perintah *Input Output Controller* (IOC) yang telah ditetapkan sebelumnya, di mana setiap perintah memicu tindakan CAN tertentu yang telah diprogram secara permanen (*hardcoded*) di dalam *firmware* V850. Para peneliti menyadari bahwa untuk dapat mengirim pesan CAN sembarang, mereka perlu memahami dan memodifikasi *firmware* V850 tersebut.

Akses Awal

Sebagaimana dijelaskan dalam pemantauan umum, eksekusi kode jarak jauh pada unit kepala (*head unit*) *Uconnect* berhasil dilakukan dengan mengeksploitasi layanan D-Bus tanpa autentikasi yang terbuka pada *port* 6667 di sistem operasi QNX. Layanan D-Bus bernama *NavTrailService* yang dirancang untuk komunikasi antarproses menyediakan metode eksekusi yang memungkinkan perintah shell apa pun dijalankan secara langsung. Dengan menggunakan empat baris kode Python melalui pustaka D-Bus, para peneliti berhasil membuka akses shell dengan hak akses *root* dari jarak jauh pada unit kepala yang belum dimodifikasi, tanpa memerlukan *jailbreak* atau perubahan sistem apa pun sebelumnya. Sebagaimana telah disebutkan, head unit tidak dapat mengirimkan pesan CAN sembarang karena adanya struktur perintah IOC yang telah ditetapkan sebelumnya pada mikrokontroler V850. Agar dapat mengirimkan pesan CAN sembarang, para peneliti perlu memodifikasi dan melakukan *reflash* (pemrograman ulang) pada *firmware* V850 tersebut.

Untuk mencapai hal ini, para peneliti melakukan rekayasa balik (*reverse engineering*) terhadap *firmware* V850 menggunakan IDA Pro, yang telah dilengkapi dengan modul prosesor untuk arsitektur V850. Skrip Python digunakan di dalam IDA Pro untuk menormalisasi hasil disassembler, memetakan referensi silang (*cross-reference*), serta mengidentifikasi fungsi-fungsi yang bertanggung jawab atas transmisi pesan CAN. Selain itu, IAR™ Workbench sebuah lingkungan pengembangan yang lazim digunakan dalam pengembangan sistem tertanam (*embedded system*) otomotif juga dirujuk karena menyediakan contoh kode CAN untuk keluarga prosesor yang sama.

Hal ini membantu para peneliti dalam mengidentifikasi dan mengonfirmasi fungsi-fungsi transmisi CAN yang relevan di dalam *firmware* tersebut. Melalui pemeriksaan ini, para peneliti berhasil mengidentifikasi suatu fungsi yang menerima CAN ID, panjang data, dan muatan pesan (*payload*), serta memetakan saluran komunikasi *Serial Peripheral Interface* (SPI) yang digunakan oleh chip OMAP untuk mengirimkan perintah ke V850. *Firmware* V850 yang telah dimodifikasi ini kemudian siap untuk di-*flash* (dimuat ke dalam perangkat). Lebih lanjut, citra (*image*) *firmware* tersebut tidak ditandatangani secara kriptografis, yang berarti tidak ada verifikasi tanda tangan digital yang dilakukan selama proses pembaruan. Berkas biner iocupdate alat yang digunakan untuk memuat *firmware* baru ke V850 melalui SPI sudah

tersedia di dalam sistem berkas head unit, sehingga para peneliti tidak perlu memuat perangkat tambahan apa pun ke dalam sistem target.

Akan tetapi, proses pembaruan *firmware* standar mengharuskan V850 berada dalam mode *bootloader*. Mode ini hanya dapat dipicu melalui reset sistem menyeluruh, yang menyebabkan prosesor OMAP berhenti beroperasi untuk sementara dan mengakibatkan para peneliti kehilangan akses jarak jauh. Untuk mengatasi kendala tersebut, mereka membuat partisi *MultiMediaCard* (MMC) agar dapat ditulis (*writable*) menggunakan perintah mount standar. Selanjutnya, mereka mengganti utilitas hex dump bernama `hd` sebuah berkas eksekusi yang dipanggil pada tahap awal urutan boot pembaruan di dalam sistem berkas MMC dengan skrip shell kustom. Skrip ini berfungsi memanggil `iocupdate` yang membawa citra (*image*) *firmware* hasil modifikasi, lalu memulai ulang sistem kembali ke mode aplikasi. Kemudian, mereka memicu sistem untuk melakukan reboot ke *mode bootloader* dengan menggunakan skrip Lua yang menjalankan modul sistem `onoff`. Keseluruhan proses re*flash*ing, termasuk siklus reboot, memakan waktu sekitar 25 detik. Setelah itu, sistem kembali ke mode aplikasi dengan *firmware* V850 yang telah dimodifikasi.

Eksploitasi

Dengan *firmware* V850 yang telah dimodifikasi, para peneliti mampu mengirimkan pesan CAN arbitrer (sewenang-wenang) ke bus CAN-C maupun CAN-IHS dari jarak jauh melalui jaringan seluler. Tahap eksploitasi kemudian mencakup identifikasi dan penyusunan pesan CAN spesifik yang diperlukan untuk memicu tindakan fisik pada kendaraan. Para peneliti menggunakan pesan CAN diagnostik standar UDS serta pesan CAN operasional biasa untuk berinteraksi dengan ECU target. Untuk setiap fungsi target, ID CAN dan muatan (*payload*) pesan yang relevan diidentifikasi melalui kombinasi rekayasa balik (*reverse engineering*) *firmware* V850 dan pemantauan lalu lintas CAN secara langsung.

Tindakan yang berdampak pada keselamatan yang dicapai dengan menyuntikkan pesan CAN dari unit kepala (*head unit*) melalui *firmware* V850 yang telah dimodifikasi meliputi mematikan mesin saat kendaraan sedang melaju, menonaktifkan rem pada kecepatan rendah dengan mengirimkan pesan diagnostik ke modul ABS, serta mengendalikan sistem kemudi elektrik (*electric power steering*/EPS) dengan terlebih dahulu menempatkan modul PAM ke dalam sesi diagnostik untuk menghentikan pesan operasionalnya, lalu mengirimkan perintah ke ECU EPS. Fungsi-fungsi yang tidak terkait dengan keselamatan, seperti sistem HVAC, volume radio, tampilan layar, dan kunci pintu, dikendalikan melalui skrip Lua yang memanggil layanan D-Bus pada unit kepala secara langsung, tanpa perlu melakukan penyuntikan pesan CAN. Secara keseluruhan, teknik-teknik ini menunjukkan bahwa penyerang jarak jauh yang tidak memiliki akses fisik sebelumnya ke kendaraan dapat memengaruhi fungsi kenyamanan maupun perilaku kendaraan yang krusial bagi keselamatan.

Persistensi

Persistensi dalam serangan ini dapat ditinjau dari dua tingkat. Pada tingkat jaringan di dalam kendaraan, pemasangan (*flashing*) *firmware* V850 yang telah dimodifikasi ke dalam mikrokontroler memastikan bahwa kemampuan untuk mengirimkan pesan CAN arbitrer tetap ada meskipun kendaraan dimatikan dan dinyalakan kembali, karena *firmware* V850 tetap

tersimpan setelah kendaraan dimatikan dan dihidupkan ulang. Ini berarti kemampuan penyerang untuk mengirimkan pesan CAN arbitrer tetap terjaga di setiap siklus penyalaan mesin, terlepas dari status perangkat lunak pada unit kepala. Pada tingkat unit kepala, akses root shell jarak jauh yang diperoleh melalui layanan D-Bus tidak bersifat persisten setelah sistem dimulai ulang (*reboot*). Artinya, setelah kendaraan dimatikan dan dinyalakan kembali, penyerang harus mengeksploitasi ulang layanan D-Bus yang tidak memerlukan autentikasi tersebut untuk mendapatkan kembali akses shell jarak jauh.

Akan tetapi, para peneliti menunjukkan bahwa backdoor yang lebih persisten dapat dibuat dengan mengunggah kunci publik SSH dan memodifikasi skrip `boot.sh` pada partisi MMC yang dapat ditulis (*writable*), sehingga daemon SSH akan dijalankan setiap kali perangkat dinyalakan (*booting*). Karena perubahan pada partisi MMC tetap ada meskipun perangkat dinyalakan ulang, hal ini memberikan *backdoor* SSH permanen bagi penyerang yang tetap bertahan meski perangkat dimatikan dan dinyalakan kembali tanpa perlu melakukan eksploitasi D-Bus berulang kali. Dengan demikian, kombinasi antara backdoor SSH yang persisten pada head unit dan modifikasi *firmware* V850 yang juga persisten akan memberikan akses jarak jauh yang andal dan dapat diulang bagi penyerang ke head unit tersebut. Selain itu, penyerang juga mendapatkan kemampuan untuk mengirim pesan CAN arbitrer ke dalam bus CAN kendaraan, yang tetap dapat dilakukan meskipun kendaraan dimatikan dan dinyalakan kembali, selama kendaraan tersebut masih dapat dijangkau melalui jaringan seluler.

Dampak

Dampak dari serangan ini sangat signifikan, baik secara teknis maupun dari perspektif industri. Pada tingkat kendaraan individual, para peneliti menunjukkan bahwa penyerang jarak jauh yang sebelumnya tidak memiliki akses fisik ke kendaraan dapat melumpuhkan fungsi-fungsi krusial terkait keselamatan, termasuk mesin dan rem, membuat setir tersentak, serta memantau atau memanipulasi lingkungan pengemudi. Semua ini terjadi saat kendaraan sedang melaju, tanpa ada indikasi apa pun bagi pengemudi bahwa kendaraannya telah disusupi. Skala dampak potensialnya meluas jauh melampaui satu kendaraan saja. Para peneliti menunjukkan bahwa serangan melalui jaringan seluler dapat diotomatisasi untuk memindai dan mengeksploitasi semua kendaraan rentan yang terhubung ke jaringan operator seluler tersebut. Karena kerentanan pada sistem *Uconnect* ini mencakup seluruh generasi produk, maka sejumlah besar kendaraan yang saat itu berada di jalan raya berpotensi untuk diakses dari mana saja selama terdapat koneksi seluler.

Dari perspektif jaringan di dalam kendaraan, studi kasus ini menggambarkan beberapa kelemahan kritis. Koneksi langsung head unit ke kedua bus CAN tanpa isolasi yang memadai berarti bahwa satu titik penyusupan pada sistem *infotainment* dapat memberikan akses ke unit kontrol elektronik (ECU) yang krusial bagi keselamatan. Mekanisme pembaruan *firmware* V850 yang dapat diakses dari *head unit* melalui antarmuka SPI tidak memiliki kontrol autentikasi yang memadai maupun verifikasi tanda tangan digital untuk *firmware* V850 itu sendiri. Hal ini memungkinkan para peneliti untuk menggantinya dengan versi yang telah dimodifikasi, yang mampu mengirimkan pesan CAN apa pun sesuai keinginan penyerang.

Selain itu, tidak adanya autentikasi pesan pada bus CAN serta kurangnya segmentasi jaringan yang memadai berarti bahwa setelah V850 dimodifikasi, tidak ada mekanisme untuk mencegah atau mendeteksi pesan-pesan yang disisipkan ke dalam bus CAN. Secara keseluruhan, kelemahan-kelemahan ini membentuk rantai serangan yang utuh mulai dari koneksi seluler jarak jauh hingga kendali langsung atas fungsi-fungsi kendaraan yang krusial bagi keselamatan yang menunjukkan konsekuensi serius akibat isolasi yang tidak memadai dan kontrol keamanan yang lemah antara sistem telematika dan jaringan di dalam kendaraan. Studi kasus yang disajikan dalam bab ini menunjukkan bagaimana kelemahan dalam komunikasi di dalam kendaraan dapat diidentifikasi dan dieksploitasi dalam kondisi nyata. Bagian selanjutnya menguraikan praktik-praktik terbaik yang dapat diadopsi oleh organisasi otomotif untuk mengatasi kelemahan tersebut serta mengurangi keseluruhan permukaan serangan (*attack surface*) pada jaringan di dalam kendaraan.

6.6 STRATEGI PENGAMANAN KOMUNIKASI JARINGAN INTERNAL KENDARAAN

Mengamankan komunikasi jaringan di dalam kendaraan memerlukan pendekatan berlapis yang menangani kelemahan pada tingkat protokol, arsitektur, dan perangkat lunak secara bersamaan. Kelemahan yang dijelaskan dalam bab ini menyoroti serangkaian celah keamanan mendasar, seperti kurangnya autentikasi pesan, komunikasi tanpa enkripsi, segmentasi jaringan yang tidak memadai, dan antarmuka diagnostik yang tidak terlindungi.

Menangani celah-celah ini dimulai dengan mengadopsi pola pikir *zero-trust* (kepercayaan nol) yaitu mengasumsikan bahwa setiap *Electronic Control Unit* (ECU) dalam jaringan dapat disusupi dan penyerang mungkin sudah memiliki akses di dalam kendaraan. Dalam praktiknya, hal ini berarti menerapkan autentikasi timbal balik (*mutual authentication*) antar-ECU yang berkomunikasi, menerapkan autentikasi dan enkripsi pesan jika diperlukan, serta melakukan segmentasi jaringan untuk membatasi dampak jika suatu node berhasil disusupi. Bagian-bagian berikut membahas praktik-praktik terbaik utama untuk mengatasi celah-celah tersebut.

Menerapkan Autentikasi Pesan

Protokol jaringan kendaraan lama (*legacy*) seperti CAN, LIN, dan FlexRay dirancang tanpa autentikasi pesan; artinya, setiap *node* pada bus dapat mengirimkan pesan apa pun tanpa verifikasi. Untuk mengatasi masalah keamanan ini, autentikasi perlu ditambahkan di atas lapisan protokol bawaan. Dengan mempertimbangkan kebutuhan bandwidth dan kinerja, autentikasi pesan sebaiknya hanya diterapkan pada pesan-pesan yang memiliki risiko keamanan tertinggi.

Untuk CAN dan LIN, *Secure Onboard Communication* (SecOC) menyediakan mekanisme standar untuk menambahkan *Message Authentication Code* (MAC) pada *Protocol Data Unit* (PDU) yang diamankan. Implementasi SecOC yang tepat harus menggunakan MAC yang cukup panjang untuk mencegah serangan tebakan (*guessing attacks*), menerapkan nilai kesegaran (*freshness values*) untuk mencegah serangan *replay*, serta membuang PDU yang gagal dalam verifikasi MAC artinya, PDU yang gagal verifikasi tidak boleh diteruskan ke aplikasi. Selain itu, kunci simetris yang digunakan untuk perhitungan MAC harus disediakan dan disimpan dengan

aman idealnya di dalam *Hardware Security Module* (HSM) dan tidak boleh dibagikan di antara kendaraan atau varian ECU yang berbeda, agar kompromi pada satu kunci tidak menjadi risiko bagi seluruh armada kendaraan. Untuk FlexRay, prinsip autentikasi yang sama juga berlaku. Perlindungan harus diterapkan di atas lapisan protokol, karena FlexRay tidak menyediakan mekanisme autentikasi bawaan. Prioritas harus diberikan pada slot yang membawa sinyal-sinyal krusial bagi keselamatan, seperti perintah kendali sasis dan sinyal *steer-by-wire*, di mana pemalsuan pesan dapat berdampak langsung terhadap keselamatan. Untuk Ethernet otomotif, autentikasi pada lapisan transpor ditangani melalui TLS, dengan ketentuan bahwa autentikasi timbal balik (*mutual authentication*) sebaiknya diterapkan di antara ECU yang berkomunikasi, guna memastikan kedua pihak memverifikasi identitas satu sama lain sebelum melakukan pertukaran data. Penjelasan lebih lanjut mengenai TLS dan pengendalian kerahasiaan data akan dibahas pada subbagian berikutnya.

Penerapan Enkripsi dan Keamanan Lapisan Transpor (*Transport Layer Security*)

Secara bawaan, semua protokol jaringan di dalam kendaraan mengirimkan data dalam bentuk teks polos (*plaintext*). Berbeda dengan CAN, LIN, dan FlexRay yang memiliki dukungan terbatas untuk enkripsi, pada *Automotive Ethernet*, penggunaan teks polos merupakan kesalahan konfigurasi, bukan keterbatasan protokol. Artinya, tersedia lebar pita (*bandwidth*) yang memadai untuk mendukung TLS, dan tidak ada alasan untuk menjalankan layanan diagnostik, pembaruan OTA (*over-the-air*), atau layanan yang berkaitan dengan keselamatan melalui koneksi yang tidak dienkripsi.

Oleh karena itu, TLS harus diterapkan pada layanan Ethernet yang menangani data sensitif, dengan mTLS (mutual TLS) diwajibkan untuk komunikasi antar-ECU guna memastikan kedua pihak saling melakukan autentikasi. Sertifikat ECU harus diterbitkan oleh Otoritas Sertifikat (*Certificate Authority*) tepercaya, mekanisme pencabutan sertifikat (*revocation*) harus ditegakkan, dan certificate pinning perlu dipertimbangkan untuk hubungan komunikasi yang telah ditetapkan sejak tahap perancangan. MACsec dapat melengkapi TLS dengan mengamankan tautan Ethernet individual pada lapisan tautan data (*data link layer*), meskipun mekanisme ini harus dipahami sebagai kontrol *hop-by-hop* (per segmen), bukan solusi *end-to-end* (dari ujung ke ujung). Perlu dipahami bahwa simpul perantara, seperti *gateway* yang mengakhiri sesi MACsec, memiliki akses penuh untuk melihat lalu lintas data dalam bentuk teks polos yang melewatinya. Pada protokol CAN klasik, enkripsi terkendala oleh batasan muatan (*payload*) sebesar 8 byte dan persyaratan waktu nyata (*real-time*). Oleh karena itu, pendekatan yang disarankan adalah menggunakan SecOC untuk autentikasi pada CAN, serta menggunakan Ethernet untuk kebutuhan yang memerlukan kerahasiaan data.

Penerapan Segmentasi Jaringan dan Kontrol Akses

Jaringan kendaraan yang bersifat datar (*flat*) dan tidak tersegmentasi dengan baik akan memperparah kelemahan pada tingkat protokol. Sebagai contoh, jika sistem infotainment pada arsitektur yang tidak tersegmentasi berhasil disusupi, penyerang jarak jauh dapat mengirim pesan secara langsung ke ECU yang krusial bagi keselamatan. Oleh karena itu, segmentasi jaringan sangat penting untuk membatasi dampak jika suatu simpul (*node*) berhasil

disusupi, dengan cara menetapkan batasan antara domain-domain yang memiliki tingkat kepercayaan dan tingkat kepentingan fungsional yang berbeda.

Arsitektur kendaraan harus mengelompokkan ECU ke dalam zona keamanan atau domain fungsional, di mana komunikasi antarzona diarahkan secara eksklusif melalui *gateway* pusat atau pengontrol zona (*zone controller*) yang menerapkan kebijakan kontrol akses secara tegas. Pada CAN, ini berarti *gateway* harus memiliki daftar putih (*whitelist*) ID pesan yang diizinkan untuk setiap batas domain, serta membuang pesan yang tidak tercantum dalam daftar tersebut. Pada Ethernet, kebijakan perutean antar-VLAN harus membatasi lalu lintas hanya pada layanan dan *port* tertentu yang diperlukan untuk operasional kendaraan yang sah. Sebagai contoh, akses ke antarmuka manajemen *switch* dan *gateway* harus dibatasi agar hanya dapat dijangkau dari VLAN manajemen khusus; hal ini bertujuan untuk mencegah *node* yang telah disusupi pada VLAN dengan tingkat kepercayaan lebih rendah agar tidak dapat berinteraksi secara langsung dengan antarmuka manajemen tersebut.

Selain itu, port trunk yang terhubung harus dikonfigurasi dengan VLAN native yang sama untuk mencegah VLAN hopping, dan kontrol akses berbasis port (IEEE 802.1X) harus diterapkan guna mencegah perangkat yang tidak sah memperoleh akses jaringan melalui koneksi fisik ke *port Ethernet* yang terbuka. Lebih lanjut, keamanan port (*port security*) harus dikonfigurasi pada *port switch Ethernet* untuk membatasi jumlah alamat MAC yang diizinkan per port, sehingga dapat mencegah serangan yang menyebabkan *overflow* pada tabel CAM.

Mengamankan Antarmuka Diagnostik

Antarmuka diagnostik seperti UDS melalui CAN (*UDS over CAN*), DoIP melalui Ethernet (DoIP over *Ethernet*), serta XCP melalui CAN maupun Ethernet, memberikan akses ke memori, konfigurasi, dan fungsi kontrol ECU. Jika tidak diamankan dengan tepat, antarmuka ini menjadi salah satu permukaan serangan (*attack surface*) dengan risiko tertinggi dalam jaringan kendaraan. Beberapa kontrol keamanan perlu diterapkan untuk mengatasi risiko-risiko tersebut. Mekanisme *Security Access* pada UDS harus diberlakukan sebelum memberikan akses ke sesi yang memiliki hak istimewa (*privileged sessions*), dan algoritma *seed-key* harus cukup kompleks untuk menangkal serangan *brute-force*. Oleh karena itu, disarankan untuk menggunakan algoritma kriptografi standar dengan panjang kunci yang memadai. Selain itu, autentikasi aktivasi routing pada DoIP harus diimplementasikan guna mencegah host yang tidak sah yang dapat menjangkau *gateway* untuk memulai sesi diagnostik yang menargetkan ECU di balik *gateway* tersebut. Sejalan dengan prinsip segmentasi jaringan yang dijelaskan pada subbagian sebelumnya, akses ke antarmuka diagnostik harus dibatasi berdasarkan segmen jaringan; misalnya, DoIP hanya boleh diakses melalui VLAN diagnostik khusus dan tidak dapat dijangkau dari domain lain, seperti domain sistem *infotainment*.

Terakhir, XCP harus dinonaktifkan dalam semua versi produksi (*production builds*). Membiarkan XCP tetap aktif dalam kondisi default tanpa autentikasi akan memberikan akses langsung dan tidak terbatas bagi node mana pun yang terhubung ke jaringan untuk membaca maupun menulis memori ECU. Risiko keamanan ini semakin diperbesar oleh kemampuan jangkauan jaringan XCPoE pada *Automotive Ethernet*. Dengan demikian, transisi dari tahap

pengembangan ke versi produksi harus mencakup langkah verifikasi eksplisit untuk memastikan bahwa XCP telah dinonaktifkan dengan benar.

Membangun Praktik Pengembangan dan Konfigurasi yang Aman

Beberapa kelemahan yang diidentifikasi dalam bab ini bukan muncul akibat keterbatasan desain protokol, melainkan karena kesalahan dalam konfigurasi maupun pengembangan. Sebagai contoh, fitur keamanan mungkin dinonaktifkan selama tahap pengembangan dan tidak diaktifkan kembali untuk versi produksi, atau terdapat kesalahan implementasi pada komponen perangkat lunak protokol.

Sangatlah penting untuk menanamkan aspek keamanan ke dalam proses pengembangan dengan mengikuti prinsip *security-by-design* (keamanan sejak tahap perancangan), alih-alih menambahkannya di akhir atau baru mempertimbangkannya saat pemantauan pasca pengembangan. Standar ISO/SAE 21434:2021 menyediakan kerangka kerja untuk membangun proses rekayasa keamanan siber di sepanjang siklus hidup pengembangan kendaraan, mulai dari fase konsep hingga tahap penonaktifan (*decommissioning*). Versi produksi harus diverifikasi secara eksplisit berdasarkan daftar periksa konfigurasi keamanan untuk memastikan bahwa XCP dinonaktifkan, SecOC diaktifkan pada semua ID pesan yang disyaratkan, *Security Access* diterapkan pada sesi UDS yang memiliki hak akses khusus, autentikasi aktivasi perutean DoIP telah dikonfigurasi, serta semua antarmuka yang hanya ditujukan untuk pengembangan telah dinonaktifkan. Selain konfigurasi, seluruh implementasi perangkat lunak protokol harus menerapkan validasi input yang ketat terhadap *payload* yang diterima mencakup validasi kolom panjang (*length field*), pemeriksaan tipe, dan pemeriksaan batas (*boundary checks*) mengingat kesalahan implementasi yang dapat menyebabkan *buffer overflow* pada *stack* protokol merupakan sumber umum kerentanan yang dapat dieksploitasi. Lebih lanjut, pengujian keamanan untuk komunikasi jaringan di dalam kendaraan termasuk pengujian fungsional keamanan, *fuzz testing*, dan *penetration testing* harus diintegrasikan ke dalam siklus hidup pengembangan, bukan sekadar diterapkan di tahap akhir, agar temuan yang ada dapat ditangani sedini mungkin.

Memungkinkan Pemantauan Berkelanjutan dan Deteksi Anomali

Kontrol keamanan preventif memang memperkecil area serangan, namun tidak dapat memberikan perlindungan menyeluruh. Penyerang yang berhasil mengakses jaringan di dalam kendaraan baik melalui ECU yang telah disusupi maupun kerentanan yang belum diketahui dapat beroperasi tanpa terdeteksi jika tidak ada sistem pemantauan. Oleh karena itu, *Intrusion Detection System* (IDS) di dalam kendaraan dapat mendeteksi aktivitas serangan serta anomali pada jaringan CAN dan Ethernet. Hal ini dilakukan dengan memantau ID pesan yang tidak terduga, anomali frekuensi transmisi, *frame* protokol yang tidak sesuai format, upaya sesi diagnostik yang tidak sah, serta penemuan layanan SOME/IP yang berasal dari sumber yang tidak dikenal. Peristiwa yang berkaitan dengan keamanan seperti kegagalan verifikasi SecOC, kegagalan akses keamanan UDS, penolakan aktivasi perutean DoIP, dan peringatan dari IDS harus dikumpulkan, dipusatkan, dan diteruskan ke *Vehicle Security Operations Center* (VSOC) di sisi *backend*. Di sana, peristiwa-peristiwa tersebut dapat dikorelasikan di seluruh armada kendaraan. Perlu dicatat bahwa pemantauan berkelanjutan harus dipandang sebagai langkah

pelengkap untuk mendeteksi hal-hal yang luput dari kontrol preventif, bukan sebagai pengganti penerapan langkah-langkah keamanan yang tepat seperti autentikasi, enkripsi, dan segmentasi jaringan sebagaimana dijelaskan dalam subbagian sebelumnya.

6.7 KEAMANAN DAN PERLINDUNGAN KOMUNIKASI JARINGAN INTERNAL KENDARAAN

Protokol jaringan dalam kendaraan (*in-vehicle network*) membentuk tulang punggung komunikasi pada kendaraan modern, yang memungkinkan koordinasi waktu nyata (*real-time*) antar-ECU, baik dalam *domain* yang krusial bagi keselamatan maupun domain lainnya. Sebagaimana telah ditunjukkan dalam bab ini, banyak dari protokol tersebut dirancang dengan mengutamakan keandalan dan determinisme, bukan keamanan. Kelemahan yang muncul seperti tidak adanya autentikasi, komunikasi dalam bentuk teks polos (*plaintext*), serta kontrol akses yang tidak memadai menjadi vektor serangan nyata yang dapat dimanfaatkan (dan memang telah dimanfaatkan) oleh pihak penyerang.

Bab ini telah mengkaji kelemahan keamanan pada protokol jaringan dalam kendaraan dan memetakannya ke dalam teknik pengujian penetrasi konkret yang dapat diterapkan oleh insinyur keamanan di lingkungan laboratorium pengujian maupun pada kendaraan secara langsung. Untuk memberikan pemahaman yang lebih mendalam mengenai penerapan teknik-teknik tersebut, bab ini juga mengulas beberapa studi kasus nyata di mana komunikasi jaringan dalam kendaraan berhasil disusupi. Studi kasus ini menunjukkan bagaimana kelemahan protokol dan kesalahan implementasi dapat dieksploitasi untuk memperoleh akses tidak sah serta mengganggu fungsi kendaraan. Perlu dicatat bahwa banyak kelemahan yang dibahas dalam bab ini bukanlah keterbatasan bawaan dari protokol itu sendiri, melainkan kesalahan dalam penerapan dan konfigurasi yang sebenarnya dapat diatasi oleh para insinyur serta arsitek keamanan. Praktik-praktik terbaik yang disajikan dalam bab ini menyediakan kerangka kerja berbasis prinsip untuk mengamankan komunikasi jaringan dalam kendaraan dari berbagai serangan yang telah dijelaskan. Setelah membahas keamanan komunikasi jaringan dalam kendaraan, bab berikutnya akan mengulas lebih dalam mengenai sistem-sistem yang terdapat di dalam kendaraan.

BAGIAN 3
MITOS KEAMANAN PADA ARSITEKTUR SISTEM KENDARAAN

Bagian ini membahas asumsi yang masih lazim di industri, yaitu bahwa lapisan-lapisan arsitektur kendaraan yang lebih dalam dianggap aman karena secara fisik tidak dapat diakses. Anda akan mengetahui bahwa ECU berbasis mikrokontroler, platform komputasi berkinerja tinggi, sensor kendaraan, dan sistem AI/ML semuanya memiliki celah serangan (*attack surface*) signifikan yang dapat dieksploitasi oleh pihak yang berniat jahat dan memiliki keahlian. Pembahasan dimulai dari serangan di tingkat perangkat keras, seperti *fault injection* (injeksi kesalahan) dan eksploitasi antarmuka *debug*, hingga eksploitasi perangkat lunak yang canggih yang menargetkan *hypervisor* serta alur inferensi pembelajaran mesin (*machine learning*).

Bab terakhir kemudian beralih ke strategi pertahanan yang berorientasi ke masa depan, dengan menguraikan bagaimana organisasi dapat menerapkan Sistem Manajemen Keamanan Siber (*Cybersecurity Management System* atau CSMS) dan prinsip *secure-by-design* (keamanan sejak tahap perancangan) dalam siklus hidup rekayasa yang aman dan berbasis AI, guna mencapai ketahanan jangka panjang di sepanjang siklus hidup kendaraan.

Bagian ini mencakup bab-bab berikut:

- Bab 7, “Meretas ECU Berbasis Mikrokontroler”
- Bab 8, “Meretas ECU Berkinerja Tinggi”
- Bab 9, “Meretas Sensor Kendaraan dan Sistem AI/ML”
- Bab 10, “Langkah ke Depan”

BAB 7
KEAMANAN DAN EKSPLOITASI ECU BERBASIS MIKROKONTROLER (MCU)

Meskipun terdapat pergeseran ke arah sistem komputasi terpusat, unit kontrol elektronik (ECU) yang dibangun di atas mikrokontroler (MCU) tetap menjadi komponen utama untuk fungsi penginderaan dan aktuasi pada kendaraan. Berbeda dengan *system-on-chip* (SoC) berkinerja tinggi yang ditemukan dalam komputer kendaraan dan sistem bantuan pengemudi tingkat lanjut, ECU berbasis MCU tradisional memanfaatkan sumber daya yang terbatas untuk menjalankan kontrol *real-time* yang deterministik. MCU ini biasanya mengintegrasikan satu atau beberapa inti CPU dengan memori dan periferal internal yang terbatas, yang dioptimalkan untuk berinteraksi langsung dengan berbagai sensor dan aktuator kendaraan. Perangkat ini menjalankan sistem operasi *real-time* (RTOS) yang ringan atau bahkan beroperasi tanpa sistem operasi (*bare metal*), berbeda dengan sistem operasi yang lebih kompleks pada SoC otomotif. Anda dapat menemukan ECU berbasis mikrokontroler di seluruh bagian kendaraan modern; perangkat ini mengelola fungsi-fungsi spesifik seperti kontrol mesin, sistem pengereman anti-terkunci (*anti-lock braking*), pengembangan kantung udara (*airbag*), *power steering*, dan kontrol bodi kendaraan. Banyak dari fungsi ini bersifat krusial bagi keselamatan, sehingga unit kontrol harus merespons berbagai peristiwa dalam hitungan milidetik, sembari tetap beroperasi di bawah batasan ketat terkait memori, komputasi, keandalan, dan keselamatan.

Secara historis, MCU otomotif tidak dirancang dengan mempertimbangkan adanya pihak yang berniat jahat. Seiring dengan meningkatnya konektivitas kendaraan dan beralihnya perhatian penyerang ke jaringan internal kendaraan, ECU yang memiliki keterbatasan sumber daya ini pun menjadi target serangan. Bergantung pada arsitektur kelistrikan/elektronik (E/E) kendaraan, ECU tertanam dapat diakses secara langsung melalui jaringan atau antarmuka fisik, atau mungkin terletak di lapisan yang lebih dalam dalam arsitektur kendaraan tersebut. Dalam kedua kasus tersebut, penyerang yang ingin memengaruhi perilaku kendaraan akan berupaya untuk menyusupi ECU tertanam, baik secara langsung maupun tidak langsung.

Selain itu, mempelajari arsitektur ECU berbasis MCU tidak hanya berguna untuk menembus ECU tertanam secara mendalam, tetapi juga penting ketika mempelajari keamanan HPC (seperti yang akan kita lihat di bab berikutnya). Sangat umum bagi sistem semacam itu untuk mengintegrasikan MCU tertanam, misalnya untuk menangani tugas-tugas keselamatan waktu nyata (*real-time*), seperti dalam domain keselamatan tertanam (*embedded safety domain*), atau berfungsi sebagai pendamping SoC yang utamanya berperan sebagai pengawas (*watchdog*) dan sistem cadangan (*fail-over*) bagi keseluruhan sistem. MCU ini menjalankan tumpukan perangkat lunak (*software stack*) yang sama dengan ECU tertanam yang berdiri sendiri, sehingga strategi serangan yang digunakan terhadap ECU tersebut dapat diterapkan kembali untuk menyerang HPC (*High-Performance Computer*). Selain berdampak langsung pada kendali kendaraan, serangan terhadap ECU semacam ini dapat berujung pada perolehan

aset keamanan yang berguna untuk eksploitasi yang lebih canggih seperti mendapatkan kunci kriptografi jangka panjang untuk jaringan di dalam kendaraan, atau kredensial yang digunakan untuk terhubung ke infrastruktur pengisian daya kendaraan.

Dalam bab ini, kita akan mendalami ranah keamanan ofensif yang menyasar ECU berbasis MCU. Kita akan mulai dengan tinjauan mengenai arsitektur perangkat keras dan tumpukan perangkat lunak yang umum digunakan. Selanjutnya, kita akan mengkaji kelemahan-kelemahan yang sering ditemukan, baik pada perangkat keras maupun perangkat lunaknya. Kemudian, kita akan menguraikan metodologi untuk menyusun rantai serangan (*attack chains*) terhadap perangkat dengan sumber daya terbatas ini mulai dari tahap pengintaian awal dan ekstraksi *firmware*, hingga eksploitasi dan upaya untuk mempertahankan akses (*persistence*). Beberapa studi kasus akan disajikan untuk mengilustrasikan insiden peretasan ECU berbasis MCU di dunia nyata beserta dampaknya. Terakhir, kita akan membahas praktik terbaik keamanan dan langkah-langkah pertahanan untuk menangkal serangan yang paling umum, dengan menekankan pentingnya pendekatan keamanan berlapis. Di akhir bab ini, Anda akan memperoleh wawasan mengenai teknik-teknik yang digunakan untuk meretas ECU berbasis MCU, serta cara untuk meminimalkan efektivitas serangan tersebut. Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

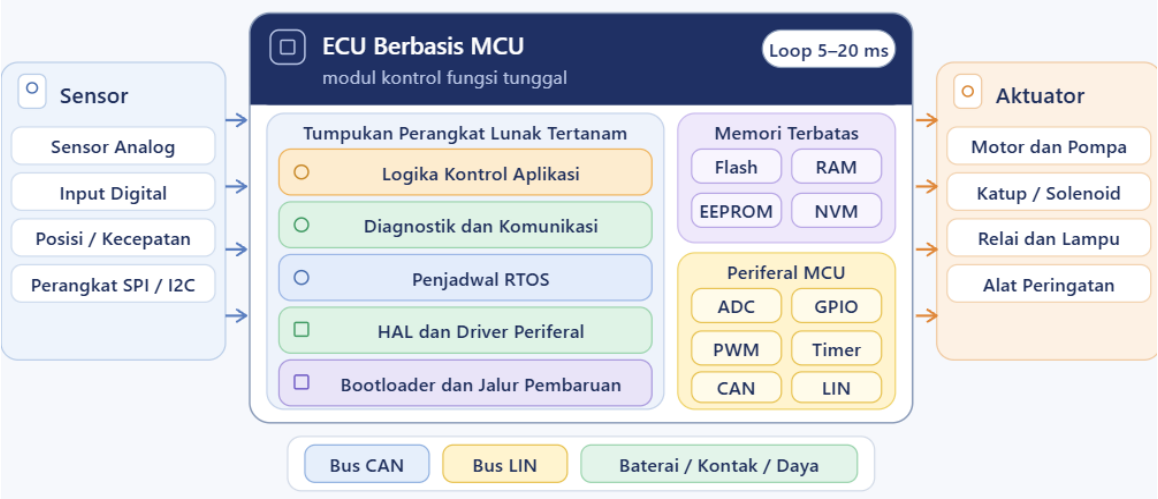
- Gambaran umum ECU berbasis MCU
- Memahami anatomi perangkat keras dan perangkat lunak
- Kelemahan perangkat keras dan perangkat lunak yang umum pada ECU berbasis MCU
- Teknik pengujian penetrasi dan penyusunan rantai serangan (*attack-chain*)
- Studi kasus kompromi pada ECU berbasis MCU
- Praktik terbaik keamanan untuk ECU tertanam (*embedded ECU*) yang Tangguh

7.1 ARSITEKTUR DAN KARAKTERISTIK ECU BERBASIS MIKROKONTROLER (MCU)

Sebelum membahas teknik serangan terhadap ECU berbasis MCU, kita perlu memahami arsitektur sistem tersebut serta aspek-aspek unik yang menjadikannya target sekaligus membuatnya rentan terhadap serangan. MCU otomotif modern sebagian besar merupakan mikrokontroler 32-bit yang dirancang untuk eksekusi waktu nyata (*real-time*) yang krusial bagi keselamatan. Contoh arsitektur MCU mencakup Infineon TriCore (yang ditemukan dalam keluarga AURIX) dan Renesas RH850, yang mendukung kombinasi antara inti CPU eksklusif (*proprietary*) dan inti yang tersedia secara umum, seperti keluarga ARM Cortex-R serta RISC-V. Perangkat-perangkat ini sering kali mengadopsi arsitektur Harvard dengan memori yang terintegrasi erat (*tightly coupled memory*) untuk menghilangkan latensi akses yang tidak dapat diprediksi dan memastikan waktu eksekusi yang konsisten. MCU yang dirancang untuk aplikasi kritis terhadap keselamatan (*safety-critical*) menyertakan berbagai mekanisme keamanan, seperti eksekusi *lockstep*, *watchdog timer*, perlindungan *error-correcting code* (ECC) untuk memori, serta rutinitas pengujian mandiri (*built-in self-test*). Tujuan dari desain ini adalah untuk memastikan bahwa kesalahan seperti kegagalan perangkat

keras acak atau perubahan bit (*bit flip*) sementara dapat dideteksi dengan cepat, sehingga sistem dapat beralih ke kondisi aman yang telah ditentukan.

Perangkat keras pelindung memori juga sering diintegrasikan untuk memisahkan komponen perangkat lunak yang memiliki tingkat kepentingan keselamatan berbeda. Biasanya, *Memory Protection Unit* (MPU) terdistribusi menyediakan berbagai konteks perlindungan yang membatasi akses komponen perangkat lunak tertentu terhadap wilayah memori tertentu. Mekanisme ini membantu mencegah kode yang tidak semestinya menimpa data yang krusial bagi keselamatan atau mengubah logika kendali. Saat ini, banyak MCU yang menggunakan arsitektur multi-inti (*multi-core*), contohnya perangkat AURIX TC3xx yang mengintegrasikan enam inti CPU dengan kecepatan hingga 300 MHz, serta mampu menghasilkan *throughput* lebih dari 4000 DMIPS untuk beban kerja waktu nyata (*real-time*). Tingkat kinerja ini diperlukan untuk menjalankan fungsi kendali yang spesifik tujuan utama dari ECU ini sekaligus memastikan bahwa tenggat waktu yang ketat (*hard timing deadlines*) untuk peristiwa seperti penyalaan mesin, siklus pengereman, atau kendali torsi dapat selalu terpenuhi. Gambar 7.1 menampilkan contoh ECU berbasis MCU beserta berbagai sensor, aktuator, dan antarmuka jaringan yang didukungnya.



Gambar 7.1 Tampilan sederhana komponen perangkat keras dan perangkat lunak ECU berbasis MCU

ECU berbasis MCU tradisional, sebagaimana ditunjukkan dalam gambar, memiliki beberapa karakteristik utama. Perangkat ini biasanya beroperasi dengan sumber daya memori yang terbatas, sering kali hanya beberapa megabita untuk memori *flash* dan RAM. ECU dirancang untuk eksekusi yang deterministik, dengan waktu siklus kendali (*control-loop timing*) berkisar antara 5 hingga 20 milidetik. Selain itu, perangkat ini menyediakan antarmuka elektrik langsung ke sensor dan aktuator termasuk input analog dan digital serta output *Pulse Width Modulation* (PWM) dan memiliki dukungan bawaan untuk jaringan dalam kendaraan (*in-vehicle networks*) seperti CAN dan LIN. Antarmuka ini memungkinkan ECU berfungsi sebagai sistem tertanam (*embedded system*) mandiri yang bertanggung jawab atas tugas spesifik, seperti pengaturan waktu percikan api mesin, pengendalian campuran udara-bahan bakar,

modulasi tekanan pengereman, manajemen pengisian daya baterai, bantuan kemudi, logika pengembangan kantung udara (*airbag*), serta pengaturan iklim kabin. Karena sistem-sistem ini memengaruhi perilaku fisik kendaraan secara langsung, kegagalan atau kesalahan dalam sistem dapat berdampak langsung dan nyata terhadap operasional maupun keselamatan.

Mengingat perannya yang spesifik, komponen-komponen ini biasanya tidak memerlukan kecepatan clock dalam skala multi-gigahertz ataupun memori berkapasitas gigabyte. Sebaliknya, prioritas utamanya adalah konsistensi waktu (*timing*), toleransi terhadap kesalahan (*fault tolerance*), dan keselamatan fungsional (*functional safety*). Filosofi desain ini berbeda dengan SoC (*System-on-Chip*) otomotif berkinerja tinggi seperti yang digunakan untuk sistem hiburan (*infotainment*) atau komputer pusat kendaraan yang akan kita bahas pada bab berikutnya. Meskipun platform tersebut menawarkan kapasitas komputasi total yang jauh lebih besar, platform tersebut umumnya tidak mampu memberikan determinisme respons dalam hitungan milidetik, yang justru dapat disediakan oleh MCU yang dirancang untuk kendali waktu nyata (*real-time control*). MCU otomotif modern sering kali menyertakan koprosesor khusus, seperti modul keamanan perangkat keras (*hardware security module*), untuk menangani operasi kriptografi. Hal ini mencerminkan kesadaran yang kian berkembang bahwa persyaratan keamanan siber harus ditangani bersamaan dengan aspek keselamatan fungsional yang konvensional.

Dari sisi perangkat lunak, MCU biasanya menjalankan sistem operasi waktu nyata (RTOS) khusus. Standar industri yang telah lama digunakan adalah OSEK/VDX beserta penerusnya, AUTOSAR OS, yang menyediakan RTOS dengan konfigurasi statis, penjadwalan prioritas tetap, dan penanganan interupsi yang dirancang demi prediktabilitas. AUTOSAR OS diterapkan sebagai bagian dari AUTOSAR Classic Platform, yang menstandardisasi arsitektur perangkat lunak untuk berbagai ECU produksi. Standardisasi ini menguntungkan para insinyur dengan meningkatkan interoperabilitas dan penggunaan kembali kode; namun, seperti yang akan kita lihat nanti dalam bab ini, hal tersebut juga memberikan akses kepada pihak penyerang terhadap spesifikasi yang tersedia untuk umum, yang mendokumentasikan layanan, antarmuka, dan perilaku diagnostik yang didukung.

Sebagai alternatif, pemasok ECU dapat memilih solusi *bare-metal* yang mengandalkan RTOS ringan seperti FreeRTOS atau varian yang lebih aman, yaitu SafeRTOS. Terlepas dari implementasinya, sistem operasi ini hanya menyediakan layanan esensial seperti penjadwalan tugas, penghitung waktu (*timing counters*), dan perlindungan memori. Sistem ini sengaja menghindari kompleksitas serta fitur-fitur yang lazim ditemukan pada sistem operasi serba guna seperti Linux. Akibatnya, perangkat lunak aplikasi umumnya ditulis dalam bahasa C dengan batasan yang mengacu pada pedoman MISRA C. Tumpukan perangkat lunak (*software stack*) yang dihasilkan bersifat terbatas, dirancang untuk tujuan spesifik, dan sangat berbeda dari lingkungan multi-aplikasi yang ditemukan pada SoC otomotif.

Secara historis, ketiadaan antarmuka pengguna langsung dan konektivitas eksternal membuat banyak insinyur berasumsi bahwa ECU semacam itu pada dasarnya terisolasi dari ancaman jarak jauh. Bagian selanjutnya dari bab ini akan menantang asumsi tersebut dengan mengkaji bagaimana penyerang dapat berhasil menyasar sistem-sistem yang tampak berdiri

sendiri ini. Untuk itu, kita perlu menelaah lebih dalam anatomi perangkat keras maupun perangkat lunak dari sistem-sistem tersebut.

7.2 ANATOMI PERANGKAT KERAS DAN PERANGKAT LUNAK ECU BERBASIS MCU

Melakukan eksploitasi yang berhasil pada mikrokontroler otomotif memerlukan pemahaman mengenai anatomi tingkat rendahnya, baik dari sisi perangkat keras maupun perangkat lunak. Pada bagian ini, kami membedah komponen-komponen perangkat keras yang lazim ditemukan dalam ECU berbasis mikrokontroler (permukaan serangan dari perspektif perangkat keras) serta struktur perangkat lunak dan *firmware* yang dijalankannya (permukaan serangan dari perspektif perangkat lunak). Pengetahuan ini akan menjadi landasan untuk menyusun eksploitasi-eksploitasi tertentu.

Komponen Perangkat Keras Pada MCU Otomotif

ECU otomotif konvensional terdiri dari papan sirkuit cetak (*printed circuit board*), sebagaimana ditunjukkan pada Gambar 7.2, yang memuat setidaknya satu mikrokontroler beserta komponen elektronik pendukung, semuanya ditempatkan di dalam sebuah rumah (*casing*) yang kokoh. Saat rumah ECU tersebut dibuka, akan terlihat serangkaian komponen perangkat keras yang relatif konsisten.



Gambar 7.2 Tampilan PCB tipikal dari ECU berbasis MCU

Pada bagian-bagian berikut, kita akan meninjau komponen-komponen yang paling umum ditemukan dalam ECU berbasis mikrokontroler (MCU).

Mikrokontroler

Mikrokontroler merupakan sirkuit terpadu (IC) utama pada papan sirkuit yang berfungsi sebagai mesin eksekusi untuk semua perangkat lunak waktu nyata (*real-time*) yang berjalan di dalam ECU. MCU otomotif umumnya mencakup satu atau lebih inti CPU yang mendukung berbagai mode eksekusi, seperti mode istimewa (*privileged mode*) dan mode pengguna (*user mode*), serta dilengkapi dengan perlindungan arsitektural seperti MPU (*Memory Protection Unit*). Meskipun arsitektur inti yang mendasarinya memengaruhi detail pengembangan eksploit seperti apakah target menggunakan TriCore, ARM, atau set instruksi CPU lainnya dari sudut pandang penyerang, inti MCU pada dasarnya adalah komponen yang

bertanggung jawab untuk mengeksekusi kode yang disisipkan atau dimanipulasi. Karakteristik seperti *endianness*, ukuran *word*, dan keberadaan instruksi khusus termasuk primitif kriptografi yang dipercepat oleh perangkat keras menjadi relevan saat menyusun *shellcode* atau *payload* untuk *return-oriented programming* (ROP) yang disesuaikan dengan keluarga MCU tertentu.

Memori (Flash, RAM, EEPROM)

Mikrokontroler otomotif biasanya mengintegrasikan memori *flash* di dalam cip (*on-chip*) yang digunakan untuk menyimpan *bootloader*, *firmware* aplikasi, dan parameter kalibrasi. Kapasitasnya bervariasi, mulai dari beberapa ratus kilobita pada perangkat kelas bawah hingga beberapa megabita pada MCU kelas atas. *Static Random-Access Memory* (SRAM) internal digunakan untuk menangani data saat waktu berjalan (*runtime*) dan umumnya memiliki kapasitas yang lebih terbatas, sering kali berkisar antara beberapa ratus kilobita hingga beberapa megabita. Selain memori internal, ECU mungkin dilengkapi dengan EEPROM serial eksternal atau perangkat *flash* eksternal untuk menyimpan data kalibrasi khusus atau parameter personalisasi kendaraan. Saat mengidentifikasi permukaan serangan (*attack surface*) pada ECU berbasis MCU, kita biasanya memberikan perhatian khusus pada antarmuka memori eksternal, karena komponen ini sering menjadi target serangan yang efektif untuk mengekstrak *firmware* dan data kalibrasi.

Antarmuka dan Bus Komunikasi

Mikrokontroler terhubung ke jaringan kendaraan melalui berbagai periferal komunikasi. Yang paling umum adalah *Controller Area Network* (CAN), yang diimplementasikan melalui pengontrol CAN di dalam cip (*on-chip*) dan *transceiver* eksternal. Banyak ECU mendukung beberapa saluran CAN; sebagai contoh, ECU dapat terhubung secara bersamaan ke jaringan *powertrain* (sistem penggerak) dan jaringan kontrol bodi kendaraan, yang mencakup antarmuka *Onboard Diagnostic* (OBD). Antarmuka tambahan sering kali mencakup LIN untuk komunikasi sensor dan aktuator berkecepatan rendah, FlexRay pada beberapa sistem kelas atas versi lama, serta Ethernet otomotif yang kini semakin banyak digunakan untuk berkomunikasi dengan pengontrol domain (*domain controller*) dan aplikasi komputasi terpusat.

Dari sudut pandang ofensif, antarmuka-antarmuka ini menentukan permukaan serangan yang memungkinkan terjadinya serangan lokal maupun jarak jauh. Sebagai contoh, seperti yang telah kita lihat, sebuah ECU yang terhubung ke port OBD-II dapat diakses melalui lalu lintas CAN yang disuntikkan dengan menggunakan protokol *Unified Diagnostic Service* (UDS). CAN dan Ethernet semakin memperluas permukaan serangan dengan menyediakan kapabilitas yang sensitif terhadap keamanan, seperti akses baca/tulis memori dan pemrograman ulang *firmware*. Selain jaringan di dalam kendaraan, beberapa MCU juga menghadirkan antarmuka UART atau USB saat digunakan dalam *gateway* dan modul kontrol bodi (*body control modules*); antarmuka ini dapat menjadi titik masuk jika tidak dilindungi dengan memadai.

I/O Langsung dan Antarmuka Sensor

Di luar bus komunikasi, MCU berinteraksi langsung dengan lingkungan fisik melalui I/O analog dan digital. Ini mencakup *Analog-to-Digital Converter* (ADC) untuk membaca sinyal sensor, input digital untuk sakelar atau sinyal modulasi lebar pulsa (*pulse-width modulated*/PWM), serta output yang digunakan untuk mengendalikan aktuator seperti motor servo. Meskipun antarmuka ini mungkin tampak tidak terlalu berkaitan dengan masalah keamanan siber, perannya menjadi sangat krusial begitu penyerang berhasil mengeksekusi kode arbitrer pada MCU. Kendali atas jalur I/O ini memungkinkan penyerang untuk menimbulkan dampak fisik, seperti memalsukan nilai sensor atau menyalahgunakan aktuator.

Periferal Internal

MCU mengintegrasikan berbagai periferal di dalam chip, termasuk pengatur waktu (*timer*) presisi tinggi, watchdog, pengontrol komunikasi, ADC, unit PWM, dan akselerator kriptografi. Postur keamanan ECU secara keseluruhan sangat bergantung pada bagaimana periferal ini dikonfigurasi dan dikendalikan. Sebagai contoh, watchdog timer dirancang untuk mengatur ulang (*reset*) sistem jika eksekusi perangkat lunak terhenti; namun, penyerang dapat mengeksploitasi fitur ini untuk memicu reset yang tidak diinginkan atau menonaktifkannya guna mempertahankan akses (*persistensi*).

Banyak MCU modern juga menyertakan *Hardware Protected Security Environment* (HPSE) yang umumnya selaras dengan standar SAE J3101 untuk mengisolasi kunci dan operasi kriptografi dari inti aplikasi utama. Jika diintegrasikan dengan benar, modul-modul ini dapat menjadi aset pertahanan yang kuat; sebaliknya, jika dirancang atau diperkuat dengan buruk, modul tersebut justru bisa menjadi target menarik bagi penyerang untuk mengekstrak kunci kriptografi. Memahami fitur keamanan apa saja yang tersedia, bagaimana kunci disediakan (*provisioning*), serta apakah perlindungan tersebut bergantung pada rahasia yang unik atau rahasia bawaan (*default*), merupakan langkah pengintaian (*reconnaissance*) yang krusial bagi setiap penyerang.

Port Debug dan Pemrograman

Antarmuka seperti *Joint Test Action Group* (JTAG) atau *Serial Wire Debug* (SWD) sangat diperlukan selama tahap pengembangan, karena memungkinkan teknisi untuk menghentikan eksekusi, memeriksa memori, dan memprogram memori *flash*. MCU menyediakan *debugger* di dalam cip (*on-chip debugger*) untuk mempercepat pengembangan dan memecahkan masalah yang sulit direproduksi. MCU modern biasanya menyertakan mekanisme penguncian untuk antarmuka *debug*, baik melalui kata sandi maupun mekanisme *challenge-response* kriptografis, yang umumnya ditegakkan oleh HPSE (*High-Privilege Security Environment*).

Pada sistem produksi, antarmuka ini seharusnya dinonaktifkan atau dibatasi; namun, kegagalan dalam menonaktifkan antarmuka tersebut dengan benar bukanlah hal yang jarang terjadi. Port *debug* yang terbuka atau tidak terkunci dengan semestinya memberikan kendali langsung atas mikrokontroler, sehingga memungkinkan ekstraksi atau modifikasi *firmware* dengan upaya minimal. Bahkan ketika akses *debug* telah dinonaktifkan, banyak MCU tetap mendukung mode pemrograman alternatif atau *bootloader* yang memantau perintah pembaruan *firmware* melalui antarmuka CAN atau serial.

Keamanan mekanisme ini bergantung pada ketangguhan mekanisme autentikasi dan kontrol akses. Penggunaan kredensial yang lemah atau bersifat umum (*global*), serta penyediaan jalur pembaruan yang tidak dilengkapi dengan perlindungan kriptografis, dapat melemahkan desain yang sebenarnya sudah kuat. Sebagaimana akan kita lihat pada bagian mengenai kelemahan perangkat keras, penyalahgunaan antarmuka debug dan pemrograman tetap menjadi salah satu prasyarat paling umum untuk menyusun eksploitasi pada ECU.

Pengantar singkat mengenai permukaan serangan perangkat keras ECU ini diharapkan dapat memotivasi para pakar keamanan ofensif untuk mendalami komponen perangkat keras dalam sistem tersebut, termasuk kemasan fisik dan PCB, memori internal maupun eksternal, antarmuka jaringan, jalur I/O langsung, serta titik akses untuk kebutuhan debug atau pemrograman. Pengintaian (*reconnaissance*) yang cermat terhadap setiap elemen tersebut merupakan langkah krusial untuk melancarkan serangan berbasis perangkat keras yang sukses. Setelah memahami blok-blok penyusun perangkat keras yang utama, kini kita akan beralih membahas anatomi perangkat lunak dari sistem target kita.

Arsitektur Perangkat Lunak dan Model Eksekusi

Seperti yang telah dibahas sebelumnya, perangkat lunak yang berjalan pada MCU otomotif dibentuk oleh batasan-batasan perangkat keras dan mengikuti arsitektur berlapis yang khas. Memahami struktur dan cara eksekusi perangkat lunak ini sangat penting untuk mengidentifikasi kelemahan yang dapat dieksploitasi. Pada sebagian besar sistem berbasis MCU, kita akan menjumpai *flash bootloader* serta tumpukan aplikasi perangkat lunak yang memanfaatkan sistem operasi waktu nyata (*Real-Time Operating System* atau RTOS).

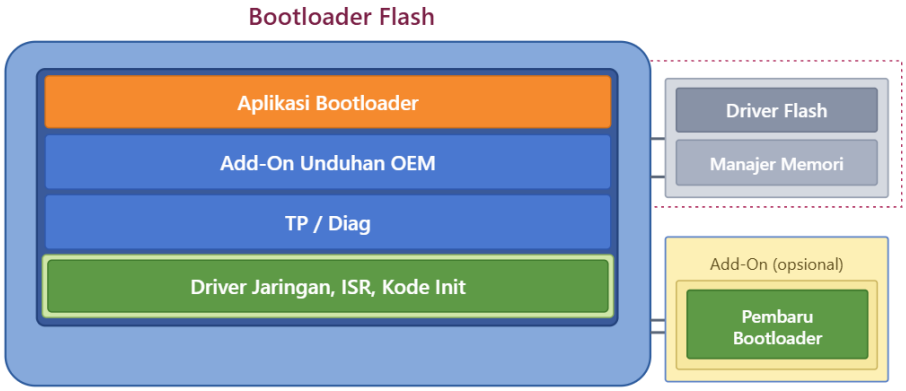
Meskipun banyak ECU yang beredar saat ini masih mengandalkan arsitektur perangkat lunak propietary (milik vendor tertentu), sistem-sistem tersebut tetap memiliki banyak komponen dasar yang serupa dengan standar AUTOSAR Classic seperti adanya RTOS, lapisan abstraksi perangkat keras (*hardware abstraction layer*), *middleware* untuk pengelolaan memori dan diagnostik, serta lapisan antarmuka aplikasi yang memfasilitasi fungsi penginderaan, aktuasi, dan berbagai fungsi kontrol lainnya. Mengingat keterbukaan dan kematangan spesifikasi AUTOSAR, kita akan menggunakan AUTOSAR sebagai model arsitektur perangkat lunak acuan dalam pembahasan ini, dengan catatan bahwa pengamatan serupa pada umumnya juga berlaku untuk sistem tertanam (*embedded systems*) yang tidak berbasis AUTOSAR. Namun, sebelum kita mendalami arsitektur perangkat lunak aplikasi, mari kita cermati komponen yang sering kali luput dari perhatian namun sangat vital bagi setiap ECU: *flash bootloader*.

Bootloader dan Proses Booting

ECU berbasis MCU sering kali mengeksekusi kode secara langsung dari memori *flash* internal, alih-alih memuatnya ke dalam RAM. Saat daya diberikan atau terjadi reset, eksekusi dimulai dari *bootloader* utama yang biasanya tersimpan di area memori *flash* yang terlindungi. Perlindungan ini dapat diterapkan melalui mekanisme penguncian *flash* berbasis perangkat keras atau pemeriksaan perangkat lunak yang mencegah pembaruan tidak sah pada area memori *bootloader* tersebut. Kode awal ini melakukan inisialisasi dasar mikrokontroler seperti konfigurasi *clock, watchdog timer*, dan periferal komunikasi penting (misalnya CAN) sebelum

menyerahkan kendali kepada aplikasi utama. *Bootloader* secara khusus mengelola periferal yang memungkinkannya melakukan pembaruan perangkat lunak, yang merupakan fungsi utamanya.

Bootloader umumnya menyediakan rutin *Unified Diagnostic Services* (UDS) untuk mendukung pembaruan *firmware*, termasuk layanan seperti *Request Download* (0x34), *Transfer Data* (0x36), dan *Request Transfer Exit* (0x37). Sebagaimana ditunjukkan pada Gambar 7.3, *firmware bootloader* biasanya mencakup driver perangkat keras untuk pengaturan dan inisialisasi, driver komunikasi, protokol transport ringan beserta tumpukan perangkat lunak diagnostik, tumpukan kriptografi ringan, *driver flash*, serta logika aplikasi *bootloader* yang mengatur proses pembaruan perangkat lunak.



Gambar 7.3 Contoh arsitektur *flash bootloader*

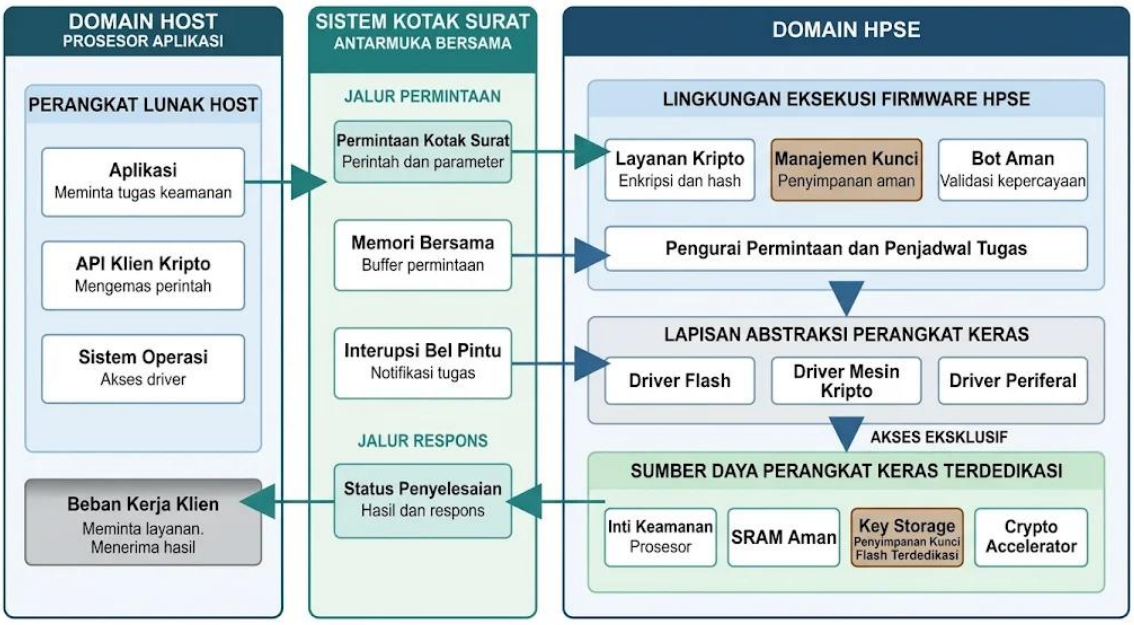
Selain mengelola pembaruan perangkat lunak, banyak ECU mengandalkan *flash bootloader* untuk menangani *secure boot* (proses *booting* yang aman) bagi aplikasi perangkat lunak. Dalam hal ini, *flash bootloader* memverifikasi tanda tangan digital atau kode autentikasi pesan (*message authentication code*) pada perangkat lunak aplikasi dan data kalibrasi sebelum menyerahkan kendali kepada aplikasi tersebut. Pada desain yang lebih baru, HPSE (*Hardware-based Security Element*) tertanam dapat berpartisipasi dalam urutan *booting* dengan memverifikasi integritas *firmware*-nya sendiri, *firmware* dari *bootloader* utama, serta perangkat lunak aplikasi secara bertahap, guna memenuhi batasan waktu saat memulai sistem (*startup timing constraints*). Sebagaimana akan dibahas pada bagian selanjutnya, kelemahan dalam desain *bootloader* sering kali menjadi celah bagi penyerang untuk memasang kode yang tidak sah ke dalam ECU.

Firmware HPSE

Banyak MCU modern menawarkan HPSE untuk mendukung berbagai kasus penggunaan keamanan, seperti *secure boot* (proses *booting* yang aman), komunikasi aman, autentikasi kriptografis, dan pembaruan perangkat lunak yang aman. Inti dari HPSE adalah prosesor keamanan khusus yang memiliki akses eksklusif ke berbagai sumber daya seperti akselerator kriptografis, SRAM aman, serta wilayah memori *flash* khusus yang berfungsi untuk menyimpan kunci kriptografis. HPSE menjalankan *firmware* yang, berdasarkan rancangannya, dimaksudkan untuk menjadi basis kode tepercaya (*trusted code base*).

Biasanya, *firmware* HPSE memuat penjadwal sederhana atau RTOS (sistem operasi waktu nyata) yang ringan. Lapisan abstraksi perangkat keras (*hardware abstraction layer*) memungkinkan fungsi-fungsi seperti pengelolaan *flash*, driver mesin kriptografi, dan driver periferal lainnya yang berada di bawah kendali khusus HPSE. Pada batas HPSE terdapat antarmuka komunikasi bersama yang memungkinkan HPSE untuk bertukar data secara aman dengan perangkat lunak aplikasi melalui register khusus atau buffer memori bersama yang telah ditentukan.

Sistem "kotak surat" (*mailbox*) ini memungkinkan pihak-pihak yang berkomunikasi untuk saling menukar perintah atau penunjuk (*pointer*) ke memori bersama yang berisi parameter tugas kriptografi. Mekanisme interupsi biasanya berfungsi layaknya bel pintu untuk memberi tahu setiap titik akhir (*endpoint*) bahwa ada data baru yang tiba dan perlu diproses, atau bahwa suatu respons telah siap. Dalam konteks sistem berbasis AUTOSAR, prosesor aplikasi menjalankan driver kripto untuk berkomunikasi dengan *firmware* eksklusif HPSE. Arsitektur ini diilustrasikan pada Gambar 7.4, yang memperlihatkan pemisahan antara prosesor aplikasi host dan domain HPSE.



Gambar 7.4 Antarmuka *firmware* HPSE ke prosesor aplikasi

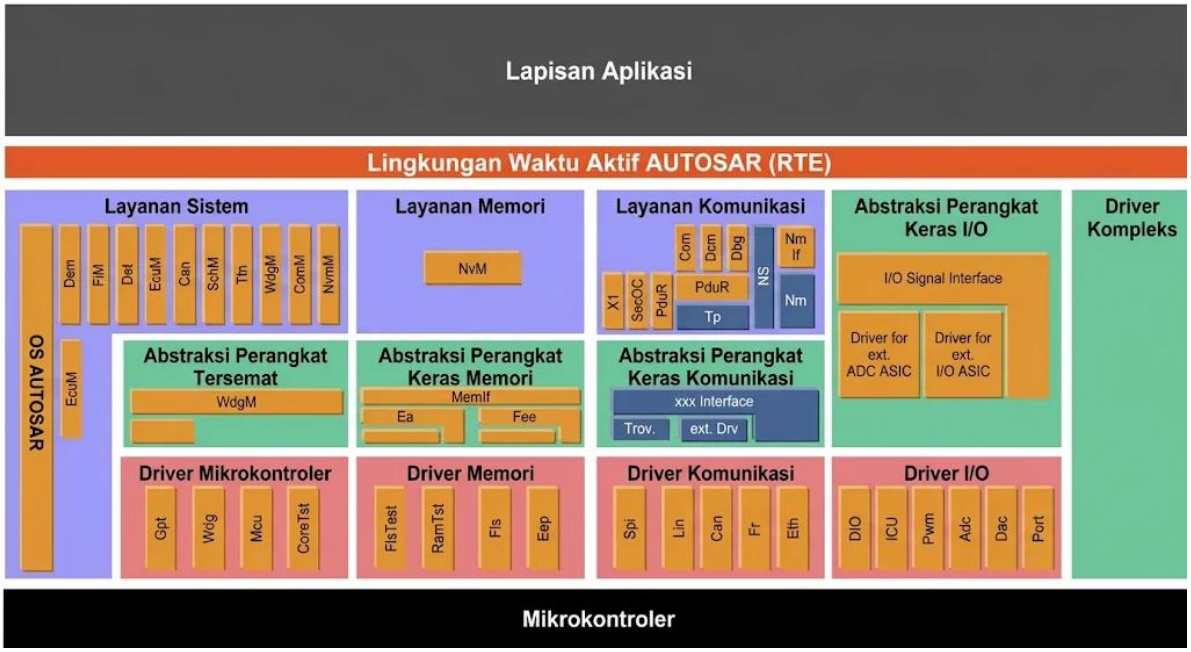
Tentu saja, seorang penyerang akan tertarik pada protokol komunikasi memori bersama (*shared memory*) antara aplikasi dan *firmware* HPSE untuk mengidentifikasi potensi kelemahan keamanan yang dapat dieksploitasi. Kemampuan untuk menyalahgunakan protokol ini dapat berujung pada pemeriksaan informasi tugas kriptografis, penyisipan kode ke dalam lingkungan eksekusi HPSE, atau bahkan pencurian kunci kriptografis. Risiko utamanya berkisar pada penguraian (*parsing*) permintaan yang tidak valid, termasuk penanganan yang keliru terhadap kolom panjang tugas, pengidentifikasi perintah, struktur data bersarang, atau *offset* (seperti penunjuk/pointer) ke dalam memori bersama. Komunikasi melalui memori

bersama juga menghadirkan risiko *time-of-check versus time-of-use* (TOCTOU), di mana *host* memodifikasi *buffer* setelah proses validasi namun sebelum digunakan oleh prosesor HPSE.

Mengingat *firmware* HPSE kemungkinan menggunakan penjadwal tugas (*job scheduler*), penyalahgunaan *state machine* (mesin status) menjadi target menarik lainnya seperti memanggil layanan dengan hak akses khusus (*privileged services*) saat berada dalam status siklus hidup yang tidak terduga, atau mencampuradukkan konteks sesi di antara berbagai permintaan. Kontrol akses yang tidak memadai juga dapat memungkinkan konteks prosesor aplikasi yang telah disusupi untuk memanggil layanan di luar cakupan otorisasi yang seharusnya. Karena *firmware* HPSE tidak memiliki standar baku, keamanan *firmware* tersebut sangat bergantung pada tingkat kematangan keamanan dari pemasok *firmware* serta apakah mereka telah merancangnya dengan model kepercayaan (*trust model*) yang tepat.

Komponen Dasar Autosar

Untuk memahami bagian aplikasi perangkat lunak dari ECU berbasis MCU, kita akan meninjau tiga area berbeda dalam arsitektur klasik AUTOSAR sebagaimana ditunjukkan pada Gambar T.5: RTOS, layanan komunikasi, dan AUTOSAR *Runtime Environment* (RTE). Anda disarankan untuk mempelajari spesifikasi AUTOSAR yang relevan dengan bidang minat Anda saat berupaya mengidentifikasi layanan perangkat lunak yang dapat dieksploitasi.



Gambar 7.5 Diagram blok perangkat lunak AUTOSAR Classic 4.x

Sistem Operasi Waktu Nyata (RTOS) dan Penjadwalan Tugas

Setelah eksekusi aplikasi dimulai, proses tersebut biasanya berjalan di bawah kendali sistem operasi waktu nyata (*real-time operating system*/RTOS), seperti AUTOSAR OS. Sistem operasi ini merupakan evolusi dari standar OSEK/VDX yang dirancang untuk menyediakan lingkungan eksekusi yang dikonfigurasi secara statis dan bersifat deterministik bagi aplikasi otomotif yang krusial bagi keselamatan (*safety-critical*). Berbeda dengan sistem operasi tujuan umum (*general-purpose operating system*), AUTOSAR OS menetapkan sebagian besar

keputusan arsitektural pada tahap kompilasi (*build time*), bukan pada saat waktu berjalan (*runtime*).

Pada intinya, AUTOSAR OS menerapkan pemisahan melalui konsep "*OS-Application*". Konsep ini merepresentasikan partisi logis perangkat lunak dengan tingkat hak akses, wilayah memori, dan hak akses yang berbeda-beda. AUTOSAR OS menggunakan istilah "kepercayaan" (*trust*) sebagai properti yang menyiratkan ketiadaan gangguan antar-aplikasi dengan tingkat kekritisan yang beragam, jika ditinjau dari perspektif keselamatan (*safety*). Hal ini bisa membingungkan, karena insinyur keamanan mungkin berasumsi bahwa "*Trusted Application*" (Aplikasi Tepercaya) kebal terhadap serangan tertentu; padahal, istilah tersebut hanya berarti bahwa aplikasi tersebut memiliki sumber daya memori dan perangkat keras yang terisolasi demi pertimbangan keselamatan. Secara teoretis, serangan terhadap ECU berbasis AUTOSAR bisa saja berasal dari sebuah "*Trusted Application*" jika aplikasi tersebut terpapar pada permukaan serangan (*attack surface*) ECU dan mengandung kelemahan keamanan yang dapat dieksploitasi.

Penjadwalan Tugas dan Preemsi

Sistem operasi (OS) AUTOSAR menggunakan model penjadwalan berbasis prioritas tetap. Prioritas tugas ditetapkan secara statis selama konfigurasi sistem dan tidak dapat diubah saat waktu berjalan (*runtime*). Hal ini membuat urutan eksekusi dapat diprediksi; di satu sisi, ini bermanfaat untuk menjamin karakteristik *real-time*, namun di sisi lain, informasi ini sangat berguna bagi penyerang yang sedang melakukan pengintaian (*reconnaissance*). Tugas-tugas dibagi menjadi dua kategori. *Basic Task* (tugas dasar) tidak dapat mengalami pemblokiran (*blocking*) dan hanya berpindah di antara status *Suspended* (ditangguhkan), *Ready* (siap), dan *Running* (berjalan). *Extended Task* (tugas diperluas) dapat menunggu peristiwa (*event*) OS, sehingga memiliki siklus hidup eksekusi yang lebih kompleks.

Penjadwalan bersifat *fully pre-emptive* (preemsi penuh), yang berarti jika tugas dengan prioritas lebih tinggi menjadi *Ready*, tugas tersebut akan langsung menggantikan tugas dengan prioritas lebih rendah yang sedang berjalan. Dalam sistem *multi-core*, setiap inti (*core*) menjalankan penjadwalnya sendiri secara independen, dengan tugas serta *Interrupt Service Routine* (ISR) yang terikat secara statis pada inti tertentu. Dari sudut pandang serangan, keterikatan yang deterministik ini dapat mempermudah perancangan eksploitasi perangkat lunak yang mengandalkan perilaku waktu (*timing*) yang konsisten.

Penanganan Layanan Interupsi

Interrupt Service Routine (ISR) menempati posisi istimewa dalam model eksekusi OS AUTOSAR. Semua interupsi dianggap memiliki prioritas yang lebih tinggi daripada tugas apa pun, terlepas dari konfigurasi prioritas tugas tersebut. Fakta ini menjadikan kode penanganan interupsi sebagai target yang sangat bernilai. OS AUTOSAR mendefinisikan dua jenis interupsi, sebagaimana diilustrasikan dalam Tabel 7.1.

Tabel 7.1 Perbedaan dalam penanganan rutin layanan interupsi (*Interrupt Service Routine*/ISR) Kategori 1 dan Kategori 2

Category 1 IS™	**Category 2 IS™**
Melewati sistem operasi (OS) AUTOSAR sepenuhnya dan langsung dieksekusi saat interupsi terjadi.	Dikelola dan dibungkus oleh AUTOSAR OS, sehingga eksekusinya melalui penanganan interupsi yang dikendalikan oleh OS.
Dirancang untuk meminimalkan latensi interupsi.	Dirancang untuk menyeimbangkan latensi penanganan interupsi dengan kontrol serta perlindungan dari OS.
Memiliki prioritas yang lebih tinggi daripada tugas (*task*), terlepas dari bagaimana prioritas tugas tersebut dikonfigurasikan.	Memiliki prioritas yang lebih tinggi daripada tugas (*task*), terlepas dari bagaimana prioritas tugas tersebut dikonfigurasikan.
Berjalan di luar cakupan perlindungan memori, waktu, dan layanan yang ditegakkan oleh sistem operasi.	Dapat tunduk pada kebijakan perlindungan memori, waktu, dan layanan yang ditegakkan oleh sistem operasi (OS).

Akibat cara AUTOSAR OS mengelola interupsi, kerentanan apa pun dalam ISR Kategori 1 akan dieksekusi dengan pengawasan OS yang sangat minim, menjadikannya target yang menarik.

Sebaliknya, interupsi Kategori 2 dikelola (dibungkus) oleh OS, yang kemudian menerapkan kebijakan perlindungan layanan dan waktu. Selain itu, banyak MCU menyediakan tabel vektor interupsi yang dapat dipindahkan (*relocatable*), sehingga *flash bootloader* dan aplikasi dapat mengelola rutin layanan interupsi secara terpisah, masing-masing dalam domain eksekusinya sendiri. Jika penyerang mampu mengubah *offset* tabel vektor interupsi, maka interupsi dapat dialihkan ke kode yang dikendalikan oleh penyerang. Oleh karena itu, pengetahuan mengenai tata letak tabel vektor dan penempatan memori yang dapat diperoleh melalui lembar data (*datasheet*) atau analisis *firmware* menjadi langkah krusial dalam mengeksploitasi rutin layanan interupsi.

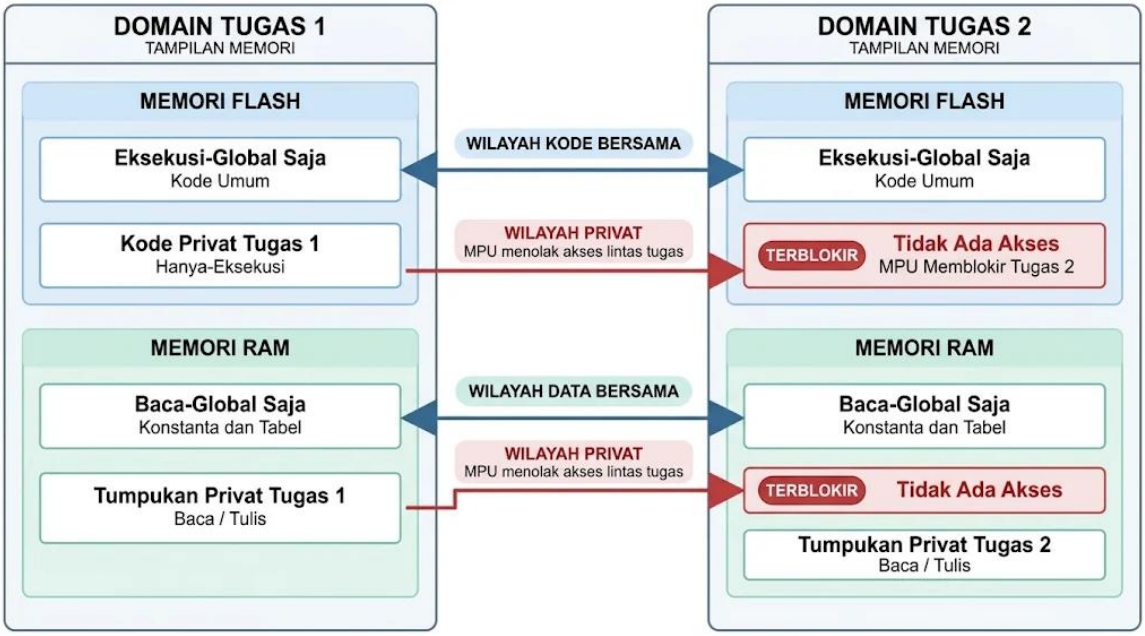
Eksekusi Dalam Mode Istimewa (*Privileged Mode*) dan Mode Pengguna (*User Mode*)

Model keamanan AUTOSAR OS sangat bergantung pada mode prosesor yang ditegakkan oleh perangkat keras. Aplikasi OS yang "tepercaya" (*trusted*) dieksekusi dalam mode istimewa dan memiliki akses khusus ke memori, periferal, serta seluruh API OS. Karena MCU tidak memiliki pemisahan tingkat hak akses yang kompleks seperti pada SoC, sering kali sistem operasi dan aplikasi OS yang "tepercaya" berjalan dalam konteks tepercaya yang sama, karena keduanya berbagi tingkat hak akses yang sama!

Aplikasi OS yang "tidak tepercaya" (*non-trusted*) biasanya dieksekusi dalam mode pengguna. Akses aplikasi tersebut terhadap wilayah memori dan register kendali perangkat keras seperti konfigurasi MPU atau pewaktu sistem (*system timer*) dapat dibatasi, biasanya untuk menjamin kebebasan dari gangguan (*freedom from interference*). Pemisahan ini merupakan mekanisme utama yang digunakan untuk mengisolasi kesalahan (*fault*) dan membatasi dampak dari kode aplikasi yang berperilaku tidak semestinya.

Perlindungan Memori dan Isolasi Aplikasi

Isolasi memori dalam implementasi AUTOSAR OS biasanya ditegakkan menggunakan *Memory Protection Unit* (MPU). MPU membagi ruang alamat menjadi beberapa wilayah dengan hak akses berbeda yang terikat pada aplikasi OS yang sedang dieksekusi. Gambar 7.6 menunjukkan contoh pembagian antara dua tugas (*task*) yang berasal dari aplikasi OS yang berbeda. Baik wilayah memori *flash* maupun RAM dapat dibatasi aksesnya, bergantung pada tugas mana yang sedang memegang konteks eksekusi.



Gambar 7.6 Model peta memori sistem berbasis MCU

Sistem operasi (OS) mencegah suatu aplikasi OS menulis ke bagian data milik aplikasi OS lainnya. Wilayah *stack* juga diisolasi jika terdapat dukungan perangkat keras. Pada sistem yang tidak memiliki perlindungan perangkat keras yang memadai, OS dapat menyediakan pemantauan *stack* sebagai mekanisme upaya terbaik (*best-effort*), yaitu dengan memeriksa terjadinya *overflow* (peluapan) saat peralihan konteks (*context switch*), alih-alih mencegahnya secara langsung. Selain perlindungan memori dasar, AUTOSAR menerapkan kepemilikan atas objek-objek OS seperti *Alarm*, *Event*, dan *Schedule Table*. Setiap objek dimiliki oleh aplikasi OS tertentu; upaya untuk mengakses objek tanpa hak kepemilikan yang sah akan mengakibatkan pelanggaran perlindungan layanan (*service protection violation*).

Isolasi Temporal dan Perlindungan Waktu

Di samping isolasi spasial, OS AUTOSAR menerapkan isolasi temporal untuk mencegah perambatan kesalahan (*fault propagation*) antar-tugas (*task*). Hal ini diimplementasikan melalui mekanisme perlindungan waktu yang menetapkan anggaran eksekusi (*execution budget*) secara eksplisit untuk setiap tugas. Setiap tugas atau interupsi Kategori 2 dapat diberi anggaran waktu eksekusi. Selain itu, terdapat anggaran penguncian (*lock budget*) yang membatasi durasi suatu tugas atau interupsi dalam menguasai sumber daya bersama atau

menahan status interupsi agar tetap nonaktif (*disabled*). Terakhir, kerangka waktu (*time frame*) menetapkan batas minimum jeda antar-kedatangan (*inter-arrival time*) untuk melindungi sistem dari lonjakan interupsi (*interrupt storm*) atau pemicu yang tidak berfungsi sebagaimana mestinya. Mekanisme ini sangat berguna jika antarmuka yang dikendalikan dari luar diizinkan untuk memicu interupsi.

Perlu dicatat bahwa pelanggaran terhadap batasan-batasan ini ditangani melalui *Protection Hook*. Bergantung pada konfigurasi perangkat lunaknya, OS dapat menghentikan tugas yang melanggar, menghentikan seluruh aplikasi OS, atau memicu pemadaman sistem secara menyeluruh. Dari sudut pandang keamanan, mekanisme respons ini mencakup strategi pembatasan (*containment*) sekaligus potensi vektor serangan penolakan layanan (*denial-of-service*). Penyerang yang mampu memicu kebijakan *Protection Hook* secara sengaja dapat menyebabkan sistem berulang kali diatur ulang (*reset*) atau mengalami penurunan fungsionalitas.

Akses Periferal dan Layanan Komunikasi

Pada tingkat terendah dari tumpukan (*stack*) AUTOSAR Classic terdapat *Microcontroller Abstraction Layer* (MCAL). Driver MCAL menyediakan akses langsung ke periferal di dalam mikrokontroler (*on-chip*), seperti antarmuka SPI, pengontrol CAN, antarmuka LIN, Ethernet MAC, dan UART. Driver ini terikat erat dengan mikrokontroler tertentu dan mengekspos parameter konfigurasi tingkat rendah, termasuk filter penerimaan CAN, perilaku interupsi, penggunaan DMA, serta penentuan ukuran *buffer*.

Dari perspektif serangan, MCAL membentuk permukaan serangan (*attack surface*) perangkat lunak pertama yang terpapar terhadap input perangkat keras atau lalu lintas jaringan yang tidak sesuai format maupun tidak terduga. Penyaringan yang lemah atau terlalu longgar pada lapisan ini dapat menyebabkan pesan yang seharusnya tidak mencapai lapisan perangkat lunak yang lebih tinggi justru diteruskan ke atas, sehingga memperluas permukaan serangan sebelum validasi pada tingkat protokol atau aplikasi sempat dilakukan.

Di atas MCAL, tumpukan komunikasi AUTOSAR memperkenalkan tingkat abstraksi yang semakin tinggi. Modul khusus jaringan seperti *CAN Interface*, *LIN Interface*, dan *Ethernet Interface* menormalisasi akses ke berbagai bus dan memisahkan lapisan atas dari detail perangkat keras. Modul-modul ini menyalurkan data ke layanan yang beroperasi pada PDU (*Protocol Data Units*), sinyal, atau sesi, alih-alih pada *frame* data mentah. Meskipun abstraksi ini menyederhanakan pengembangan dan portabilitas aplikasi, hal ini juga mengalihkan banyak keputusan terkait keamanan ke dalam artefak konfigurasi statis yang dibuat selama tahap integrasi. Setelah diterapkan, konfigurasi ini biasanya dianggap sudah benar dan tidak akan berubah sepanjang masa pakai ECU (*Electronic Control Unit*).

Pada lapisan atas tumpukan tersebut, tepat di bawah lapisan aplikasi, AUTOSAR mendefinisikan beberapa layanan komunikasi yang sangat relevan dari sudut pandang keamanan ofensif:

- ***Communication* (COM)**: Lapisan COM memungkinkan pertukaran data berbasis sinyal antar komponen perangkat lunak. Lapisan ini menentukan sinyal mana yang dikirim atau diterima, bagaimana sinyal tersebut dikemas ke dalam *Protocol Data Units* (PDU),

serta kapan transmisi dilakukan. Parameter konfigurasi AUTOSAR mengontrol pengidentifikasi CAN, periodisitas transmisi, semantik pembaruan sinyal, pemantauan *timeout*, dan aktivasi grup. Konfigurasi yang terlalu longgar dapat menyebabkan ECU memproses pesan CAN dari pengirim yang tidak dimaksudkan, atau mengirimkan pesan yang tidak ditujukan bagi konsumen eksternal. Bagi penyerang yang memiliki akses ke bus, transmisi periodik yang dapat diprediksi dikombinasikan dengan penanganan *timeout* yang lemah dapat dimanfaatkan untuk melakukan serangan *replay*, *spoofing*, atau *denial-of-service* pada tingkat tugas (*task-level*).

- **Manajemen Jaringan (*Network Management*/Nm)**: Lapisan ini mengatur bagaimana sebuah ECU berpartisipasi dalam koordinasi status tidur (*sleep*) dan bangun (*wake-up*) jaringan. Modul Nm menentukan kapan saluran komunikasi aktif, berapa lama saluran tersebut tetap aktif, serta pesan mana yang menandakan adanya aktivitas jaringan. Konfigurasi Nm yang keliru dapat memungkinkan penyerang untuk menjaga ECU tetap aktif tanpa batas waktu, menguras baterai kendaraan, atau memicu transisi berulang antar status operasional. Pada jaringan yang saling terhubung erat, dampak ini dapat merambat melampaui satu ECU saja dan memengaruhi berbagai domain jaringan.
- **Komunikasi Onboard yang Aman (*Secure Onboard Communication*/SecOC)**: Untuk sistem yang menerapkan autentikasi pesan, SecOC menambahkan perlindungan kriptografis pada PDU (*Protocol Data Unit*) tertentu. Protokol ini memperkenalkan nilai kesegaran (*freshness values*) dan tag autentikasi untuk menangkal serangan *replay* (pengulangan pesan) serta pemalsuan pesan (*spoofing*). Meskipun hal ini secara signifikan meningkatkan ketahanan jaringan, kompleksitas implementasinya dapat menciptakan celah untuk melampaui perlindungan tersebut. Manajemen kunci yang lemah, kemungkinan pengaturan ulang (*rollback*) pada penghitung kesegaran (*freshness counters*), atau penerapan parsial di mana hanya sebagian pesan yang dilindungi dapat menyebabkan asumsi kepercayaan yang tidak konsisten. Dari perspektif serangan, mencari kelemahan dalam SecOC pada dasarnya hanya mengalihkan fokus: dari upaya menembus protokol CAN yang tidak aman, menjadi upaya menembus solusi perangkat lunak SecOC yang diimplementasikan dengan buruk.
- **Manajer Komunikasi Diagnostik (*Diagnostic Communication Manager*/Dcm)**: Protokol ini menyediakan fungsionalitas diagnostik bagi klien diagnostik, baik yang berada di dalam maupun di luar kendaraan. Dcm berperan sebagai perantara antara permintaan diagnostik yang masuk dengan *callback* aplikasi internal yang menjalankan layanan UDS (*Unified Diagnostic Services*). Setiap layanan diagnostik dipetakan secara statis ke fungsi penangan (*handler functions*) dan jika ada dilindungi oleh pemeriksaan sesi serta keamanan. Sebagai contoh, suatu layanan diagnostik mungkin diizinkan dalam sesi pemrograman, namun dilarang dalam sesi standar (*default session*). Demikian pula, layanan diagnostik tertentu mungkin diizinkan hanya jika tingkat keamanan tertentu telah diaktifkan, dan akan ditolak jika tidak. Kesalahan konfigurasi di area ini merupakan sumber risiko yang signifikan. Rutinitas diagnostik yang tidak

disertai autentikasi yang memadai, atau prasyarat yang tidak tepat untuk kontrol sesi, dapat memungkinkan penyerang untuk menjalankan operasi sensitif seperti akses memori, pembaruan kalibrasi, pengaturan ulang (*reset*) ECU, atau masuk ke mode pemrograman *firmware* dalam kondisi yang tidak aman. Karena Dcm dirancang secara khusus untuk menerima input eksternal, komponen ini sering kali menjadi salah satu *attack surface* (permukaan serangan) yang paling kaya dan paling andal dalam ECU berbasis AUTOSAR.

- **Ethernet**: Seiring dengan meningkatnya penggunaan Ethernet, kini lazim ditemukan ECU berbasis MCU yang mendukung tumpukan jaringan (*networking stack*) lengkap, termasuk driver Ethernet, ARP, ICMP, IPv4, IPv6, TCP, UDP, DHCP, dan TLS. Keberadaan komponen-komponen perangkat lunak ini harus menjadi sinyal bagi seorang insinyur keamanan ofensif untuk memeriksa adanya kerentanan yang sudah diketahui atau kesalahan konfigurasi yang dapat memperluas permukaan serangan (*attack surface*) terhadap ECU target.

Tablel 7.2 menyediakan daftar modul klasik AUTOSAR umum yang mendukung kemampuan jaringan sehingga menjadikannya target yang berguna untuk analisis keamanan.

Tabel 7.2 Daftar modul AUTOSAR yang harus dianalisis saat mencari kelemahan yang berkaitan dengan jaringan dan status

Modul	**Deskripsi**
Com (COM)	Melakukan pengemasan/pembongkaran sinyal aplikasi ke dalam/dari I-PDU serta menyediakan API berbasis sinyal untuk aplikasi.
PduR (*PDU Router*)	Mengarahkan PDU di antara modul lapisan atas (misalnya, COM, DCM) dan modul khusus bus/media transmisi, sehingga memungkinkan fungsi *gateway*.
Dcm (*Diagnostic Communication Manager*)	Menangani diagnostik yang tidak bergantung pada jaringan (UDS), mendistribusikan layanan, dan berinteraksi dengan lapisan transpor/media transmisi.
ComM (*Communication Manager*)	Menentukan mode komunikasi ECU (penuh/tidak ada/senyap) dan mengoordinasikan ketersediaan komunikasi bagi pengguna.
BswM (*BSW Mode Manager*)	Koordinator berbasis aturan yang mengelola mode BSW (*Basic Software*) dan aplikasi, serta memicu tindakan yang telah dikonfigurasi.
Nm (*Network Management*)	Mengoordinasikan transisi antara operasi normal dan kondisi *bus-sleep* di seluruh jaringan, terlepas dari jenis bus yang digunakan.
UdpNm	Manajemen jaringan melalui UDP untuk klaster IP/Ethernet guna mengoordinasikan status komunikasi dan status sleep.
SoAd (*Socket Adapter*)	Menghubungkan ekosistem PDU AUTOSAR dengan *stack* TCP/IP berbasis soket, dengan memetakan I-PDU ke soket.
DoIP (*Diagnostics over IP*)	Mentransmisikan diagnostik UDS melalui IP (biasanya TCP) dan berintegrasi dengan SoAd/TcpIp.

TcpIp (*TCP/IP stack*)	*Stack* TCP/IP AUTOSAR yang mengimplementasikan IPv4/IPv6, UDP, TCP, serta protokol terkait (ARP, ICMP, DHCP, NDP, TLS).
EthSM (*Ethernet State Manager*)	Mengontrol status operasional pengontrol/transceiver Ethernet dan melaporkan status tautan (*link state*) melalui EthIf.
EthIf (*Ethernet Interface*)	Antarmuka yang tidak bergantung pada perangkat keras, menghubungkan lapisan atas dengan *driver* MAC/PHY Ethernet.
Eth (*Ethernet driver*)	*Driver* MCAL yang menyediakan akses seragam ke perangkat keras MAC Ethernet untuk EthIf.
EthTrcv (*Ethernet Transceiver Driver*)	*Driver* MCAL yang mengabstraksi *transceiver* PHY dan menyediakan antarmuka standar ke lapisan atas (EthIf).

AUTOSAR Runtime Environment (RTE)

AUTOSAR *Runtime Environment* (RTE) adalah lapisan middleware terstandarisasi yang ditetapkan oleh AUTOSAR, yang memisahkan komponen perangkat lunak aplikasi dari perangkat keras ECU serta perangkat lunak dasar (*Basic Software*/BSW) yang mendasarinya. Tujuan utamanya adalah untuk memungkinkan portabilitas, penggunaan kembali (*reusability*), dan integrasi perangkat lunak aplikasi di berbagai ECU dan platform perangkat keras, tanpa perlu mengubah logika aplikasi.

Model arsitektur AUTOSAR mengandalkan RTE untuk menyediakan abstraksi komunikasi yang seragam antara Komponen Perangkat Lunak Aplikasi (*Application Software Components*/SW-C) dan BSW AUTOSAR. Abstraksi ini dirancang agar OEM dan pemasok Tier-1 dapat mengembangkan komponen perangkat lunak secara independen, dengan menyerahkan penentuan ikatan komunikasi (*communication bindings*) kepada RTE pada saat integrasi sistem.

Perlu dicatat bahwa RTE bukanlah middleware yang bersifat dinamis. Sebaliknya, RTE dihasilkan secara statis dari deskripsi sistem (ARXML) selama proses pembangunan (*build process*) untuk menangani tanggung jawab arsitektural inti, seperti:

- Komunikasi intra-ECU: Pemanggilan fungsi atau pertukaran data berbasis memori di antara SW-C yang dipetakan ke ECU yang sama.
- Abstraksi komunikasi antar-ECU: Akses transparan terhadap sinyal dan layanan yang pada akhirnya disalurkan melalui COM, PDU Router, dan tumpukan jaringan (*network stacks*) seperti CAN, LIN, FlexRay, atau Ethernet.
- Pemanggilan *runnable*: Pemetaan *runnable* aplikasi ke tugas (*task*) sistem operasi, peristiwa (*event*), dan batasan waktu (*timing constraint*)
- Akses mode dan layanan: Akses terkontrol ke mode ECU, diagnostik, layanan NVM, serta fungsionalitas BSW lainnya

Dari perspektif keselamatan fungsional (*functional safety*), RTE secara langsung mendukung tujuan ISO 26262 dengan menerapkan kontrak antarmuka yang ketat di antara komponen-komponen perangkat lunak. Komunikasi antar komponen perangkat lunak (SW-C) hanya terjadi melalui port dan antarmuka yang didefinisikan secara statis; hal ini mencegah akses data yang tidak diinginkan dan mengurangi risiko keterikatan (*coupling*) yang tidak disengaja. Jika dikombinasikan dengan perlindungan memori di tingkat sistem operasi (OS) dan isolasi

tugas (*task isolation*), RTE berkontribusi dalam mewujudkan "kebebasan dari interferensi" (*freedom from interference*) dengan memastikan bahwa komponen-komponen hanya berinteraksi melalui jalur yang telah disetujui. Penghapusan pengikatan dinamis (*dynamic binding*) dan alokasi memori saat waktu eksekusi (*runtime*) memungkinkan eksekusi yang deterministik sebuah prasyarat untuk membangun sistem yang aman. Selain itu, RTE menyediakan batasan komunikasi yang konsisten, sehingga mekanisme keselamatan seperti perlindungan ujung-ke-ujung (*end-to-end protection*) dapat diterapkan secara seragam.

Dari perspektif keamanan (*security*), RTE mencegah terbentuknya saluran komunikasi implisit dengan pihak-pihak yang tidak dituju. Komponen perangkat lunak tidak dapat saling bertukar data kecuali jika hubungan pengirim-penerima (*sender-receiver*) atau klien-server telah dikonfigurasi secara eksplisit. Batasan ini mengurangi komunikasi yang tidak diinginkan dan mempermudah analisis terhadap aliran data. RTE juga menghindari fitur *middleware* dinamis yang sering ditemukan dalam lingkungan eksekusi yang lebih kompleks, seperti penemuan layanan (*service discovery*) atau perutean dinamis (*dynamic routing*) saat waktu eksekusi. Dengan demikian, kompleksitas kode dapat dikurangi dan *attack surface* (permukaan serangan) yang terkait dengan perilaku dinamis pun dapat dipersempit.

Tentu saja, dapat dikatakan bahwa tanpa isolasi memori yang memadai antara komponen perangkat lunak aplikasi (*Application SW-C*) dan objek data terkait, sebuah komponen dapat mengakses port data milik komponen lain dengan cara menemukan alamat memori dari objek data tersebut. Dalam praktiknya, AUTOSAR RTE lebih tepat dipandang sebagai sarana pendukung (*enabler*) daripada penegak keamanan (*enforcer*). RTE membatasi cara komponen perangkat lunak berinteraksi, namun tidak secara langsung melindungi interaksi tersebut. Meskipun demikian, memahami cara kerja internal RTE sangatlah penting saat melakukan rekayasa balik (*reverse engineering*) terhadap suatu sistem, terutama untuk mengidentifikasi objek data kritis dan antarmuka yang berpotensi menjadi target serangan.

Kode Kalibrasi dan Penelusuran Kesalahan (Debugging)

Selain logika kendali utama yang mengatur perilaku kendaraan, banyak unit kendali elektronik (ECU) berbasis mikrokontroler (MCU) memuat perangkat lunak tambahan yang ditujukan untuk kalibrasi, validasi, dan penelusuran kesalahan selama tahap pengembangan serta pengujian. Meskipun kemampuan ini sangat penting dalam alur kerja rekayasa, sering kali fitur-fitur tersebut tetap ada baik sebagian maupun seluruhnya hingga ke tahap produksi. Jika tidak dikendalikan dengan ketat, hal ini dapat memperluas celah serangan secara signifikan.

Protokol Kalibrasi dan Akses Memori

Universal Measurement and Calibration Protocol (XCP) merupakan standar yang memungkinkan perangkat kalibrasi untuk membaca dan menulis data secara langsung ke dalam memori ECU melalui bus komunikasi kendaraan. Berbeda dengan UDS, yang utamanya ditujukan untuk diagnostik dan pencatatan kesalahan di lapangan, XCP dirancang untuk lingkungan pengembangan maupun produksi. Protokol ini mendukung pengukuran dengan laju tinggi, kalibrasi secara daring (*online*), dan dalam banyak implementasi pemrograman ulang memori *flash* pada perangkat lunak ECU.

Salah satu contoh penggunaan yang umum adalah penyetelan (*tuning*) algoritma kendali, seperti sistem pengereman, sistem penggerak (*powertrain*), atau kendali sasis untuk konfigurasi kendaraan tertentu. Parameter kalibrasi termasuk massa kendaraan, jarak sumbu roda, karakteristik ban, dan penguatan (*gain*) pengontrol dapat diakses melalui berkas deskripsi A2L. Insinyur kalibrasi menggunakan XCP untuk mengubah parameter-parameter tersebut secara waktu nyata (*real-time*) saat ECU sedang beroperasi. Setelah perilaku yang diinginkan tercapai, nilai-nilai akhir tersebut biasanya disimpan ke dalam memori *flash* non-volatil. Banyak implementasi XCP juga mendukung pemrograman ulang ECU, yang memungkinkan pemuatan citra *firmware* (*firmware image*) melalui antarmuka yang sama bukan melalui protokol *bootloader* khusus demi kemudahan selama proses pengembangan.

Dari perspektif keamanan ofensif, risikonya cukup jelas. Jika layanan kalibrasi tetap aktif pada unit produksi tanpa autentikasi dan kontrol akses yang kuat, penyerang yang memiliki akses ke bus komunikasi dapat membaca area memori sensitif atau mengubah variabel kendali saat sistem sedang berjalan. Bahkan jika akses tulis dibatasi, akses baca saja sudah cukup untuk membocorkan data rahasia yang dapat digunakan untuk melancarkan serangan lebih lanjut. Demikian pula, akses untuk pemrograman memori *flash* dapat disalahgunakan untuk memasang perangkat lunak berbahaya.

Mode Penelusuran Kesalahan dan Titik Akses Tersembunyi

Selain protokol kalibrasi formal, beberapa ECU menerapkan mode khusus untuk penelusuran kesalahan (*debugging*) atau pengembangan yang dapat mengubah perilaku eksekusi normal. Mode-mode ini mungkin melonggarkan kontrol akses, mengaktifkan penangan perintah (*command handler*) tambahan, atau mengekspos status internal sistem untuk tujuan diagnostik. Dalam beberapa kasus historis, ECU terbukti dapat memasuki status pintu belakang (*backdoor*) yang tidak terdokumentasi saat menerima nilai tertentu yang telah diprogram secara permanen (*hard-coded*) dalam permintaan diagnostik, yang pada praktiknya membuka akses ke fungsionalitas dengan hak istimewa.

Mekanisme semacam itu jarang diiklankan dan sering kali merupakan sisa-sisa dari alur kerja pengembangan. Bagi penyerang, mekanisme ini menjadi target bernilai tinggi. Dengan melakukan rekayasa balik (*reverse engineering*) terhadap citra *firmware*, seorang analis dapat mengidentifikasi pengidentifikasi pesan yang mencurigakan, tabel perintah yang tidak aktif, atau percabangan kondisional yang tampak membatasi perilaku dengan hak akses khusus. Penangan perintah (*command handler*) yang tidak terdokumentasi dapat mengindikasikan keberadaan jalur debug tersembunyi yang tidak pernah dihapus sepenuhnya.

AUTOSAR DLT

Dalam sistem AUTOSAR Classic, proses debugging dan observabilitas sering kali diimplementasikan menggunakan modul *Diagnostic Log and Trace* (DLT). DLT dirancang untuk menyediakan pencatatan (*logging*), pelacakan (*tracing*), dan diagnostik waktu nyata (*runtime*) yang terstruktur dari ECU ke perangkat eksternal melalui saluran komunikasi seperti CAN atau Ethernet.

Meskipun DLT sangat berguna selama tahap pengembangan, modul ini menciptakan permukaan serangan (*attack surface*) tersendiri jika tetap diaktifkan pada tahap produksi.

Pesan-pesan DLT dapat mengekspos status internal perangkat lunak, jalur eksekusi, informasi waktu (*timing*), serta kondisi kesalahan. Tingkat pencatatan yang sangat mendetail (*verbose*) dan konteks aplikasi yang tidak difilter dapat secara tidak sengaja membocorkan detail implementasi sensitif, yang pada akhirnya mempermudah proses rekayasa balik. Setelah memahami arsitektur perangkat keras dan perangkat lunak dari ECU berbasis MCU, kini saatnya untuk menelaah kelemahan umum yang memungkinkan eksploitasi terhadap sistem tersebut.

7.3 KELEMAHAN PERANGKAT KERAS DAN PERANGKAT LUNAK PADA ECU BERBASIS MCU

ECU berbasis MCU memiliki berbagai kelemahan keamanan yang secara garis besar terbagi menjadi dua kategori: kelemahan di tingkat perangkat keras yang berkaitan dengan paparan fisik dan elektrik, serta kelemahan di tingkat perangkat lunak yang muncul akibat cacat implementasi atau asumsi desain yang tidak tepat. Dalam praktiknya, serangan yang berhasil sering kali menggabungkan keduanya. Teknik perangkat keras dapat digunakan untuk mengekstrak *firmware*, yang kemudian memungkinkan analisis mendalam serta eksploitasi terhadap kerentanan perangkat lunak. Mengingat hampir tidak ada batasan dalam cara perancangan suatu sistem, mustahil untuk merinci setiap kemungkinan cacat yang dapat dieksploitasi. Oleh karena itu, bagian ini akan berfokus pada kelemahan keamanan yang paling umum dan sering muncul dalam rangkaian serangan nyata terhadap ECU berbasis MCU.

Kelemahan Perangkat Keras

Sebagaimana ditunjukkan dalam Gambar 7.7, kelemahan perangkat keras mencakup berbagai aspek, mulai dari antarmuka debug perangkat keras yang tidak aman hingga ketiadaan enkripsi pada memori. Pada bagian ini, kita akan membahas setiap kategori kelemahan keamanan tersebut dan mempelajari bagaimana kelemahan-kelemahan itu memungkinkan terjadinya eksploitasi keamanan.



Gambar 7.7 Gambaran singkat mengenai kelemahan perangkat keras yang umum

Antarmuka Debug/Pemrograman yang Tidak Diamankan

Salah satu titik masuk paling andal dalam rantai serangan adalah antarmuka debug atau pemrograman perangkat keras yang tidak diamankan. Antarmuka seperti JTAG dan SWD lazim disertakan dalam desain ECU untuk mendukung pengembangan, manufaktur, pemecahan masalah, atau analisis kegagalan. Pada perangkat keras produksi, antarmuka ini mungkin tetap dapat diakses melalui konektor yang terpasang, titik uji (*test pad*) yang terbuka, footprint komponen yang tidak terpasang, atau jalur (*trace*) yang mengarah ke pin perangkat terkait. Bahkan jika titik uji telah dihapus, penyerang yang memiliki akses fisik mungkin dapat melacak sinyal tersebut kembali ke via, komponen, atau pin perangkat yang dapat diakses.

Setelah sinyal yang relevan diidentifikasi, akses tidak selalu mengharuskan pembuatan perangkat uji khusus. Penguji dapat terhubung langsung dengan menyolder kabel ke titik-titik yang dapat diakses, atau menggunakan perangkat probe (*probing fixture*) jika akses berulang diperlukan. Akses debug atau pemrograman juga bisa tetap terbuka melalui jalur alternatif, seperti mode *bootloader* berbasis UART atau antarmuka pemulihan serial lainnya. Jika antarmuka ini tidak dinonaktifkan, tidak memerlukan autentikasi, atau tidak dilindungi dengan cara lain dalam tahap produksi, maka antarmuka tersebut dapat memungkinkan ekstraksi *firmware*, pemeriksaan memori, pemrograman ulang, atau pengelabuan kontrol keamanan berbasis perangkat lunak.

Dengan akses ke JTAG atau SWD yang tidak terlindungi, penyerang dapat menghentikan CPU, membaca isi memori, dan mengekstrak *firmware* untuk dianalisis secara offline. Setelah mendapatkan *firmware* tersebut, penyerang dapat menganalisisnya untuk mencari kerentanan, membuat patch berbahaya, lalu memuatnya kembali (*reflash*) ke perangkat target untuk eksplorasi dan eksploitasi lebih lanjut. Proses ini sering kali berujung pada ditemukannya data rahasia yang tertanam (*hard-coded secrets*), yang kemudian dapat digunakan untuk menyusun rantai serangan yang lebih canggih. Itulah sebabnya, dari perspektif keamanan ofensif, penilaian terhadap antarmuka debug sering kali menjadi langkah awal dalam fase penemuan (*discovery phase*).

Jalur debug berbasis UART juga sama berbahayanya. Jalur ini sering kali mengandalkan urutan perintah khusus untuk masuk ke mode pemrograman atau debug, dan mungkin dilindungi oleh kata sandi atau mekanisme tantangan-respons (*challenge-response*). Dalam beberapa kasus yang terdokumentasi, ECU produksi dikirimkan dengan antarmuka ini tanpa perlindungan yang memadai. Kelemahan yang umum ditemukan meliputi penggunaan kata sandi bawaan (*default*) atau kata sandi global, ketergantungan pada kerahasiaan mode debug dan uji yang tidak didokumentasikan, atau kelemahan dalam implementasi kriptografis pada mekanisme tantangan-respons tersebut. Bahkan ketika bit pengunci perangkat keras atau flag penonaktifan debug diatur, penyerang yang gigih telah menunjukkan keberhasilan bypass melalui injeksi kesalahan, manipulasi tegangan atau jam, atau penemuan pintu belakang pabrikan yang tidak terdokumentasi.

Pembacaan Memori dan Pintasan Perlindungan

Saat port debug dikunci dengan benar, penyerang sering kali mencoba mengekstrak *firmware* dari memori *non-volatile* pada ECU. Banyak MCU memiliki bit OTP (*One-Time Programmable*) untuk "perlindungan kode" yang mencegah pembacaan langsung terhadap isi *flash* internal setelah MCU memasuki tahap produksi. Akan tetapi, para peretas perangkat keras telah mengembangkan teknik untuk memintas perlindungan tersebut. Salah satu pendekatannya adalah dengan melakukan *voltage glitching* (gangguan tegangan) atau *clock glitching* (gangguan sinyal detak) tepat pada saat MCU memeriksa bit perlindungan; hal ini memicu kesalahan pembacaan yang kemudian membuka akses (penjelasan lebih lanjut mengenai *glitching* akan dibahas kemudian).

Pendekatan lain adalah dengan memanfaatkan mode pengujian dari produsen. Sebagai contoh, beberapa mikrokontroler memiliki pin atau mode pengujian pabrik yang, jika

diaktifkan (biasanya memerlukan pembukaan lapisan pelindung cip dan penyambungan ke titik-titik kontak yang sangat kecil), akan mengeluarkan seluruh isi memori *flash* internal. Selain itu, jika mikrokontroler menggunakan cip memori eksternal untuk menyimpan kode atau data, maka komponen tersebut menjadi sasaran yang mudah baik dengan melepas cip melalui proses penyolderan ulang maupun dengan memantau lalu lintas data pada bus alamat dan bus data. Untuk mendapatkan kode dan data kalibrasi ECU, seseorang tidak selalu harus memiliki keahlian khusus dalam rekayasa balik (*reverse engineering*). Faktanya, beberapa produsen kendaraan (OEM) bahkan menyediakan citra perangkat lunak melalui portal dealer mereka; satu-satunya syarat yang diperlukan hanyalah mengetahui nomor VIN kendaraan tersebut!

Kerentanan Terhadap Penyuntikan Kesalahan (Fault Injection atau Glitching)

Penyuntikan kesalahan melibatkan upaya sengaja untuk memunculkan kesalahan dalam operasi MCU agar perangkat tersebut melakukan sesuatu yang tidak diinginkan. Metode yang umum digunakan antara lain *voltage glitching* (menurunkan tegangan suplai secara singkat) atau *clock glitching* (menyisipkan pulsa detak/sinyal *clock* tambahan) pada momen yang tepat. Metode lain mencakup penggunaan pulsa elektromagnetik (*electromagnetic fault injection* atau EMFI) atau bahkan tembakan laser untuk mengganggu kinerja cip. Tujuannya sering kali adalah untuk melewatkan suatu instruksi atau menyebabkan CPU menjalankan perintah secara keliru. Sebagai contoh, perhatikan kode yang memeriksa kata sandi sebelum mengizinkan eksekusi program untuk berlanjut:

```
"if(password_correct)
proceed;
else reset;"
```

Gangguan (*glitch*) yang diatur waktunya secara cermat dapat membalikkan kondisi atau melewati instruksi tertentu, sehingga memungkinkan penyerang menjalankan alur kode yang mereka kehendaki tanpa perlu memberikan kata sandi yang benar. Para peneliti telah berulang kali menunjukkan kemampuan mereka untuk menyisipkan gangguan selama proses *secure boot* (booting aman) pada instruksi krusial sebagai contoh, dengan merusak pemeriksaan verifikasi tanda tangan guna memperoleh hak eksekusi kode melalui kode yang telah mereka modifikasi. Sebagaimana telah kita lihat pada bagian 3, perangkat modern untuk melakukan *glitching* kini sudah cukup mudah diakses, sehingga teknik ini tidak lagi terbatas pada laboratorium milik negara saja. Faktanya, para penghobi maupun peneliti telah berhasil melakukan serangan *glitch* pada mikrokontroler (MCU) menggunakan perangkat keras sumber terbuka (*open-source*), seperti ChipWhisperer untuk analisis *side-channel* dan penyisipan kesalahan (*fault injection*), serta ChipSHOUTER untuk penyisipan kesalahan melalui gelombang elektromagnetik.

Intinya, penyisipan kesalahan meruntuhkan asumsi kepercayaan terhadap perilaku perangkat keras: dengan gangguan yang tepat, langkah-langkah keamanan dapat dilewati. Meskipun penyisipan kesalahan itu sendiri bukanlah sebuah kelemahan, ketiadaan langkah pertahanan untuk menangkalnya dianggap sebagai suatu kelemahan. Sebagai contoh, tidak

adanya dukungan untuk sirkuit pemantau detak (*clock*) dan tegangan yang dirancang khusus untuk mendeteksi gangguan berbahaya merupakan sebuah kelemahan. Untuk register yang krusial bagi keamanan dan rentan terhadap ancaman injeksi kesalahan (*fault-injection*), ketiadaan redundansi perangkat keras seperti *Triple Modular Redundancy* (TMR) dapat dianggap sebagai suatu kelemahan, bergantung pada model ancaman dan langkah mitigasi lain yang diterapkan. TMR dapat menutupi kesalahan transien titik tunggal (*single-point transient faults*) tertentu melalui mekanisme pemungutan suara mayoritas (*majority voting*), yang juga berguna untuk mendeteksi *glitch* (gangguan sesaat).

Perlu dicatat bahwa TMR tidak memberikan perlindungan terhadap semua skenario *glitch* atau injeksi kesalahan, terutama jika kesalahan tersebut memengaruhi beberapa replika sekaligus atau memengaruhi komponen pemungut suara (*voter*) itu sendiri. Intinya, pemasok mikrokontroler (MCU) yang memahami efektivitas serangan *glitch* namun tidak menyediakan perlindungan perangkat keras apa pun, pada dasarnya memproduksi mikrokontroler yang memang lemah sejak awal.

Kebocoran Saluran Samping (Side-Channel Leakage – Analisis Waktu / Daya / Elektromagnetik)

Serangan saluran samping (*side-channel attacks*) memanfaatkan kebocoran fisik dari mikrokontroler saat melakukan komputasi untuk menyimpulkan informasi rahasia. Target yang paling relevan adalah kunci kriptografis yang digunakan oleh unit kontrol elektronik (ECU) untuk autentikasi pesan, *secure boot*, dan akses diagnostik. Dampak dari keberhasilan ekstraksi kunci melalui saluran samping bisa sangat parah, terutama jika ECU mengandalkan kunci kriptografis jangka panjang yang jarang diperbarui (*dirotasi*) atau digunakan bersama di seluruh model maupun kelas kendaraan. Dalam kasus seperti ini, keberhasilan mendapatkan satu kunci dapat memfasilitasi serangan berulang terhadap ECU yang sama, atau serangan terhadap ECU lain yang menggunakan kunci kriptografis yang sama. Hal ini menjadikan eksploitasi kebocoran saluran samping sebagai metode yang sangat menarik bagi penyerang yang ingin memperoleh informasi rahasia tersebut sebagai bagian dari rantai serangan berskala luas terhadap armada kendaraan.

Tanpa adanya mitigasi saluran samping pada perangkat keras seperti pelindung terhadap kebocoran elektromagnetik (EM) atau interferensi elektromagnetik (EMI) penyerang dapat memperoleh kunci kriptografis dengan mengamati jejak elektromagnetik selama operasi kriptografis berulang dan menerapkan analisis statistik. Demikian pula, analisis daya diferensial (*Differential Power Analysis* atau DPA) dapat dilakukan dengan mengukur perubahan konsumsi daya saat operasi kriptografis berlangsung. Saluran samping berbasis waktu (*timing side channels*) juga menghadirkan risiko serupa jika rutinitas autentikasi atau enkripsi tidak diimplementasikan dengan durasi waktu yang konstan; perbedaan waktu eksekusi dapat membocorkan informasi mengenai kunci yang digunakan.

Kelemahan-kelemahan ini bukan sekadar teori. Serangan nyata terhadap ECU sistem imobilizer telah membuktikan bahwa kunci kriptografis yang digunakan untuk mengautentikasi kunci kontak dapat diekstraksi melalui analisis daya, yang pada akhirnya melumpuhkan sistem anti-pencurian kendaraan. Faktanya, setiap operasi kriptografi termasuk pembuatan *message*

authentication code (MAC) dapat membocorkan informasi jika perangkat keras dan *firmware* yang mendasarinya tidak secara aktif memitigasi paparan melalui saluran samping (*side-channel*). Banyak mikrokontroler (MCU) tidak memiliki dukungan bawaan untuk teknik *masking*, pembuatan derau (*noise generation*), atau eksekusi dengan waktu konstan (*constant-time execution*) di dalam akselerator kriptografinya. Mesin kriptografi tersebut sering kali dioptimalkan untuk *throughput* dan determinisme, bukan untuk ketahanan terhadap kebocoran informasi.

Analisis saluran samping berbasis waktu (*timing side-channel analysis*) menjadi lebih sulit ketika operasi kriptografi dilakukan sepenuhnya di dalam HPSE (*Hardware-Protected Security Environment*), karena pemisahan perangkat keras membatasi apa yang dapat diamati oleh penyerang dari CPU tujuan umum (*general-purpose CPU*). Namun, meskipun HPSE kini telah tersedia secara luas pada MCU otomotif, beberapa pemasok ECU masih menjalankan operasi kriptografi tertentu melalui perangkat lunak pada inti CPU utama sering kali karena alasan performa atau integrasi. Implementasi berbasis perangkat lunak seperti ini tetap rentan terhadap serangan saluran samping berbasis waktu.

Penyerang yang memiliki akses fisik dalam waktu lama ke sebuah ECU dapat lebih meningkatkan tingkat keberhasilannya dengan memodifikasi *firmware* misalnya dengan menyisipkan penanda atau penundaan di sekitar operasi kriptografis sehingga pengukuran menjadi lebih mudah diselaraskan, atau mempermudah pelaksanaan injeksi kesalahan elektromagnetik (*electromagnetic fault injection*) dan analisis saluran samping (*side-channel analysis*). Dalam konteks ini, ketiadaan pertahanan terhadap analisis saluran samping di tingkat perangkat keras menjadikan penguasaan fisik sebagai keunggulan yang sangat kuat. Sama halnya dengan injeksi kesalahan, pemasok MCU mungkin sengaja tidak menyertakan mitigasi terhadap analisis saluran samping guna menghindari biaya lisensi kekayaan intelektual yang mahal, penurunan performa, serta penambahan kebutuhan ruang pada silikon (*silicon area overhead*).

Kurangnya Dukungan Untuk Memori Terenkripsi

Sebagaimana disebutkan sebelumnya, karakteristik utama dari sebagian besar arsitektur MCU (*Microcontroller Unit*) otomotif adalah kode aplikasi dieksekusi langsung dari memori *flash* tertanam. Instruksi diambil langsung dari lokasi penyimpanannya (*in-place*) alih-alih dimuat ke dalam RAM saat proses *booting*. Meskipun model eksekusi ini menyederhanakan desain dan mendukung perilaku *startup* yang deterministik, hal ini membawa implikasi keamanan yang signifikan: penyimpanan kode terenkripsi pada dasarnya tidak dapat didukung tanpa menambahkan komponen memori eksternal.

Karena CPU harus mengambil dan mendekode instruksi langsung dari memori *flash*, isi memori *flash* tersebut harus tersedia dalam bentuk teks polos (*plaintext*) pada saat eksekusi. Berbeda dengan platform komputasi kelas atas yang mampu mendekripsi halaman kode secara langsung (*on-the-fly*) saat dimuat ke dalam memori, MCU tidak memiliki mekanisme perangkat keras untuk dekripsi instruksi secara transparan. Begitu seorang penyerang berhasil membaca isi memori *flash* baik melalui antarmuka *debug* maupun ekstraksi fisik seluruh citra *firmware* dapat langsung dipahami tanpa perlu mencari kunci kriptografi tambahan.

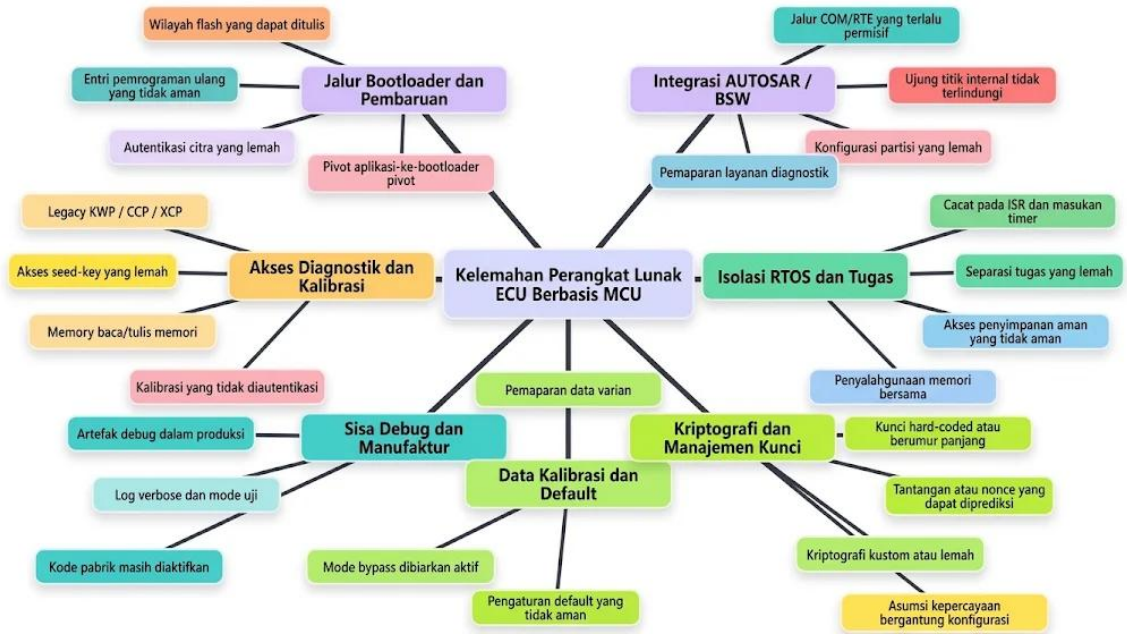
Keterbatasan arsitektural ini secara signifikan menurunkan hambatan untuk melakukan rekayasa balik (*reverse engineering*), karena *dump* memori mentah saja sudah cukup untuk mengungkap logika aplikasi, penangan diagnostik, tabel kalibrasi, algoritma keamanan, serta *state machine* protokol. Aksesibilitas ini mempercepat penemuan kerentanan dan memungkinkan penyerang untuk menargetkan bagian-bagian sistem secara presisi, seperti rutinitas autentikasi yang lemah atau fungsi *backdoor* yang seharusnya tersembunyi.

Beberapa ECU berbasis MCU mendukung perangkat *flash* eksternal yang, secara teoretis, dapat dienkripsi. Namun, pendekatan ini menambah biaya, meningkatkan kompleksitas papan sirkuit, serta membatasi performa; oleh karena itu, metode ini jarang diterapkan pada ECU otomotif yang sangat memperhatikan efisiensi biaya. Dari perspektif keamanan ofensif, hal ini membuat ECU berbasis MCU sangat rentan terhadap serangan yang bergantung pada pemahaman mendalam tentang *firmware* seperti penyalahgunaan protokol, serangan penurunan versi (*downgrade attacks*), serta penemuan fungsi debug atau fungsi manufaktur yang tersembunyi.

Seperti yang telah kita bahas, meskipun kelemahan perangkat keras memerlukan kedekatan atau kontak fisik, metode ini sangat efektif untuk mendapatkan rahasia dan data sensitif, yang kemudian menjadi batu loncatan bagi serangan berbasis perangkat lunak yang lebih canggih. Pada bagian selanjutnya, kita akan meninjau beberapa kelemahan perangkat lunak yang sering dieksploitasi oleh penyerang untuk menyusup ke dalam ECU berbasis MCU.

Kelemahan Perangkat Lunak

Sebagaimana ditunjukkan dalam Gambar 7.8, kelemahan perangkat lunak mencakup berbagai aspek, mulai dari *bootloader* yang tidak aman hingga celah dalam pengelolaan kunci (*key management*) dan implementasi kriptografi. Pada bagian ini, kita akan mengulas setiap kategori kelemahan keamanan tersebut dan mempelajari bagaimana celah-celah ini dapat dimanfaatkan untuk melakukan eksploitasi keamanan.



Gambar 7.8 Gambaran kelemahan perangkat lunak yang umum pada ECU berbasis MCU

Bootloader dan Mekanisme Pembaruan Firmware yang Tidak Aman

Sebagaimana telah dibahas pada bagian mengenai *bootloader*, komponen ini memiliki dua tanggung jawab krusial: mengelola proses *booting* perangkat lunak aplikasi dan menangani pembaruan perangkat lunak. Hal ini menjadikan *bootloader* sebagai salah satu komponen yang paling kritis dari sisi keamanan dalam sebuah ECU berbasis mikrokontroler (MCU). *Bootloader* beroperasi pada tingkat hak akses tertinggi, dijalankan di awal urutan *startup*, serta memiliki kendali langsung atas kode aplikasi apa saja yang diizinkan untuk tetap berada di dalam perangkat. Kelemahan pada lapisan ini sering kali berujung pada kompromi sistem secara menyeluruh dan permanen.

Dalam menjalankan fungsi *secure boot* (proses *booting* yang aman), adanya persyaratan ketat terkait waktu startup serta keterbatasan sumber daya memori sering kali menuntut optimalisasi yang justru mengorbankan aspek keamanan siber. Sebagai contoh, demi memenuhi tenggat waktu saat perangkat dinyalakan (*power-on*), beberapa implementasi hanya memverifikasi bagian memori yang dianggap esensial untuk proses *startup* dan alur kendali (*control flow*). Kompromi dalam desain seperti ini dapat menciptakan celah keamanan yang mungkin dieksploitasi oleh pihak penyerang, sebagaimana akan dijelaskan di bawah ini. Kelemahan umum pada *flash bootloader* meliputi hal-hal berikut:

- **Cakupan integritas yang tidak lengkap**: Data kalibrasi atau bagian lain yang bukan berupa kode mungkin tidak disertakan dalam pemeriksaan *secure boot* (boot aman) dengan asumsi bahwa bagian tersebut tidak memengaruhi alur kendali (*control flow*). Dari sudut pandang penyerang, pengecualian ini dapat menciptakan area yang dapat ditulis, yang kemudian bisa memengaruhi perilaku sistem tanpa memicu kegagalan *secure boot*.
- **Kunci verifikasi simetris yang digunakan bersama**: Beberapa *flash bootloader* mengandalkan kunci simetris untuk memverifikasi keaslian citra aplikasi (*application image*) sebelum menyerahkan kendali eksekusi. Jika semua ECU dari model dan tipe tertentu menggunakan kunci simetris yang sama, maka dengan mengekstrak kunci tersebut dari satu ECU, penyerang dapat membuat MAC (*Message Authentication Code*) yang valid untuk citra yang telah dimodifikasi agar dapat dijalankan pada ECU lain yang serupa.
- **Paparan materi kunci kriptografis**: Karena kunci yang digunakan bersama ini sangat berharga, *flash bootloader* menjadi target yang menarik untuk serangan melalui analisis saluran samping (*side-channel analysis*), injeksi kesalahan (*fault injection*), serta teknik ekstraksi kunci lainnya. Begitu kunci simetris berhasil didapatkan, penyerang dapat membuat citra yang lolos autentikasi dan dapat dimuat (*boot*) pada ECU target.

Keberhasilan mem-bypass *secure boot* dapat menjadi batu loncatan untuk upaya penelusuran lebih lanjut. Sebagai contoh, *firmware* yang dikendalikan oleh penyerang dapat memungkinkan pemeriksaan yang lebih mendalam terhadap perilaku ECU, ekstraksi data rahasia, atau identifikasi kelemahan yang nantinya dapat dieksploitasi dari jarak jauh.

Mekanisme pembaruan *firmware* membuat situasi menjadi lebih rumit. Biasanya, ECU otomotif mendukung pembaruan *firmware* melalui protokol diagnostik UDS. Pada banyak MCU, pemrograman *flash* mengharuskan driver *flash* dijalankan dari RAM saat bank *flash* sedang dihapus atau ditulis. Hal ini menciptakan periode di mana eksekusi kode dari RAM diperbolehkan untuk bagian ".data", sementara izin akses memori dilonggarkan, sehingga memperluas area serangan selama operasi pembaruan. Perlu dicatat bahwa mekanisme seperti *Data Execution Prevention* (DEP) tidak dapat digunakan dalam kasus ini karena rutin *flash* memang dirancang untuk dijalankan dari RAM saat proses penulisan perangkat lunak ke memori *flash* berlangsung.

Cacat implementasi *bootloader* memperbesar risiko ini. Karena *bootloader* memproses data yang disediakan secara eksternal selama pembaruan *firmware*, *bootloader* rentan terhadap kerentanan kerusakan memori klasik seperti luapan tumpukan atau pemeriksaan batas yang tidak tepat pada paket data yang masuk. Kerentanan ini sangat berharga bagi penyerang karena *bootloader* berjalan dalam mode istimewa dan jarang diperbarui. *Bootloader*, menurut definisinya, adalah mekanisme yang digunakan untuk memperbarui semua hal lainnya. Oleh karena itu, satu cacat yang dapat dieksploitasi dapat memungkinkan kompromi yang berkelanjutan, memungkinkan penyerang untuk menanamkan pintu belakang, menonaktifkan fitur keamanan, atau mengeksfiltrasi citra *firmware* lengkap.

Selain bug keamanan memori, *bootloader* mungkin memiliki kelemahan logis, seperti menerima citra yang tidak ditandatangani jika flag tertentu diatur atau memasuki mode pemrograman *flash* jika pesan CAN tertentu diterima selama startup. Jika memasuki sesi pemrograman dan mengeluarkan unduh perintah-perintah tersebut tidak dibatasi oleh otorisasi kriptografis yang kuat, sehingga setiap node pada bus CAN berpotensi memprogram ulang ECU target. Kompleksitas tambahan muncul akibat adanya optimasi kinerja dalam proses pembaruan. *Bootloader* sering kali menyertakan pustaka dekompresi untuk mempercepat pengunduhan *firmware* dengan cara mentransfer citra (*image*) terkompresi melalui jaringan. Pustaka-pustaka ini menghadirkan celah serangan tersendiri. Jika dekompresi dilakukan sebelum autentikasi atau verifikasi integritas, penyerang dapat mengeksploitasi urutan proses ini melalui payload yang dirancang khusus seperti "bom kompresi" yang dapat menyebabkan kehabisan memori atau luapan buffer (*buffer overflow*).

Kelemahan terakhir yang perlu dipertimbangkan adalah kemampuan untuk beralih ke *bootloader* dari aplikasi yang telah disusupi. Karena *flash bootloader* tetap ada saat aplikasi berjalan, serangan injeksi kode pada aplikasi dapat mengalihkan eksekusi ke ruang memori *flash bootloader* baik melalui instruksi percabangan langsung maupun dengan memanipulasi *flag* memori global yang menandakan pengaktifan mode *boot* setelah *soft reset*. *Bootloader* kemudian mengasumsikan bahwa urutan autentikasi yang sah telah dilakukan oleh aplikasi pada siklus sebelumnya, sehingga mungkin mengizinkan klien diagnostik yang tidak berwenang untuk melakukan sesi pengunduhan perangkat lunak.

Secara keseluruhan, *bootloader* dan mekanisme pembaruan *firmware* yang tidak aman menghadirkan beberapa karakteristik berisiko tinggi: eksekusi dengan hak akses istimewa, paparan langsung terhadap input eksternal, transisi status yang kompleks, serta kode yang

bertahan lama dan sulit untuk diperbaiki (*patched*). Bagi seorang insinyur keamanan ofensif, lapisan ini sering kali menjadi salah satu target yang paling menarik, karena satu kelemahan saja dapat menyebabkan kompromi permanen pada seluruh perangkat tanpa perlu merangkai berbagai kerentanan tambahan.

Kelemahan Terkait Autosar

Sebagaimana telah dibahas dalam bagian mengenai arsitektur perangkat lunak dan model eksekusi, perangkat lunak ECU disusun sebagai sistem berlapis dengan *bootloader* di lapisan terbawah dan aplikasi di lapisan teratas, sementara AUTOSAR berperan sebagai sistem operasi (OS), MCAL, dan layanan *middleware*. Kelemahan pada salah satu lapisan ini dapat memberikan titik pijak awal bagi penyerang. Karena kode diambil langsung dari memori *flash*, mekanisme mitigasi eksploitasi yang lazim ditemukan pada sistem tujuan umum seperti *address space layout randomization* (ASLR) tidak tersedia. Beberapa bentuk pertahanan, termasuk *data execution prevention* (DEP) dan *stack canaries*, memang mulai banyak digunakan, namun cakupannya tidak konsisten. Akibatnya, ECU berbasis mikrokontroler (MCU) tetap rentan terhadap kelemahan korupsi memori klasik, terutama karena sistem ini beroperasi dengan model memori datar (*flat memory model*) dan akses langsung ke memori fisik. Ketiadaan virtualisasi serta isolasi yang terbatas membuat dampak dari celah keamanan semacam itu menjadi jauh lebih parah dibandingkan pada sistem yang memiliki pemisahan hak akses yang lebih ketat.

Dalam sistem AUTOSAR Classic, konfigurasi yang tidak tepat muncul sebagai faktor risiko utama yang menentukan postur keamanan ECU. Keputusan mengenai pengidentifikasi CAN (*CAN identifiers*) mana yang diterima, layanan diagnostik mana yang dilindungi oleh pembatasan akses, antarmuka komunikasi mana yang tetap diaktifkan dalam tahap produksi, serta pesan mana yang diautentikasi semuanya dapat berdampak buruk terhadap keamanan jika tidak ditentukan dengan benar. Oleh karena itu, saat mencari kelemahan dalam sistem berbasis AUTOSAR, sangat penting untuk memeriksa konfigurasi AUTOSAR tersebut. Contoh konfigurasi yang lemah mencakup filter penerimaan CAN yang terlalu longgar, layanan diagnostik tanpa pembatasan, titik akhir (*endpoint*) TCP/IP yang tidak diamankan pada ECU dengan kapabilitas Ethernet, serta parameter manajemen jaringan yang memungkinkan perilaku aktivasi (*wake-up*) yang tidak diinginkan. Adopsi luas AUTOSAR Classic menghadirkan risiko sistemik tambahan berupa monokultur arsitektural. Kerentanan yang ditemukan dalam modul terstandarisasi seperti *driver* CAN, lapisan transpor (*transport layer*), atau *stack* manajemen memori dapat menyebar ke berbagai platform kendaraan dan produsen, terutama jika pemasok perangkat lunaknya sama.

Meskipun AUTOSAR Classic membantu menyederhanakan pengembangan dan pemindahan (*porting*) perangkat lunak ECU, standardisasi itu sendiri dapat menjadi kelemahan dari perspektif keamanan. Karena pengidentifikasi modul dan layanan, format konfigurasi seperti ARXML, serta struktur *Basic Software* dan RTE bersifat publik dan konsisten di antara berbagai pemasok, penyerang yang memperoleh *dump firmware* dapat dengan mudah mengurai dan mengidentifikasi *stack* komunikasi, modul diagnostik, *driver* kriptografi, dan layanan manajemen memori. Hal ini mengarahkan perhatian kita pada cacat implementasi,

khususnya dalam modul-modul AUTOSAR yang terpapar ke jaringan, seperti *driver* jaringan (CAN, LIN, Ethernet), lapisan transpor, dan penangan pesan diagnostik. Pelanggaran praktik penulisan kode yang aman (*secure coding*) seperti tidak adanya validasi input, *buffer overflow*, *integer overflow*, dan sebagainya menjadi sangat mengkhawatirkan dalam konteks ini, mengingat sebagian besar perangkat lunak berjalan dalam *supervisor mode* dengan pemisahan akses yang terbatas.

Sebagaimana telah dibahas dalam bagian mengenai arsitektur perangkat lunak dan model eksekusi, mekanisme pertahanan sistem operasi biasanya dirancang untuk menangani aspek keselamatan guna mencegah kerusakan data yang tidak disengaja (bukan akibat tindakan jahat) serta menjamin tidak adanya interferensi antara kode yang memiliki tingkat kekritisan keselamatan (ASIL) berbeda. Tanpa penerapan partisi wilayah memori melalui *Memory Protection Unit* (MPU) secara ketat, satu kejadian *buffer overflow* pada tugas yang berinteraksi dengan periferal misalnya penangan pesan CAN dapat merusak memori di seluruh unit kontrol elektronik (ECU) dan mengganggu logika kendali yang krusial bagi keselamatan. Kode yang bersifat kritis terhadap keselamatan itu sendiri pun mungkin masih mengandung *bug* yang menyebabkan kerusakan memori; jika eksekusi mencapai domain tersebut, isolasi berbasis MPU mungkin tidak lagi memberikan perlindungan yang memadai. Akibatnya, penyerang dapat mematahkan jaminan "bebas dari interferensi" (*freedom-from-interference*) dengan mengeksploitasi kerentanan di dalam partisi yang "dipercaya" (*trusted*), yang kemudian memungkinkan serangan berbasis *return-oriented programming* (ROP) atau *jump-oriented programming* (JOP).

Demikian pula, *stack overflow* pada penangan protokol UDS dapat menyebabkan pengalihan alur eksekusi ke dalam *bootloader*, yang selanjutnya dapat dimanfaatkan untuk melakukan operasi penghapusan (*erase*) atau penulisan (*flash write*) pada memori *flash*. Operasi penghapusan menjadi sangat signifikan jika kunci kriptografis disimpan di dalam memori *flash* biasa; penghapusan kunci tersebut dapat mengecoh perangkat lunak sehingga menganggap kunci kriptografis yang ada telah diatur ke nilai standar (berisi angka satu semua atau nol semua).

Kurangnya Kontrol Akses Perangkat Lunak Untuk HPSE

Secara historis, banyak ECU berbasis MCU beroperasi tanpa menerapkan prinsip hak akses minimum. Terkadang muncul argumen bahwa memperkuat perangkat lunak MCU adalah upaya yang sia-sia, dengan asumsi bahwa begitu eksekusi kode arbitrer berhasil dilakukan, seluruh jaminan keamanan akan hilang. Asumsi ini dapat dengan mudah dibantah dengan mempertimbangkan model keamanan pada MCU yang mendukung aset bernilai tinggi yang dilindungi oleh HPSE.

HPSE diharapkan memiliki akses eksklusif ke penyimpanan *flash* aman yang memuat kunci kriptografis jangka panjang. Model keamanan ini mengasumsikan bahwa meskipun perangkat lunak aplikasi utama telah disusupi, rahasia-rahasia tersebut harus tetap terlindungi. Dalam praktiknya, jaminan ini bergantung pada ketangguhan *firmware* HPSE serta protokol memori bersama (*shared memory*) antara perangkat lunak aplikasi dan HPSE. Cacat pada validasi input atau kerentanan kerusakan memori (*memory corruption*) di dalam

firmware HPSE, ditambah dengan implementasi protokol komunikasi memori bersama yang tidak tepat, dapat memungkinkan penyerang untuk menyisipkan dan mengeksekusi kode mereka sendiri di dalam lingkungan HPSE. Begitu batasan tersebut ditembus, ekstraksi kunci menjadi sangat mudah, karena *firmware* itu sendiri memiliki akses sah ke penyimpanan aman yang memuat kunci-kunci tersebut.

Dampak dari kegagalan semacam ini menjadi lebih besar jika kunci bersifat global atau digunakan bersama oleh sekumpulan ECU. Bahkan ketika kunci bersifat unik untuk setiap perangkat, ekstraksi berskala besar di berbagai unit dapat memungkinkan rekonstruksi rahasia umum yang dapat digunakan untuk menyerang kendaraan lain. Dalam konteks ini, kontrol akses berbasis perangkat lunak yang lemah tidak hanya menyebabkan kompromi lokal, tetapi juga merusak asumsi dasar mengenai isolasi yang menjadi landasan bagi mekanisme keamanan tingkat yang lebih tinggi.

Kode Debug yang Tertinggal atau Pintu Belakang (Backdoor)

Sering kali, *firmware* ECU menyertakan kode khusus yang ditujukan untuk keperluan pengembangan, pengujian, dan manufaktur. Jika kode ini tidak dihapus atau dinonaktifkan sepenuhnya sebelum tahap produksi, hal tersebut dapat menimbulkan kelemahan keamanan yang serius. Dalam bentuk yang paling ringan, fungsi debug yang tertinggal dapat mengekspos log kesalahan terperinci yang mengungkapkan isi memori atau status internal pada saat kegagalan terjadi. Meskipun berguna bagi pengembang selama tahap awal pengembangan dan pemecahan masalah, informasi semacam ini juga dapat memberikan wawasan yang tepat kepada penyerang mengenai tata letak memori, alur kendali (*control flow*), atau kondisi kesalahan.

Dalam kasus yang lebih parah, artefak pengembangan dapat berupa fungsi pintu belakang (*backdoor*) yang nyata. Contoh umum adalah adanya layanan diagnostik pembacaan memori yang tidak didokumentasikan yang seharusnya hanya digunakan untuk diagnostik pabrik yang memungkinkan pengambilan (*dumping*) data dari rentang memori fisik mana pun. Contoh lainnya mencakup rutinitas pemeliharaan yang mengabaikan pemeriksaan status kendaraan atau kondisi kunci kontak (*ignition*) saat menerima pesan CAN tertentu, sehingga memungkinkan layanan diagnostik diaktifkan bahkan ketika kendaraan sedang bergerak. Perilaku-perilaku ini biasanya ditujukan untuk lingkungan pemrograman di lini perakitan atau lingkungan layanan, namun jika tetap dapat diakses dalam *firmware* produksi, hal tersebut melanggar asumsi keselamatan dan keamanan yang krusial.

Selain itu, perangkat lunak manufaktur yang digunakan untuk pengaturan pabrik dan pengujian akhir lini produksi harus dihapus atau dinonaktifkan secara permanen sebelum ECU beralih ke mode operasional. Jika tidak, perangkat lunak tersebut akan memberikan fungsionalitas yang kuat dan memiliki hak akses khusus kepada penyerang sesuatu yang seharusnya tidak pernah dibuka untuk akses di luar lingkungan pabrik. Dari perspektif keamanan, sisa kode debug dan pintu belakang yang tidak terdokumentasi menjadi sasaran menarik, baik pada tahap penemuan maupun tahap eksploitasi.

Saat bekerja dalam sistem berbasis AUTOSAR, ada baiknya untuk mencari komponen seperti modul *Diagnostic Log and Trace* (DLT), yang dapat mempermudah proses re*verse*

engineering dengan menyediakan pencatatan (*logging*) waktu nyata yang terstruktur serta memiliki format pesan yang jelas. Format-format ini sering kali mencakup pengidentifikasi modul, kode kesalahan, nama fungsi, atau alamat memori. Jika DLT tetap aktif dalam *firmware* produksi, rangkaian log tersebut dapat mengarahkan penyerang secara langsung ke komponen *Basic Software* (BSW) tertentu yang menunjukkan kesalahan khusus sesuatu yang mungkin menarik untuk dieksploitasi. Bahkan dalam konfigurasi DLT yang tidak aktif, sering kali masih terdapat sisa-sisa berupa tabel *string*, templat pesan, atau struktur pencarian (*lookup structures*) yang dapat membantu memetakan fungsionalitas internal serta merekonstruksi alur kendali (*control flow*) pada *Electronic Control Unit* (ECU).

XCP dan Protokol Warisan Dalam Tahap Produksi

Mirip dengan kode debug, keberadaan protokol kalibrasi dan diagnostik warisan (*legacy*) seperti KWP2000, CCP, dan XCP pada ECU berbasis MCU modern merupakan kelemahan keamanan sistemik. Protokol-protokol ini umumnya tidak memiliki mekanisme autentikasi maupun enkripsi, sehingga memungkinkan penyerang menyalahgunakan perintah kalibrasi atau diagnostik yang sah untuk melakukan pembacaan dan penulisan memori tanpa izin. Jika tetap dapat diakses, protokol ini menjadi titik masuk yang dapat digunakan untuk mengekstrak *firmware* dan memodifikasi alur eksekusi.

Meskipun kapabilitas XCP sangat penting selama tahap pengembangan, fitur ini justru menjadi kerentanan jika tetap ada dalam *firmware* tahap produksi. Jika XCP melalui CAN atau XCP melalui Ethernet masih dapat diakses pada kendaraan yang telah digunakan, penyerang yang memiliki akses ke bus komunikasi dapat mengirimkan perintah untuk mendeteksi fitur guna mengetahui apakah layanan kalibrasi, pemrograman, atau akuisisi data masih aktif. Bahkan ketika akses ke protokol kalibrasi dinonaktifkan, *firmware* yang berhasil diekstraksi mungkin menunjukkan bahwa penangan (*handler*) XCP dan rutin pemrograman masih disertakan dalam kode biner. Dari perspektif rekayasa balik (*reverse engineering*), fungsionalitas XCP sering kali mudah diidentifikasi. Pengidentifikasi perintah, tabel pengiriman (*dispatch table*), serta kumpulan rutin untuk kalibrasi, manajemen halaman, akuisisi data (DAQ), dan pemrograman biasanya tersusun dalam area memori tertentu. Larik penunjuk fungsi (*function pointer array*) sering digunakan untuk memetakan nilai perintah ke rutin penangan yang sesuai. Lapisan perantara ini memperbanyak jalur akses menuju fungsi-fungsi dengan hak istimewa tinggi, yang dapat dimanfaatkan untuk menyusun rangkaian serangan.

Keamanan yang Dapat Dikalibrasi

Banyak ECU berbasis MCU mengekspos parameter kalibrasi yang secara langsung memengaruhi aspek keamanan. Parameter ini biasanya disediakan untuk menyederhanakan alur kerja teknis selama pengembangan, di mana fleksibilitas dan kemampuan observasi lebih diutamakan. Nilai-nilai kalibrasi tersebut dapat mengatur fitur-fitur seperti pengaktifan pencatatan kesalahan (*error logging*) yang mendetail, pelonggaran pemeriksaan autentikasi klien diagnostik, mode untuk melewati proses *secure boot*, atau penonaktifan autentikasi pesan jaringan. Jika parameter-parameter ini tetap ada dalam *firmware* tahap produksi, maka parameter tersebut menjadi target bernilai tinggi bagi penyerang yang ingin melemahkan postur keamanan ECU tanpa harus menembus perlindungan kriptografi yang lebih kuat.

Kelemahan struktural yang sering muncul adalah penggunaan citra kalibrasi (*calibration images*) yang ditandatangani secara digital, yang memungkinkan penurunan tingkat fitur keamanan tanpa terikat pada perangkat keras tertentu. Meskipun citra semacam itu diperlukan dalam tahap pengembangan atau pemecahan masalah di lapangan, ketiadaan batasan yang mengikatnya pada perangkat spesifik berarti penyerang yang memperoleh citra tersebut dapat dengan mudah memasangnya kembali (*reflash*) ke perangkat target dan langsung menonaktifkan fitur-fitur keamanan krusial. Tanpa adanya keterikatan antara artefak kalibrasi dengan karakteristik perangkat yang tidak dapat diubah (immutable), sistem secara implisit memercayai konfigurasi keamanan yang sebenarnya tidak dirancang untuk dapat dipindahkan antarperangkat.

Data kalibrasi juga cenderung mendapatkan pengawasan yang lebih longgar dibandingkan kode yang bersifat krusial bagi keamanan. Begitu penyerang mendapatkan akses untuk menulis data kalibrasi, nilai-nilai tertentu dapat dimodifikasi untuk melonggarkan atau menonaktifkan perlindungan cukup dengan mengubah beberapa byte konfigurasi tanpa perlu menyentuh kode yang dapat dieksekusi. Menemukan parameter kalibrasi yang tepat untuk melakukan serangan penurunan tingkat keamanan memang bisa menjadi tantangan; namun, akses ke kode debug pada perangkat target akan sangat membantu, karena dapat mempercepat proses penemuan "pengaturan" yang tepat untuk menonaktifkan fitur keamanan tertentu.

Perlindungan Kriptografis yang Lemah atau Tidak Ada Sama Sekali

Sebagaimana telah dibahas dalam bagian mengenai kelemahan HauKwauc, enkripsi citra *firmware* dalam memori *flash* internal tidak dimungkinkan untuk unit mikrokontroler (MCU). Akan tetapi, bukan berarti citra *firmware* boleh dibiarkan dalam bentuk teks polos (*plaintext*) saat dikirimkan melalui saluran yang tidak aman selama proses pembaruan perangkat lunak. Sayangnya, hal ini masih menjadi salah satu kelemahan dalam sesi pemrograman *flash* melalui diagnostik aktif. Karena data biner dipecah ke dalam bingkai-bingkai (*frames*) diagnostik melalui bus CAN, penyerang yang mampu memantau (*sniff*) lalu lintas bus selama sesi pengunduhan dapat menyusun ulang *firmware* tersebut dalam bentuk teks polos.

Selain itu, dalam skenario pembaruan perangkat lunak melalui metode OTA (*Over-the-Air*), idealnya citra pembaruan harus dienkripsi dari sumber hingga ke ECU tujuan. Namun, sering kali terdapat kelemahan di mana proses dekripsi terjadi lebih awal misalnya di dalam Unit Telematika atau ECU gerbang (*gateway*) sehingga citra tersebut menjadi tidak terenkripsi, bahkan tidak lagi memiliki tanda tangan digital, setelah diproses oleh elemen antarmuka kendaraan. Kelemahan ini membuka celah bagi penyerang untuk mencuri (eksfiltrasi) maupun memanipulasi citra tersebut sebelum dipasang pada ECU target. Kelemahan umum dalam perlindungan kriptografis meliputi hal-hal berikut:

- Pengiriman data diagnostik dalam bentuk teks polos (*plaintext*): *Firmware* yang ditransfer melalui CAN atau jaringan dalam kendaraan lainnya dapat ditangkap dan direkonstruksi jika sesi pembaruan tidak dienkripsi.

- Dekripsi OTA yang terlalu dini: *Image* pembaruan dapat didekripsi oleh ECU perantara, sehingga *firmware* dalam bentuk teks polos terpapar di dalam jaringan kendaraan.
- Autentikasi yang tidak ada atau tidak lengkap: *Firmware* mungkin dienkripsi, tetapi tidak ditandatangani atau diautentikasi dengan benar, sehingga memungkinkan penyerang untuk memodifikasi *image* tanpa terdeteksi.
- Perlindungan yang lemah setelah data masuk ke dalam kendaraan: Meskipun saluran OTA eksternal terlindungi, jalur internal dari unit telematika atau *gateway* menuju ECU target dapat membuat *image* tersebut rentan terhadap penyadapan (*sniffing*) atau manipulasi (*tampering*).

Meskipun enkripsi dan autentikasi telah diterapkan, pengelolaan kunci yang buruk dapat menimbulkan jenis kelemahan serius lainnya. Banyak ECU menyimpan rahasia jangka panjang di dalam memori *flash* biasa atau jika tersedia di dalam penyimpanan aman yang didukung oleh HPSE (*Hardware-Protected Secure Element*) atau modul keamanan perangkat keras serupa. Jika kunci disimpan dalam memori *flash* biasa, kunci tersebut sering kali dapat diperoleh kembali melalui ekstraksi *firmware*. Dalam satu kasus yang terdokumentasi, sebuah *bootloader* menggunakan kunci AES yang disematkan langsung (*hard-coded*) di dalam biner *firmware* untuk mengautentikasi citra pembaruan (*update images*). Begitu penyerang berhasil mengidentifikasi kunci tersebut, mereka dapat membuat pembaruan berbahaya yang lolos proses autentikasi. Kelemahan dalam pengelolaan kunci umumnya mencakup hal-hal berikut:

- **Kunci simetris yang disematkan (*hard-coded*)**: Kunci yang tertanam dalam biner *firmware* dapat ditemukan melalui rekayasa balik (*reverse engineering*) dan digunakan kembali untuk mengautentikasi citra *firmware* yang berbahaya.
- **Kunci yang digunakan bersama di antara berbagai ECU**: Kunci yang diekstraksi dari satu ECU mungkin dapat diterapkan pada ECU lain yang memiliki model, platform, atau implementasi dari pemasok yang sama.
- **Jangkar kepercayaan (*trust anchor*) yang dapat diubah**: Meskipun kunci publik yang digunakan untuk *secure boot* tidak bersifat rahasia, kunci tersebut harus tetap tidak dapat diubah (*immutable*). Jika penyerang dapat menulis ulang sektor yang memuat kunci publik tersebut, mereka mungkin bisa mengganti jangkar kepercayaan dan memuat *firmware* buatan mereka sendiri.
- **Perlindungan berbasis perangkat keras yang tidak memadai**: Rahasia jangka panjang yang disimpan di luar perangkat keras yang aman dapat terekspos melalui ekstraksi *firmware*, akses *debug*, atau pengungkapan memori.

ECU berbasis MCU juga sering kali memiliki kesalahan dalam desain dan implementasi kriptografi. Tingkat keacakan yang terbatas dapat membuat tantangan diagnostik menjadi mudah ditebak; penggunaan *nonce* yang berulang dapat merusak protokol yang sebenarnya sudah aman; dan ketiadaan kebijakan penggunaan kunci dapat mengekspos rahasia melalui serangan tipe *oracle*. Konstruksi yang lemah, seperti enkoding berbasis XOR atau algoritma kriptografi eksklusif (*proprietary*), sering kali secara keliru dianggap sebagai langkah perlindungan, sehingga menciptakan rasa aman semu. Berbagai penelitian publik telah berulang kali menunjukkan bahwa kriptografi eksklusif dalam sistem otomotif terutama pada

sistem *immobilizer* dan ECU untuk *key fob* (kunci kendali jarak jauh) dapat direkayasa balik dan dipecahkan keamanannya.

Terkait kontrol akses diagnostik, jalan pintas dalam implementasi kriptografi semakin mengaburkan batasan antara perlindungan yang sah dan perilaku *backdoor* (pintu belakang). Contoh yang umum adalah penggunaan *seed* (nilai awal) dan kunci yang bersifat tetap atau bawaan (*default*) untuk fungsi *SecurityAccess* dalam protokol UDS (*Unified Diagnostic Services*) di seluruh lini kendaraan. Setelah penyerang berhasil mengekstrak algoritma serta pasangan *seed/key* dari *firmware* melalui rekayasa balik (*reverse engineering*), mereka dapat membuka akses diagnostik pada setiap ECU yang menggunakan implementasi tersebut. Bahkan ketika metode kriptografi diterapkan untuk menjalankan mekanisme *challenge-response*, penggunaan sumber acak yang tidak aman secara kriptografis untuk nilai *challenge* dapat memungkinkan penyerang memaksa ECU untuk mengulangi pembuatan *challenge*, sehingga nilai respons yang sebelumnya telah ditangkap dapat digunakan kembali. Setelah mempelajari berbagai kelemahan perangkat keras dan keamanan yang kerap ditemukan pada ECU berbasis MCU, kini kita akan beralih fokus pada penyusunan rantai serangan (*attack chain*) yang efektif, dengan memanfaatkan kelemahan-kelemahan tersebut untuk mencapai tujuan serangan.

7.4 RANTAI SERANGAN DAN TAHAPAN EKSPLOITASI ECU BERBASIS MCU

Menyerang ECU berbasis MCU jarang sekali hanya melibatkan satu eksploitasi tunggal. Biasanya, proses ini merupakan rangkaian langkah yang dimulai dari akses awal hingga kendali berkelanjutan atas perangkat tersebut. Rantai serangan yang sukses sering kali memerlukan kombinasi antara akses fisik, analisis *offline*, dan eksploitasi berbasis jaringan. Pola yang umum dilakukan meliputi ekstraksi *firmware* melalui metode fisik, identifikasi kelemahan lewat rekayasa balik (*reverse engineering*), modifikasi (*patching*) *firmware* untuk menguji eksploit, serta demonstrasi eksploitasi pada target melalui serangan yang menyerupai peretasan di dunia nyata. Tergantung pada cakupan pengujian penetrasi yang dilakukan, upaya eksploitasi dapat berakhir pada tingkat komponen perangkat lunak, komponen perangkat keras, ECU target, atau bahkan mencakup keseluruhan kendaraan.

Sebagai contoh, produsen kendaraan (OEM) mungkin ingin mengetahui bagaimana kelemahan yang ditemukan pada suatu ECU dapat memberikan kendali nyata atas kendaraan tersebut. Hal ini mungkin memerlukan demonstrasi pergerakan lateral (*lateral movement*) ke ECU lain seperti *gateway* untuk menyasar fungsi kendaraan tertentu yang tidak dapat diakses secara langsung oleh ECU yang telah dikompromikan. Di sisi lain, pemasok ECU mungkin lebih tertarik untuk melihat bagaimana eksploitasi dapat melanggar tujuan keamanan terkait aset tertentu, misalnya kemampuan untuk mengeksekusi kode arbitrer atau melumpuhkan kontrol keamanan yang sudah ada pada tingkat ECU. Bagi pemasok cip, cakupan pengujian biasanya terbatas pada MCU tertentu, di mana eksploitasi harus mampu menunjukkan keberhasilan dalam menembus mekanisme keamanan perangkat keras seperti memintas mekanisme penguncian JTAG atau mengekstraksi kunci kriptografi dari HPSE (*Hardware-based Secure Element*).

Bergantung pada jenis pengujian penetrasi yang dilakukan apakah *black box*, *gray box*, atau *white box* beberapa tahapan tertentu mungkin dilewati. Sebagai contoh, dalam pengujian penetrasi *white box*, perangkat keras ECU akan disediakan bersama dengan lembar data (*data sheet*) yang merinci jenis MCU, daftar komponen pada PCB, serta mungkin daftar komponen perangkat lunak yang digunakan atau bahkan berkas biner (*binary*) perangkat lunak tersebut. Sebaliknya, dalam pengujian penetrasi *black box*, pengujian mungkin dimulai tanpa informasi apa pun, hanya dengan unit ECU yang masih tertutup rapat dalam selongsongnya. Pada bagian selanjutnya, alih-alih menguraikan perbedaan di setiap tahapan berdasarkan jenis pengujian penetrasi, kita akan mengasumsikan pelaksanaan pengujian penetrasi *black box* di mana informasi yang diberikan sangat terbatas yang pada dasarnya meniru skenario peretas sungguhan yang berupaya menyusupi target.

Langkah 1: Pengintaian Fisik dan Akses

Ketika cakupan serangan berada pada tingkat ECU, serangan biasanya dimulai dengan pengintaian perangkat keras. Penyerang mengidentifikasi ECU target, baik yang terpasang di dalam kendaraan maupun unit terpisah yang diperoleh melalui pasar sekunder, seperti tempat barang rongsokan (*salvage yard*). Pemeriksaan fisik berfokus pada papan sirkuit cetak (*printed circuit board*/PCB), termasuk penandaan pada MCU, perangkat memori eksternal, susunan pin konektor (*pinout*), serta titik uji (*test point*) yang terbuka. Tujuannya adalah untuk memahami bagaimana ECU mendapatkan daya, bagaimana cara kerjanya dalam berkomunikasi, dan antarmuka mana yang dapat memberikan akses ke fungsi internal.

Penyerang memetakan pin konektor ke jalur daya, ground, bus CAN atau LIN, switch Ethernet, serta potensi antarmuka pemrograman atau debugging dengan menggunakan kombinasi multimeter, osiloskop, dan penganalisis logika (*logic analyzer*). Lembar data (*datasheet*) yang tersedia untuk umum sering digunakan untuk mengidentifikasi port debug seperti JTAG, SWD, atau port debug khusus (*proprietary*), serta pin mode boot khusus yang dapat mengubah perilaku saat perangkat dinyalakan. Bahkan sebelum membuka ECU, pengintaian parsial dapat dilakukan melalui port OBD-II dengan meminta informasi pengenal seperti nomor suku cadang atau versi perangkat lunak, yang secara tidak langsung dapat mengungkap keluarga MCU atau asal-usul *firmware* yang digunakan.

Jika antarmuka debug tampak dapat diakses, penyerang mungkin akan menguji dengan hati-hati apakah antarmuka tersebut terkunci. Jika terlindungi, teknik injeksi kesalahan (*fault injection*) seperti manipulasi tegangan, clock, atau gangguan elektromagnetik (*glitching*) dapat dievaluasi sebagai metode untuk melewati perlindungan tersebut. Tujuan dari fase ini adalah pengumpulan informasi. Hasil yang diharapkan adalah cetak biru perangkat keras yang jelas, yang memuat daftar semua komponen perangkat keras penting serta antarmuka perangkat keras yang telah ditemukan. Pada tahap ini, pembuatan perangkat khusus (*rig*) akan sangat berguna untuk menelusuri lebih lanjut antarmuka yang menarik seperti JTAG, UART, CAN, SPI, dan sebagainya sebagai persiapan untuk fase berikutnya.

Langkah 2: Ekstraksi atau Perolehan *Firmware*

Setelah peta perangkat keras ditetapkan, tujuan penyerang berikutnya adalah memperoleh *firmware*. Akses terhadap *firmware* merupakan prasyarat untuk melakukan

analisis secara luring (*offline*). Terdapat beberapa metode ekstraksi yang umum digunakan. Jika antarmuka debug tidak terkunci, *firmware* sering kali dapat dibaca langsung dari memori. Jika terkunci, serangan *glitch* (gangguan teknis yang disengaja) dapat digunakan untuk menonaktifkan perlindungan sementara selama operasi pembacaan.

Apabila *firmware* disimpan dalam memori eksternal, penyerang dapat melepas komponen memori *flash* tersebut atau membacanya saat masih terpasang di sirkuit (*in-circuit*) menggunakan perangkat pemrogram yang sesuai. Pendekatan ini memerlukan pengetahuan mengenai jenis perangkat memori dan antarmukanya, namun sama sekali tidak melibatkan interaksi dengan mekanisme keamanan pada MCU. Antarmuka diagnostik juga dapat menjadi celah lain. Beberapa ECU menyediakan fungsi pembacaan memori berdasarkan alamat atau pengunggahan *firmware* melalui layanan UDS (layanan Rp 350.000). Skema autentikasi yang lemah atau algoritma *seed-key* yang diimplementasikan dengan buruk terkadang dapat disalahgunakan untuk mendapatkan citra (*image*) *firmware* secara utuh.

Bootloader, kode aplikasi, atau bagian konfigurasi dapat memberikan wawasan mengenai alur eksekusi, asumsi keamanan, dan potensi kelemahan. Setelah diekstraksi, bahkan citra *firmware* yang tidak lengkap pun memberikan keuntungan yang signifikan. Hal ini mengubah tantangan dari sekadar pengujian buta (*blind probing*) menjadi sebuah latihan analisis *white-box* (analisis dengan akses penuh terhadap detail internal). Pada tahap ini, penyerang hanya tinggal satu langkah lagi untuk mendapatkan visibilitas yang setara dengan melihat kode sumber (*source-level visibility*) terhadap perilaku ECU tersebut.

Langkah 3: Rekayasa Balik (*Reverse Engineering*) dan Penemuan Kerentanan

Berbekal citra *firmware* dan mungkin data kalibrasi, penyerang melanjutkan ke tahap rekayasa balik. Pada langkah inilah sebagian besar analisis keamanan dilakukan untuk menemukan kerentanan. Berkas biner biasanya dimuat ke dalam perangkat lunak disassembler atau kerangka kerja analisis yang dikonfigurasi untuk arsitektur CPU yang sesuai, seperti Ghidra atau IDA Pro. Analisis awal berfokus pada identifikasi komponen perangkat lunak, termasuk penangan diagnostik (*diagnostic handlers*), tumpukan komunikasi (*communication stacks*), pintu belakang (*backdoor*) tersembunyi, logika pembaruan *firmware*, dan rutinitas kriptografi. Di sinilah pemahaman mengenai anatomi perangkat lunak akan sangat berguna. Seperti yang telah dibahas pada bagian sebelumnya, jika perangkat lunak yang digunakan berbasis AUTOSAR, maka penyerang dapat mengakses spesifikasi terbuka dari standar AUTOSAR untuk mengidentifikasi fungsi-fungsi krusial yang mungkin dapat dieksploitasi atau diperiksa lebih lanjut.

Referensi string sering kali memberikan petunjuk awal mengenai area yang menarik; misalnya, ditemukannya karakter ASCII "AUTH_FAIL" memberi petunjuk kepada penyerang bahwa terdapat rutinitas autentikasi yang mengendalikan operasi sensitif dari sisi keamanan. Pesan diagnostik, kode galat (*error codes*), atau indikator mode juga dapat mengungkap keberadaan pemeriksaan keamanan dan mesin status (*state machines*). Penyerang akan mencari pengenal protokol yang diketahui, penangan pengunduhan *firmware*, dan rutinitas penyalinan memori yang mungkin rentan terhadap penyalahgunaan. Demikian pula,

mekanisme pembaruan perangkat lunak diperiksa secara cermat untuk menentukan bagaimana kode baru diterima dan apakah pemeriksaan keaslian diterapkan dengan benar.

Salah satu teknik yang umum digunakan adalah mencari tabel pencarian (*lookup tables*) yang berisi alamat fungsi panggilan balik (*callback function*) untuk rutinitas diagnostik. Jika cakupan pengujian mencakup HPSE (*Hardware Security Module*/*Secure Element*), maka penyerang akan mencari fungsi-fungsi yang berinteraksi dengan HPSE tersebut, serta bagian data yang digunakan untuk pertukaran parameter tugas kriptografi dengan prosesor utama. Perhatian khusus diberikan pada keberadaan kredensial yang tertanam secara permanen (*hard-coded*), seperti kata sandi bawaan, pasangan kunci tantangan (*seed-key challenge*) yang tetap, atau bahkan kunci kriptografi dalam bentuk teks polos (*plaintext*). Penemuan kredensial semacam ini merupakan indikator kuat bahwa eksploitasi yang lebih canggih mungkin dapat dilakukan jika penyalahgunaan kredensial tersebut digabungkan dengan langkah-langkah serangan lainnya.

Setelah kode didekompilasi, analisis kode statis dapat dijalankan untuk mendeteksi kelemahan keamanan perangkat lunak. Dengan menggunakan perangkat berbasis LLM (*Large Language Model*), peringatan-peringatan yang muncul dapat disaring untuk memusatkan perhatian pada cacat pengodean yang benar-benar nyata (*true positive*) dan berpotensi untuk dieksploitasi. Setelah peta lengkap mengenai fungsi perangkat lunak dan bagian data berhasil disusun dengan memadai, penyerang akan mencoba membuat grafik alur kendali (*control flow graph*) untuk mengevaluasi apakah kerentanan kerusakan memori (*memory corruption*) yang ditemukan dapat dieksploitasi. Jalur kode yang memproses input eksternal akan diperiksa secara cermat untuk melihat apakah terdapat pemeriksaan batas (*bounds checking*) yang tidak tepat atau penanganan buffer yang tidak aman.

Dalam beberapa kasus, jika ECU berfungsi penuh dalam pengaturan *Hardware-in-the-Loop* (HIL), teknik *fuzzing* dapat digunakan untuk melengkapi analisis statis dengan memicu *crash* atau perilaku yang tidak wajar. Hal ini dapat dilakukan pada tingkat jaringan melalui lalu lintas CAN atau diagnostik yang dirancang khusus, ataupun dengan menyiapkan *fuzzing harness* untuk komponen perangkat lunak tertentu. Dalam kedua skenario tersebut, terjadinya *crash* biasanya menandakan bahwa kode tersebut mengandung potensi kerentanan kerusakan memori yang dapat disalahgunakan. Seperti yang terlihat dalam pola ini, proses rekayasa balik (*reverse engineering*) ibarat mencari jarum dalam tumpukan jerami. Memiliki perangkat yang tepat serta menerapkan teknik yang efisien untuk memisahkan informasi penting dari data yang tidak relevan sangatlah krusial agar proses tetap berjalan sesuai jalur dan tidak membuang waktu pada upaya yang sia-sia.

Untuk mempercepat proses rekayasa balik dan eksploitasi, penyerang biasanya memerlukan cara untuk memasang *firmware* yang telah dimodifikasi ke dalam perangkat target. Tujuannya adalah untuk menguji temuan mereka dan menyingkirkan jalur yang tidak membuahkan hasil saat melakukan eksplorasi yang lebih mendalam. Menemukan cara untuk melewati kontrol akses pemrograman *flash* baik melalui diagnostik maupun dengan menggunakan *serial programmer* melalui antarmuka JTAG menjadi langkah yang sangat krusial. Setelah metode untuk melakukan *patching firmware* secara andal ditemukan, fase ini

akan berubah menjadi proses iteratif; hipotesis yang disusun selama tahap analisis diuji pada perangkat target, disempurnakan, lalu diuji kembali hingga diperoleh sekumpulan kelemahan yang dapat dieksploitasi.

Pada akhir fase rekayasa balik, kita harus memiliki daftar kelemahan yang dapat dieksploitasi dan telah diprioritaskan; masing-masing kelemahan disertai dengan dampak yang jelas, seperti kemampuan untuk melakukan pembaruan *firmware* tanpa autentikasi atau mem-bypass proses *secure boot*. Pada tahap ini, penyerang dapat beralih dari eksplorasi ke perencanaan eksploitasi yang terarah.

Langkah 4: Eksploitasi

Eksploitasi bertujuan untuk mengubah kelemahan keamanan yang bersifat teoretis menjadi serangan nyata yang selaras dengan tujuan serangan. Dalam tahap ini, sering kali dilakukan penggabungan beberapa kelemahan keamanan untuk membuktikan keberhasilan eksploitasi. Sebagai contoh, menunjukkan bahwa bus kendaraan dapat diakses dari jarak jauh melalui ECU gerbang (*gateway ECU*) bisa menjadi langkah awal dalam rangkaian serangan yang menargetkan ECU yang tertanam jauh di dalam sistem, seperti modul kontrol rem. Langkah berikutnya mungkin melibatkan pemicuan kerentanan kerusakan memori (*memory corruption*) melalui perintah CAN atau perintah diagnostik yang dirancang khusus, yang kemudian menimpa data kontrol dan mengalihkan alur eksekusi.

Dalam pengujian komponen tertentu seperti MCU, atau sekumpulan modul perangkat lunak tertanam (*embedded software*), pembuktian adanya kelemahan keamanan yang dapat dieksploitasi bisa menjadi langkah akhir dari penugasan tersebut. Sebagai contoh, keberhasilan menunjukkan adanya kelemahan pada *firmware* HPSE dari vendor MCU yang memungkinkan kebocoran kunci kriptografis melalui analisis waktu *side-channel* dari sisi CPU utama dapat dianggap sebagai penyelesaian sukses dari penugasan *penetration testing* (pengujian penetrasi). Dalam skenario ini, diasumsikan bahwa tujuan keamanan vendor *firmware* HPSE adalah untuk melindungi kunci kriptografis agar tidak bocor akibat aplikasi yang telah disusupi yang berjalan pada CPU utama. Demikian pula, jika lingkup pengujian mencakup *flash bootloader* tertentu, maka keberhasilan dalam menunjukkan kemungkinan untuk masuk ke mode pemrograman *flash* serta menyelesaikan sesi pemrograman *flash* dengan memintas protokol autentikasi akan dianggap sebagai penyelesaian sukses dari penugasan *penetration testing*. Di sini, diasumsikan bahwa tujuan keamanan vendor *flash bootloader* adalah untuk mencegah akses tidak sah ke fungsi-fungsi pemrograman *flash*.

Dalam kasus pengujian ECU secara menyeluruh atau bahkan model kendaraan tertentu, penugasan *penetration testing* kemungkinan besar akan dijalankan dengan mengikuti langkah-langkah yang tercantum di bawah ini, sampai tujuan serangan tercapai, baik pada tingkat ECU maupun tingkat kendaraan.

Langkah 5: Eskalasi dan Ketekunan

Langkah selanjutnya setelah menunjukkan keberhasilan eksploitasi individu adalah mencoba meningkatkan kontrol dan ketekunan. Dengan ECU berbasis MCU, eskalasi berarti kemampuan untuk menjalankan kode tanpa batasan. Jika eksploitasi terjadi pada partisi OS yang tidak memiliki akses ke bus CAN, maka eskalasi mungkin melibatkan kemampuan untuk

mempengaruhi partisi OS lain yang memiliki konektivitas jaringan. Ini mungkin melibatkan penyalahgunaan MPU yang tidak tepat konfigurasi atau penyalahgunaan kerentanan pengkodean dalam sistem operasi itu sendiri. Namun, dalam banyak kasus, untuk ECU berbasis MCU, eskalasi dicapai melalui eksekusi *firmware* atau perangkat lunak aplikasi milik sendiri. Misalnya, setelah penyerang dapat menunjukkan kemampuan untuk memprogram ulang perangkat lunak, mereka cukup mengunduh perangkat lunak versi mereka sendiri yang memungkinkan akses langsung ke jaringan.

Eskalasi dan ketekunan tidak selalu sama. Jika tujuan serangan memerlukan kompromi dari entitas yang memiliki hak istimewa seperti HPSE, maka penyerang sekarang perlu menunjukkan bahwa aplikasi yang dikompromikan yang berjalan pada proses utama dapat mempengaruhi HPSE untuk mencapai tujuan serangan. Ini mungkin melibatkan memasukkan kode ke dalam HPSE atau mengubah aliran kontrolnya melalui antarmuka memori bersama. Ketekunan di sisi lain melibatkan demonstrasi keberhasilan pembaruan memori *flash*. Di sini penting untuk membedakan antara kerentanan yang memungkinkan pembaruan ECU yang dimiliki penyerang vs. kerentanan yang memungkinkan pembaruan ECU apa pun meskipun ECU hanya dapat diakses dari jarak jauh. Misalnya, jika ada kerentanan di ECU yang memungkinkan paket OTA diprogram tanpa tanda tangan yang valid, maka hal tersebut dianggap sebagai serangan terus-menerus. Seorang penyerang yang dapat mengatur server pembaruan OTA palsu mungkin dapat memasukkan gambar yang ditandatangani secara tidak valid ke ECU target tanpa kemampuan deteksi. Dengan kegigihan, dapat dikatakan bahwa ECU sekarang sepenuhnya berada di bawah kendali penyerang.

Langkah 6: Pergerakan Lateral di Seluruh Jaringan Kendaraan

Saat melakukan pengujian penetrasi (*penetration testing*) pada tingkat kendaraan, pergerakan lateral sering kali diperlukan untuk menunjukkan tercapainya tujuan serangan. Dalam tahap ini, ECU yang telah dikompromikan digunakan sebagai batu loncatan untuk menyasar ECU lainnya. Sebagai contoh, penyerang yang berhasil mengendalikan ECU yang terhubung ke jaringan dapat menyuntikkan lalu lintas data ke segmen jaringan bersama untuk memengaruhi ECU lain. Di sinilah teknik penemuan jaringan (*network discovery*) yang telah dibahas pada bab 6 menjadi sangat berguna. Pergerakan lateral juga dapat berarti berpindah dari kendaraan menuju lingkungan operasional, seperti EVSE (peralatan pasokan kendaraan listrik) atau bahkan infrastruktur *cloud* (*backend*).

Setelah satu ECU berhasil dikompromikan, penyerang sering kali beralih ke ECU lainnya. Karena jaringan di dalam kendaraan secara implisit memercayai lalu lintas internal, pergerakan lateral dapat dilakukan dengan cukup mudah. ECU yang telah dikompromikan dapat menyamar sebagai *node* lain, membuka sesi diagnostik, atau meneruskan pesan berbahaya ke berbagai segmen jaringan. Demikian pula, *gateway* yang dirancang untuk menegakkan segmentasi justru bisa menjadi celah kerentanan jika berhasil dikompromikan. Begitu *gateway* tersebut dilampaui, penyerang dapat menyebarkan serangan ke bus jaringan lain seperti jaringan *powertrain* (sistem penggerak), sasis, dan bodi kendaraan. Strategi ini pernah terlihat dalam kasus serangan jarak jauh terhadap kendaraan, di mana penyerang masuk melalui unit telematika, lalu mengirimkan perintah CAN internal ke ECU yang krusial bagi keselamatan.

Pergerakan lateral melipatgandakan dampak serangan sekaligus mempersulit deteksi, karena menyebabkan perilaku menyimpang yang terkoordinasi di berbagai modul.

Perspektif Serangan Menyeluruh (*End-To-End*)

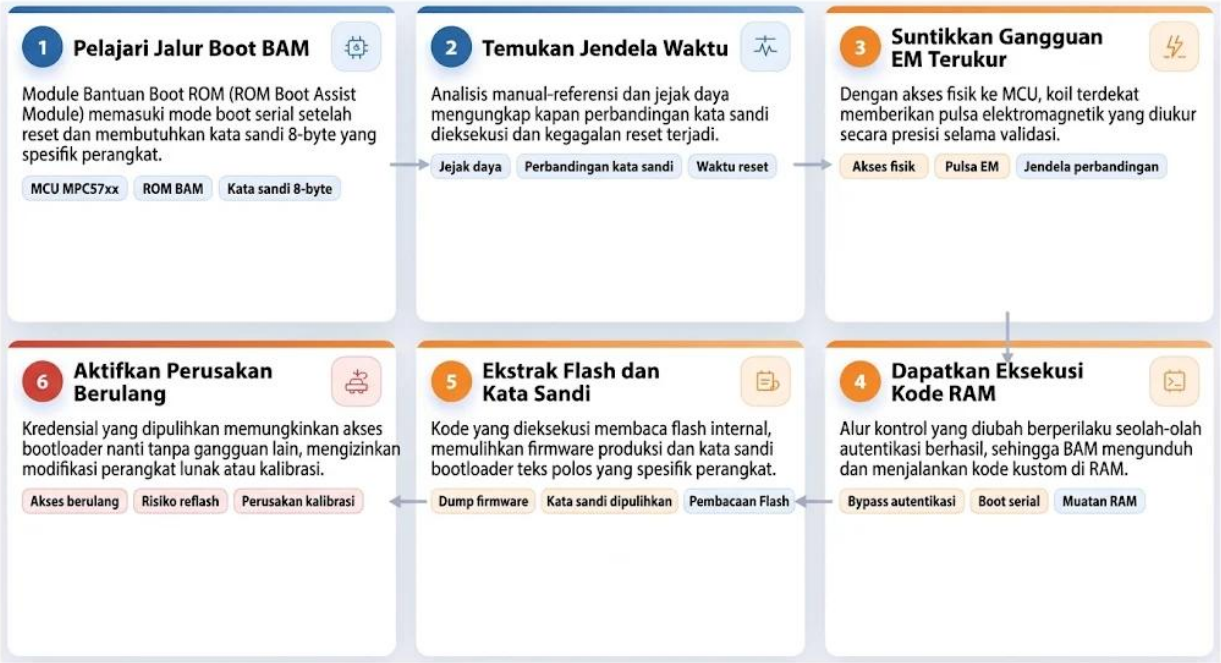
Rangkaian serangan yang lengkap dapat menggabungkan akses fisik, ekstraksi *firmware*, eksploitasi kelemahan logika, dan eksploitasi jaringan. Meskipun tidak semua serangan memerlukan tingkat kecanggihan seperti ini, pertahanan berlapis cenderung mendorong penyerang untuk menggunakan teknik yang saling terhubung (berantai). Dengan bergerak secara sistematis mulai dari tahap pengintaian (*reconnaissance*) hingga pergerakan lateral penyerang dapat mengupas satu demi satu lapisan model keamanan pada ECU berbasis MCU. Setelah menemukan kelemahan keamanan yang dapat dieksploitasi, penguji penetrasi dapat menyajikan temuan mereka beserta bukti eksploitasi yang menunjukkan langkah-langkah yang diambil, serta bagaimana langkah tersebut tetap selaras dengan asumsi keamanan awal yang telah disepakati bersama pemasok ECU atau OEM. Setelah memahami bagaimana rangkaian serangan ini disusun, kita dapat meninjau kasus-kasus nyata di mana urutan serangan serupa pernah dilakukan, sehingga kerangka kerja ini memiliki landasan berdasarkan eksploitasi yang benar-benar terjadi di lapangan.

7.5 STUDI KASUS KOMPROMI ECU BERBASIS MCU

Untuk memperkuat konsep-konsep yang telah dipaparkan sebelumnya, kita akan menelaah tiga studi kasus mengenai kompromi yang diungkapkan ke publik yang melibatkan ECU berbasis MCU. Setiap kasus menyoroti vektor serangan, tahapan serangan, dan dampak yang berbeda. Studi-studi ini diambil dari penelitian keamanan yang dipresentasikan dalam forum seperti Black Hat untuk menggambarkan perkembangan peretasan mobil selama satu dekade terakhir, dengan setiap kasus dipetakan ke dalam kerangka kerja MITRE ATT&CK agar lebih mudah dipahami secara terstruktur. Setiap studi kasus melibatkan lebih dari satu kelemahan keamanan. Anda diajak untuk mengidentifikasi kelemahan-kelemahan keamanan tersebut dengan memanfaatkan pengetahuan dari bagian sebelumnya mengenai kerentanan pada sistem berbasis MCU.

Studi Kasus 1: Memintas Kata Sandi *Bootloader* ECU Melalui Injeksi Kesalahan Elektromagnetik (*EM Fault Injection*)

Studi kasus ini berfokus pada ECU kendali *powertrain* (sistem penggerak) dari sebuah mobil Corvette keluaran tahun 2019, yang diklaim tahan terhadap upaya pencurian maupun modifikasi ilegal (*tuning*). Seorang penyerang yang ingin memodifikasi ECU *powertrain* demi meningkatkan performa kendaraan memerlukan cara untuk mengubah perangkat lunak serta data kalibrasi di dalamnya. Peneliti menunjukkan bahwa dengan menggunakan serangan injeksi kesalahan elektromagnetik yang diatur waktunya secara presisi, metode autentikasi *Boot Assist Module* (BAM) dapat dilumpuhkan, sehingga memungkinkan pelaku untuk memanipulasi perangkat lunak dan kalibrasi ECU tersebut. Gambar 7.9 memperlihatkan urutan langkah yang dilakukan untuk mencapai tujuan serangan.



Gambar 7.9 Urutan langkah untuk mengambil *firmware* ECU powertrain dan kata sandi *bootloader* serial

Penemuan dan Pengintaian

Target utama dalam studi kasus ini adalah BAM (*Boot Assist Module*), yaitu *bootloader* tahap pertama yang menyiapkan cip sebelum menyerahkan kendali kepada kode pengguna. MCU seri MPC57xx dari NXP/Freescale mengimplementasikan BAM di dalam ROM, yang akan langsung dijalankan setelah *reset*. Ketika pin perangkat keras tertentu diaktifkan, BAM akan memasuki mode *bootloader* serial yang menerima data melalui antarmuka serial, mengunduh kode ke dalam RAM, lalu mengeksekusi kode tersebut. ECU yang dievaluasi dalam studi ini mengandalkan mekanisme kata sandi *bootloader* bawaan MCU untuk mencegah modifikasi *firmware* yang tidak sah. *Bootloader* ini memerlukan kata sandi sepanjang 8 byte. Pada ECU produksi yang terkunci, kata sandi ini bersifat spesifik untuk setiap perangkat dan berbeda dari nilai standar yang didokumentasikan secara publik. Jika kata sandi yang dimasukkan salah, MCU akan melakukan *reset* setelah jeda pemrosesan singkat guna menggagalkan upaya *brute-force*.

Peneliti memulai dengan mempelajari manual referensi NXP yang tersedia untuk umum guna memahami urutan *booting*, peran BAM, serta alur logika perbandingan kata sandi. Mereka menggunakan analisis daya (*power analysis*) untuk menyimpulkan waktu alur kendali (*control-flow timing*) saat rutinitas perbandingan kata sandi dijalankan. Tahap persiapan ini sangat krusial, karena keberhasilan injeksi kesalahan (*fault injection*) menuntut akurasi waktu hingga ke tingkat sub-instruksi.

Akses Awal

Dengan menggunakan ChipSHOUTER sebagai perangkat injeksi kesalahan elektromagnetik (EMFI), peneliti menempatkan kumparan tembaga kecil di dekat kemasan MCU. Untuk melihat diagram perangkat EMFI tersebut, silakan merujuk ke. Berdasarkan

analisis daya dari tahap penemuan, mereka menyinkronkan pulsa elektromagnetik berenergi tinggi dengan momen saat *bootloader* membandingkan kata sandi yang dimasukkan dengan nilai yang tersimpan. Jika waktunya tepat, kesalahan tersebut akan menyebabkan penyimpangan alur kendali yang membuat kata sandi dianggap valid.

Akibatnya, *bootloader* melanjutkan proses seolah-olah kata sandi yang benar telah dimasukkan. Hal ini memberikan akses kepada peneliti untuk langsung mengeksekusi kode dalam konteks *bootloader*, sehingga mereka dapat memanfaatkan BAM untuk mengunduh kode ke dalam RAM yang kemudian dieksekusi. Dari perspektif taktis, langkah ini menggabungkan eksekusi kode dan peningkatan hak akses (*privilege escalation*) yang dicapai melalui injeksi kesalahan pada perangkat keras. Meskipun mekanisme autentikasi kata sandi tidak rusak secara kriptografis, mekanisme perlindungan tersebut berhasil dilewati dengan cara memanipulasi alur eksekusi prosesor.

Eksploitasi dan Persistensi

Setelah berhasil melewati pemeriksaan kata sandi *bootloader*, peneliti memperoleh kemampuan untuk mengunduh dan menjalankan kode kustom di dalam RAM melalui *bootloader* serial pada *Boot Assist Module*. Konteks eksekusi ini memberikan akses baca ke memori *flash* internal MCU, yang memungkinkan ekstraksi citra *firmware* produksi dari ECU. Meskipun demonstrasi tersebut tidak menyebarkan perangkat lunak berbahaya (*malware*) yang bersifat persisten ataupun mengubah konten memori *flash*, demonstrasi itu menunjukkan bahwa penulisan ulang (*reflashing*) atau modifikasi permanen pada *firmware* secara teknis dapat dilakukan setelah akses ke memori *flash* diperoleh. Karena kata sandi disimpan di dalam memori *flash* internal, maka saat *firmware* berhasil didapatkan, kata sandi yang spesifik untuk perangkat tersebut pun turut terungkap.

Dengan diperolehnya kata sandi ini, akses ke *bootloader* di kemudian hari tidak lagi memerlukan teknik *fault injection* (penyuntikan galat). Autentikasi yang sah menggunakan kredensial yang telah didapatkan tersebut sudah cukup untuk memasuki mode *boot* serial. Hal ini mengubah serangan yang sebelumnya mengharuskan *fault injection* fisik secara berulang menjadi metode akses yang dapat digunakan kembali berdasarkan kompromi kredensial, sehingga penyerang memiliki kemampuan penuh untuk memanipulasi citra perangkat lunak (*software image*).

Dampak

Kasus ini memberikan pelajaran yang lebih luas bagi sistem berbasis MCU: mekanisme keamanan yang hanya bergantung pada alur eksekusi yang tepat dapat dilumpuhkan jika penyerang mampu memicu galat pada perangkat keras (*hardware faults*). *Fault injection* dapat membatalkan asumsi dasar mengenai alur kendali (*control flow*), sehingga perlindungan pada tingkat yang lebih tinggi dapat dilewati melalui cara-cara fisik. Pelajaran lainnya berkaitan dengan cara penyimpanan kata sandi. Menggunakan kata sandi yang unik untuk setiap perangkat adalah praktik yang baik, namun menyimpannya dalam bentuk teks biasa (*plaintext*) di memori *flash* standar membuat proses ekstraksinya menjadi sangat mudah begitu penyerang berhasil menjalankan kode pada perangkat target. Akibat kelemahan-kelemahan ini, klaim bahwa ECU kebal terhadap manipulasi (*tampering*) dan penyetelan ulang (*tuning*)

terbukti tidak benar, yang memungkinkan pihak ketiga untuk meningkatkan performa sistem tersebut melampaui batasan yang ditetapkan oleh produsen peralatan asli (OEM).

Studi Kasus 2: Cara Mengeksploitasi *Bootloader* yang Tidak Aman (DEF CON)

Presentasi keamanan ini mengulas bagaimana *flash bootloader* pada kendaraan yang diakses melalui protokol UDS di atas jaringan CAN dapat disalahgunakan untuk memodifikasi *firmware* ECU, yakni ketika pihak penyerang mengambil alih atau memulai sesi pemrograman tanpa izin. Meskipun bersifat teoretis, pembahasan ini didasarkan pada survei terhadap berbagai implementasi *flash bootloader* dan mengisyaratkan adanya kelemahan keamanan nyata, tanpa menyebutkan produsen kendaraan (OEM) atau pemasok ECU tertentu yang terdampak. Penyerang yang mengeksploitasi kelemahan tersebut dapat melumpuhkan fungsi normal ECU dengan memaksanya masuk ke mode pemrograman, memasang *firmware* yang tidak sah, atau mencegat citra *firmware* (*firmware images*) saat sedang ditransmisikan.

Lebih parahnya lagi, *bootloader* itu sendiri dapat diganti dengan varian berbahaya yang terkadang disebut sebagai *bootloader trojan*. Jika hal ini terjadi, penyerang akan mendapatkan kendali permanen atas ECU, yang tetap bertahan meskipun perangkat dimatikan dan dinyalakan kembali (*power cycle*) atau dilakukan pembaruan perangkat lunak secara normal. Dalam kondisi tersebut, pemulihan sistem sering kali memerlukan intervensi fisik atau penggantian modul. Sekarang, mari kita cermati lebih dalam mengenai kelemahan-kelemahan yang ada serta pemetaannya terhadap fase-fase serangan.

Penemuan dan Pengintaian (Discovery and reconnaissance)

Peneliti menjelaskan bagaimana seseorang dapat mengamati lalu lintas diagnostik yang berkaitan dengan sesi pemrograman *flash* pada jaringan di dalam kendaraan. Dengan memantau lalu lintas CAN yang terkait dengan urutan transfer *firmware* dan aliran pesan diagnostik, dimungkinkan untuk memeriksa urutan operasi, perilaku waktu (*timing*), serta pertukaran autentikasi yang digunakan selama proses pemrograman. Karena *firmware* dikirimkan tanpa enkripsi, citra *firmware* dapat diperoleh dengan melakukan penyadapan (*sniffing*) pada bus CAN selama sesi pemrograman *flash* berlangsung, yang kemudian memungkinkan rekonstruksi *firmware* tersebut.

Peneliti memaparkan skenario di mana urutan pemrograman dapat dicegat atau dimanipulasi secara waktu nyata (*real-time*). Dengan menempatkan perantara di antara perangkat penguji diagnostik dan ECU (seperti menggunakan perangkat CANalyzer), pesan-pesan diagnostik dapat diputar ulang (*replay*), ditunda, atau diubah selama proses pemrograman *flash*. Kelemahan umum yang mungkin ditemukan selama fase penemuan meliputi:

- Tidak adanya autentikasi sebelum memasuki sesi pemrograman. Biasanya, vendor ECU diinstruksikan untuk melakukan pemeriksaan kelayakan (*plausibility check*) dalam aplikasi misalnya berdasarkan kecepatan kendaraan sebelum menerima permintaan untuk beralih ke mode pemrograman.
- Tidak adanya verifikasi tanda tangan (*signature verification*) yang wajib dilakukan terhadap citra *firmware* sebelum pembaruan diselesaikan. Dalam banyak kasus,

pemeriksaan integritas hanya mengandalkan checksum atau *cyclic redundancy check* (CRC) yang mudah dihitung.

- Algoritma *seed/key* yang lemah untuk membuka akses ke layanan diagnostik pemrograman *flash* yang bersifat kritis.

Akses Awal (Initial Access)

Akses awal diperoleh dengan menggunakan koneksi fisik melalui port OBD-II serta perangkat yang mendukung protokol UDS melalui CAN. Akses ini memungkinkan komunikasi langsung dengan ECU untuk memulai sesi pemrograman *flash*. Studi tersebut menekankan bahwa akses semacam ini sering kali dianggap aman (tidak berbahaya), yang justru menjadikannya titik masuk yang praktis bagi penyerang.

Peneliti juga menganalisis arsitektur yang mendukung pembaruan melalui udara (*over-the-air* atau OTA). Pada ECU berbasis mikrokontroler (MCU), komponen yang paling tepat untuk berinteraksi dengan klien OTA adalah *flash bootloader*. Biasanya, klien OTA yang berjalan di ruang aplikasi menangani komunikasi dengan komponen gerbang (*gateway*) kendaraan seperti Unit Telematika sementara *flash bootloader* menjalankan proses pengunggahan perangkat lunak serta permintaan pengunduhan. Dalam konteks ini, peneliti menggambarkan OTA secara fungsional setara dengan menempatkan perangkat penguji diagnostik pada jaringan jarak jauh. Hal ini menunjukkan bahwa kelemahan dalam protokol pemrograman *flash* dan desain *bootloader* menjadi terekspos ke permukaan serangan yang lebih luas melampaui batas fisik ECU akibat adanya klien OTA yang dapat digunakan untuk membangun pijakan awal.

Eksploitasi

Dengan memulai sesi pemrograman pada ECU, penyerang dapat dengan mudah menangguhkan aplikasi yang sedang berjalan normal. Tentu saja, masuk ke sesi pemrograman saat kendaraan sedang bergerak dianggap sebagai pelanggaran keselamatan, karena ECU akan kehilangan fungsi normalnya; kondisi ini pada dasarnya merupakan serangan penolakan layanan (*denial-of-service*).

Selain penolakan layanan, pembahasan tersebut juga menjelaskan bahwa sebagian perintah diagnostik dapat diputar ulang (*replay*) karena perilaku *seed-key* yang dapat diprediksi dalam implementasi layanan Rp 270.000 (*SecurityAccessControl*). Peneliti juga menyoroti bagaimana lalu lintas data dapat dicegat dan dimodifikasi selama transfer *firmware*, yang memungkinkan penyisipan atau pengubahan muatan (*payload*) pembaruan. Jika tanda tangan digital tidak digunakan, penyerang hanya perlu memperbarui nilai pemeriksaan integritas *firmware* agar *firmware* yang telah dimodifikasi dapat diterima oleh *bootloader* setelah proses pengunduhan selesai.

Selain pada ECU itu sendiri, peneliti menyoroti kelemahan dalam perangkat pendukung pemrograman (*flashing toolchain*). Perangkat-perangkat ini sering kali menyimpan citra (*image*) *firmware*, dan dalam beberapa kasus, memuat data *seed-key* atau algoritma eksklusif. Artinya, rekayasa balik (*reverse engineering*) terhadap perangkat pemrograman tersebut dapat menimbulkan dampak serius, karena kredensial yang bocor serta citra *firmware* yang diekstraksi dapat disalahgunakan dalam skala yang lebih luas.

Persistensi

Jelas bahwa kemampuan untuk mencegat sesi pemrograman *flash* dapat memungkinkan seseorang untuk memasukkan *firmware* buatannya sendiri. Secara teoretis, pihak OEM masih dapat mengeluarkan pembaruan perangkat lunak untuk mengembalikan *firmware* ke versi yang sah. Namun, dampak paling parah yang dibahas dalam presentasi tersebut adalah kemungkinan untuk mengganti *bootloader* itu sendiri. Jika mekanisme pembaruan *bootloader* tidak mengharuskan adanya tanda tangan digital yang sah pada citra *bootloader*, maka urutan pemrograman *flash* yang sama dapat digunakan untuk mengganti *bootloader* dengan versi berbahaya. Dengan menimpa komponen *boot* menggunakan kode yang dikendalikan oleh penyerang, persistensi permanen pun tercapai. Bentuk kompromi ini tetap bertahan meskipun *firmware* utama diprogram ulang, serta dapat secara selektif mengizinkan atau memblokir pembaruan di masa mendatang.

Begitu persistensi terbentuk pada tingkat komponen *boot*, deteksi menjadi sulit dilakukan. Prosedur diagnostik standar untuk membaca versi perangkat lunak mungkin menampilkan nilai yang telah dimanipulasi (*spoofed*), sementara *bootloader* berbahaya tersebut diam-diam menjalankan keinginan penyerang. Dalam banyak kasus, pemulihan memerlukan prosedur perbaikan khusus atau bahkan penggantian ECU secara menyeluruh, yang berdampak signifikan terhadap biaya, keselamatan, dan cakupan penarikan kembali (*recall*) produk.

Dampak

Studi teoretis ini mengilustrasikan mengapa jalur pembaruan *firmware* pada ECU berbasis MCU menuntut tingkat ketelitian yang sama dengan fungsi operasional utama. *Bootloader* bukanlah utilitas pasif, melainkan komponen yang kuat dan memiliki hak akses istimewa; jika tidak diamankan dengan benar, komponen ini memberikan akses langsung bagi penyerang untuk menguasai sistem secara persisten. Selain itu, antarmuka ke klien OTA (*Over-the-Air*) membuat *flash bootloader* sangat rentan terhadap serangan jarak jauh.

Perlu dicatat bahwa dalam beberapa tahun terakhir, produsen kendaraan (OEM) telah meningkatkan keamanan spesifikasi *flash bootloader* secara signifikan. Peningkatan ini mencakup kewajiban penggunaan tanda tangan kriptografis pada citra (*image*) pembaruan perangkat lunak, serta penerapan mekanisme *challenge-response* yang aman secara kriptografis untuk melindungi layanan diagnostik (melalui *service* Rp 270.000). Meski demikian, kemampuan untuk mendeteksi eksploitasi semacam ini harus menjadi pengingat bahwa kelemahan dalam penerapan kontrol keamanan di tingkat *bootloader* tetap menjadi masalah kritis yang harus ditangani dengan serius.

Studi Kasus 3: Mobil Saya, Kunci Saya: Memperoleh Kunci Penandatanganan SecOC Pada Bus CAN

Studi kasus ini mendokumentasikan bagaimana para peneliti berhasil mendapatkan kunci kriptografis SecOC dari ECU sistem kemudi (*power steering*) pada sebuah platform kendaraan yang mulai menerapkan autentikasi kriptografis untuk pesan CAN terkait sistem kemudi sejak sekitar tahun model 2020. Untuk mencapai hal ini, para peneliti terlebih dahulu mengekstrak *firmware* dengan memintas perlindungan *debug* melalui metode *fault injection*

(injeksi kesalahan). Setelah mendapatkan *firmware* tersebut, mereka melakukan rekayasa balik (*reverse engineering*) terhadap aplikasi maupun *bootloader*-nya. Analisis ini mengungkap bahwa *bootloader* mendukung pengunduhan *payload* yang telah ditandatangani dan dienkripsi ke dalam RAM, kemudian mengeksekusinya melalui pemanggilan rutin (*routine call*). Dengan menyalahgunakan mekanisme ini, para peneliti mampu menjalankan kode buatan penyerang yang dapat membaca data kunci sensitif dari RAM. Setelah kunci diperoleh, para peneliti dapat membuat pesan sistem kemudi yang sah dan terautentikasi yang kemudian diterima oleh ECU target serta mencatat bahwa eksekusi kode tersebut berpotensi dialihfungsikan menjadi kemampuan layaknya *signing-oracle* (sistem yang dapat menghasilkan tanda tangan kriptografis secara otomatis). Gambar 7.10 memperlihatkan urutan langkah yang dilakukan untuk mencapai tujuan serangan.



Gambar 7.10: Langkah-langkah eksfiltrasi kunci kriptografis SecOC

Penemuan dan Pengintaian

Pada tahap awal penemuan, para peneliti mengamati bahwa kendaraan model tahun lebih baru yang menjadi target mereka telah menerapkan perlindungan SecOC pada pesan-pesan CAN tertentu, dengan sebagian dari setiap bingkai (*frame*) CAN dialokasikan untuk *message authentication code* (MAC) kriptografis yang dipangkas. Berdasarkan pengamatan tersebut, mereka memilih ECU sistem kemudi daya (*power steering*) sebagai target utama, dengan pertimbangan bahwa komponen ini memvalidasi pesan-pesan *steer-by-wire* dan kemungkinan besar diimplementasikan pada MCU yang memiliki fitur keamanan (*hardening*) lebih terbatas.

Untuk memetakan komponen perangkat keras, para peneliti memperoleh sebuah ECU mandiri, membongkarnya, dan mengidentifikasi MCU yang digunakan sebagai RH850/P1M-E (R7F701381). Melalui penelusuran dokumen publik, mereka menemukan antarmuka pemrograman serial berbasis UART yang dapat digunakan untuk mengekstraksi *firmware*. Antarmuka tersebut dikunci menggunakan kode ID perlindungan, namun penelitian

sebelumnya telah menunjukkan cara untuk melewati penguncian tersebut melalui serangan *glit*ch (gangguan teknis yang disengaja).

Oleh karena itu, mereka menerapkan teknik *fault injection* (injeksi kesalahan) pada domain regulator tegangan inti internal MCU. Proses ini mencakup pelepasan kapasitor bypass dan pemberian gangguan (*glitch*) pada kedua jalur tegangan inti menggunakan transistor N-FET. Dengan memanfaatkan perangkat murah berbasis Raspberry Pi Pico dan perangkat keras eksternal, mereka menyuntikkan gangguan dengan pengaturan waktu yang sangat presisi selama pertukaran protokol debug. Tindakan ini memicu MCU untuk memberikan respons positif yang membuka kunci antarmuka pemrograman serial, sehingga memungkinkan mereka untuk menyalin (*dump*) *firmware* tersebut.

Pada penemuan tahap kedua, para peneliti melakukan rekayasa balik terhadap *firmware* yang dibuang menggunakan alur kerja berbasis Ghidra. Mereka menganalisis aplikasi dan *bootloader* untuk memahami struktur kode, penangan diagnostik, logika penguraian CAN, serta SecOC implementasi, serta perilaku penanganan kunci. Mereka juga memeriksa konten data*flash* untuk mencari rahasia yang tertanam di dalam kode (*hardcoded secrets*).

Melalui analisis ini, mereka mengonfirmasi bahwa kunci SecOC dimuat ke dalam RAM, alih-alih mengandalkan HPSE untuk perlindungan kunci secara berkelanjutan. Mereka juga memverifikasi bahwa kunci tersebut tidak dapat digunakan kembali pada kendaraan lain dan setidaknya terikat pada VIN (*Vehicle Identification Number*) tertentu. Meskipun hal ini membatasi kemungkinan serangan untuk diterapkan pada seluruh armada kendaraan, serangan tersebut tetap dapat memberikan kendali penuh atas ECU yang menjadi target. Dengan demikian, tujuan serangan pun menjadi jelas: menemukan cara untuk menyalin (*dump*) isi RAM guna mengekstrak kunci-kunci SecOC tersebut.

Pada tahap ketiga penemuan, para peneliti mencari penangan (*handler*) XCP dan CCP, serta layanan UDS yang mungkin berguna dalam ekstraksi konten RAM. Mereka tidak menemukan XCP maupun CCP dukungan, sehingga mereka mengalihkan perhatian ke layanan UDS pada *flash bootloader*. Temuan menarik utamanya di sana adalah dukungan untuk pengunduhan dan eksekusi rutin *flash* di dalam RAM melalui layanan Rp 310.000 (*RoutineControl*). Jadi, gagasan di baliknya adalah membuat blob RAM untuk injeksi kode, yang kemudian diunduh ke dalam RAM melalui layanan UDS guna melancarkan eksploitasi lebih lanjut.

Namun, terdapat beberapa hambatan yang harus mereka atasi terlebih dahulu. Blob ini dilindungi oleh CRC32 untuk menjamin integritas, AES-CMAC untuk memastikan keaslian, serta dienkripsi menggunakan AES-CBC dengan kunci yang diturunkan (*derived key*). Menghitung CRC32 dari suatu blob baru adalah hal yang sepele, namun menghitung AES-CMAC yang valid serta mengenkripsi blob tersebut memerlukan pengetahuan mengenai kunci-kunci kriptografis. Untungnya, melalui rekayasa balik (*reverse engineering*) *firmware*, para peneliti berhasil menemukan bahwa kunci yang diperlukan untuk menghitung CMAC dari blob RAM dan melakukan enkripsi CBC diperoleh dari nilai-nilai yang ditanamkan secara permanen (*hardcoded*) di dalam *firmware* yang berhasil dipulihkan tersebut.

Kini mereka telah memiliki semua komponen yang diperlukan untuk menyusun muatan berbahaya. Mereka menulis muatan berbasis bahasa C yang menyalin bagian-bagian tertentu dari RAM lalu mentransfernya ke pengontrol CAN, sehingga isi RAM yang diekstraksi tersebut dapat diamati melalui bus CAN. Kini, mereka telah memiliki semua komponen yang diperlukan untuk menyusun payload berbahaya. Mereka menulis payload dalam bahasa C yang menyalin wilayah RAM tertentu dan mentransfernya ke pengontrol CAN, sehingga isi RAM yang diekstraksi dapat diamati melalui bus CAN. Sebuah perangkat lunak berbasis Python kemudian menyusun blob akhir dengan menghitung CMAC, menangani enkripsi, serta menyertakan nilai-nilai hardcoded yang diperlukan. Sebuah perangkat Python menyusun blob akhir dengan menghitung CMAC, menangani enkripsi, serta menyertakan nilai-nilai yang telah ditetapkan sebelumnya (*hardcoded*).

Akses Awal

Akses awal diperoleh dengan memanfaatkan fungsionalitas diagnostik UDS standar untuk mengalihkan ECU ke mode *bootloader* dan melakukan autentikasi melalui *SecurityAccess* (layanan Rp 270.000) menggunakan kunci AES simetris yang berhasil dipulihkan (melalui rekayasa balik *firmware*). Dengan autentikasi yang sah, peneliti memperoleh akses sebagai klien diagnostik tepercaya dan diizinkan untuk menjalankan prosedur kontrol serta prosedur pengunduhan rutin yang biasanya ditujukan untuk pembaruan perangkat lunak (*reflashing*) dan pemeliharaan ECU. Hal ini memungkinkan blob RAM terenkripsi untuk ditransfer dan dieksekusi tanpa memicu kondisi kesalahan.

Eksploitasi

Setelah berjalan di RAM, eksploitasi menjadi sangat mudah. Payload yang disisipkan secara langsung mengakses memori internal dan mengekstraksi aset kriptografi, termasuk kunci MAC SecOC, dengan menyalin isinya ke dalam register CAN untuk diamati melalui bus. Payload kemudian mereset ECU agar operasi normal dapat dilanjutkan, menandakan berakhirnya fase eksploitasi.

Persistensi

Para peneliti menekankan bahwa tujuan yang mereka demonstrasikan adalah ekstraksi kunci, bukan pemasangan malware yang persisten. Namun, mereka secara eksplisit membahas implikasi pasca-eksploitasi. Setelah kunci SecOC yang relevan untuk cakupan ECU atau VIN tertentu berhasil diperoleh, penyerang mendapatkan kemampuan jangka panjang yang berakar pada rahasia kriptografi, bukan pada implan. Dengan kunci ini, pesan terautentikasi yang valid dapat dibuat berulang kali, dan dalam beberapa konfigurasi, ECU berpotensi dapat diubah kuncinya menggunakan Master_ECU_KEY yang telah dipulihkan. Para peneliti juga mencatat bahwa kemampuan eksekusi *bootloader* yang sama dapat memungkinkan persistensi dengan upaya tambahan, misalnya dengan membajak sesi pemrograman *flash* alih-alih membatasi eksploitasi hanya pada pengambilan isi RAM.

Dampak

Dampak utama yang ditunjukkan oleh studi kasus ini adalah pengendalian jaringan terhadap fungsi kemudi yang sangat penting bagi keselamatan. Dengan memulihkan material kunci SecOC, mereka mampu menghasilkan pesan kontrol kemudi yang valid dan dilindungi

secara kriptografis, yang diterima oleh ECU target dalam cakupan kuncinya. Sepanjang analisis, para peneliti menyoroti beberapa kelemahan utama.

Mereka menunjukkan bahwa akses ke antarmuka pemrograman serial pada MCU target dapat dilewati melalui teknik glitching. Mereka mengonfirmasi bahwa material kunci SecOC yang kritis tersimpan di RAM dan bahwa, pada setidaknya dua model kendaraan, sebuah kunci AES yang disebut sebagai kunci SE disimpan dalam bentuk plaintext di *flash*. Pengetahuan tentang kunci tersebut memungkinkan mereka membuat blob RAM terautentikasi yang kemudian diunduh oleh *bootloader* ke RAM dan dieksekusi tanpa masalah, dengan asumsi bahwa blob tersebut merupakan rutinitas *flash* yang diharapkan dan biasanya diunduh ke RAM selama sesi pemrograman *flash*. Terakhir, mereka mengamati adanya celah dalam perilaku anti-replay SecOC, seperti penanganan penghitung perjalanan yang tidak mencegah replay sebagaimana mestinya.

Setelah kunci diekstrak, perlindungan kriptografis tidak lagi dapat mencegah injeksi pesan untuk ECU tersebut. Dengan diperolehnya rutinitas *flash*, kunci AES CMAC, kunci enkripsi rutinitas *flash*, dan kunci Master ECU, dimungkinkan untuk mengunduh kode arbitrer ke RAM yang kemudian dieksekusi melalui *bootloader flash* dan berpotensi melakukan pembaruan kunci SecOC.

Kasus ini semakin menunjukkan bahwa meskipun komunikasi aman di dalam kendaraan telah diaktifkan pada kendaraan tersebut, mekanisme ini tetap dapat ditembus akibat kelalaian dalam praktik dasar manajemen kunci. Dan, ketiga studi kasus ini memberikan gambaran mengenai seperti apa serangan di dunia nyata terhadap ECU berbasis MCU. Jika Anda adalah seorang insinyur keamanan ofensif, kini Anda memiliki gambaran yang cukup baik mengenai kelemahan umum yang akan Anda temui dan teknik serangan yang dapat Anda terapkan terhadap ECU berbasis MCU. Bagi para insinyur keamanan defensif, kelemahan tersebut merupakan sasaran paling mudah untuk ditangani. Selain itu, terdapat beberapa praktik terbaik keamanan umum yang dapat membantu mengurangi keberhasilan serangan tersebut, seperti yang akan kita lihat selanjutnya.

7.6 STRATEGI PENGUATAN KEAMANAN ECU BERBASIS MIKROKONTROLER

Melindungi ECU berbasis mikrokontroler dari berbagai jenis serangan yang dijelaskan dalam bab ini merupakan tantangan sekaligus sesuatu yang tidak dapat dihindari. Produsen mobil dan pemasok harus berasumsi bahwa pihak penyerang yang memiliki motivasi kuat akan menargetkan MCU secara langsung, termasuk melalui akses fisik, sehingga mereka harus menerapkan strategi keamanan berlapis yang sering kami sebut sebagai strategi defense in depth. Tujuannya bukan untuk membuat serangan menjadi mustahil, melainkan untuk meningkatkan biaya yang harus dikeluarkan secara signifikan. Langkah-langkah mitigasi berikut merupakan praktik terbaik yang telah terbukti untuk memperkuat keamanan ECU berbasis MCU.

Mengunci Antarmuka Debug dan Pengembangan

Semua ECU produksi harus memiliki antarmuka debug dan pengembangan seperti JTAG dan SWD yang dinonaktifkan secara permanen atau dikunci dengan aman. Hal ini

biasanya memerlukan pemrograman memori *one-time-programmable* atau bit konfigurasi yang mencegah akses ke mode debug dan pengujian setelah ECU memasuki mode operasional. Jika akses debug harus tetap tersedia untuk keperluan servis yang sah, akses tersebut harus dilindungi menggunakan mekanisme *challenge-response* yang aman secara kriptografis, bukan kata sandi tetap.

Setiap mekanisme pintu belakang residual, seperti kombinasi strap GPIO yang memaksa mode boot atau test, harus dihilangkan atau dilindungi dengan tindakan anti-perusakan yang kuat. Panduan NHTSA secara eksplisit merekomendasikan penghapusan akses tingkat pengembang dalam sistem produksi kecuali benar-benar diperlukan, dan bahkan jika diperlukan, hanya di bawah kendali ketat. Tindakan pencegahan fisik seperti melepas bantalan test, memasang pin debug, atau mengaplikasikan epoksi dapat semakin meningkatkan kesulitan penyalahgunaan.

Kontrol ini sangat penting karena serangan di dunia nyata telah berulang kali menunjukkan bahwa akses fisik ke ECU dapat menghasilkan ekstraksi *firmware* atau kontrol eksekusi, baik melalui akses debug langsung atau melalui injeksi kesalahan dan teknik side-channel. Akibatnya, pihak yang bertahan harus merancang dengan asumsi bahwa ECU pada akhirnya akan jatuh ke tangan penyerang dan mengandalkan perlindungan yang kuat daripada keamanan melalui kerahasiaan.

Menerapkan Secure Boot dan Autentikasi *Firmware*

Secure boot merupakan pertahanan mendasar untuk menjaga integritas perangkat lunak ECU. Pada setiap *proses boot*, MCU harus memverifikasi keaslian dan integritas image *firmware* serta data kalibrasinya menggunakan tanda tangan digital yang aman secara kriptografis dengan kekuatan kunci yang sesuai untuk masa pakai kendaraan yang diharapkan. Hal ini dapat diimplementasikan dalam ROM yang tidak dapat diubah atau *bootloader* tahap pertama yang *flash* kodenya telah dikunci, atau melalui HPSE yang berfungsi sebagai akar kepercayaan berbasis perangkat keras. Jika verifikasi gagal, ECU harus menolak untuk mengalihkan kendali ke aplikasi pengguna dan sebagai gantinya memasuki kondisi fail-safe.

Seperti yang telah kita lihat dalam bab ini, serangan injeksi kesalahan dapat mengakibatkan dilewatinya pemeriksaan boot dalam kondisi tertentu. Oleh karena itu, desain *secure boot* harus diperkuat untuk menghadapi serangan umum seperti *glitching*, masalah TOCTU, manipulasi kunci publik root, dan serangan *downgrade*. Hal ini dapat dilakukan dengan menerapkan praktik pengodean defensif anti-glitch, analisis kode statis, dan pengujian keamanan yang tangguh, menyimpan kunci publik root dalam penyimpanan aman yang dikendalikan oleh HPSE, serta melindungi opsi kalibrasi keamanan dengan tanda tangan digital yang diberlakukan oleh *root of trust* atau entitas dalam rantai kepercayaan.

Menegakkan Kontrol Akses dan Perlindungan Memori

MCU menyediakan mekanisme perlindungan memori yang sering kali kurang dimanfaatkan untuk tujuan keamanan. Perancang harus mengonfigurasi MPU atau MMU (jika tersedia) untuk membagi komponen perangkat lunak ke dalam kompartemen berdasarkan tingkat kepercayaan dan kekritisan keamanan, bukan sekadar berdasarkan aspek keselamatan (*safety*). Sebagai contoh, tumpukan komunikasi CAN harus diisolasi dari algoritma kontrol agar

buffer overflow saat penguraian pesan tidak mengakibatkan injeksi kode ke dalam partisi sistem operasi yang krusial bagi keselamatan.

Jika didukung oleh RTOS, pemisahan hak akses (*privilege separation*) harus diterapkan sehingga layanan yang terpapar ke pihak luar seperti diagnostik dan penanganan input jaringan dapat berjalan dalam mode dengan hak akses yang lebih rendah. Langkah ini membatasi dampak eksploitasi dan dapat mengubah situasi yang seharusnya menjadi kompromi sistem secara menyeluruh menjadi sekadar gangguan layanan (*denial-of-service*) yang terbatas pada area tertentu.

Langkah-langkah penguatan (*hardening*) perangkat lunak tambahan, seperti stack canaries dan deteksi buffer overflow berbasis kompilator (contohnya StackGuard pada GCC), harus diaktifkan jika memungkinkan. Meskipun langkah-langkah ini menimbulkan sedikit beban tambahan (*overhead*), hal tersebut secara signifikan meningkatkan ketahanan terhadap kerentanan keamanan memori yang umum terjadi.

Terkait *firmware* HPSE, pengembangannya harus memenuhi target jaminan (*assurance target*) yang lebih tinggi dibandingkan perangkat lunak biasa, dengan perhatian khusus pada penguatan protokol komunikasi memori bersama (*shared memory*) dengan CPU utama serta konfigurasi pengontrol DMA. *Firmware* HPSE harus mengaktifkan DEP, perlindungan stack, dan pemisahan yang ketat dari inti aplikasi utama. Idealnya, materi kunci (*key material*) tidak boleh terlihat langsung oleh CPU HPSE itu sendiri; operasi kriptografi harus ditangani secara eksklusif oleh mesin perangkat keras yang hanya menampilkan handle kunci yang bersifat *opaque* (tidak dapat diintip isinya).

Penguatan Antarmuka Diagnostik dan Pembaruan

Seperti yang telah dibahas, protokol diagnostik seperti UDS memiliki kapabilitas yang luas berdasarkan desainnya, sehingga harus dianggap sebagai permukaan serangan (*attack surface*) berisiko tinggi. Semua layanan diagnostik sensitif, termasuk pengaturan ulang ECU (*reset*), akses ke mode pemrograman, dan akses memori, harus memerlukan autentikasi yang kuat. Jika menggunakan *SecurityAccess* pada UDS, algoritma *challenge-response* yang mendasarinya harus tahan terhadap serangan brute force dan rekayasa balik (*reverse engineering*), serta kunci harus dibuat unik untuk setiap kendaraan jika memungkinkan. Jika layanan UDS 0x29 digunakan, validasi sertifikat harus mencakup verifikasi rantai penuh (*full chain verification*), termasuk validasi penerbit sertifikat dan status pencabutan (*revocation status*) jika didukung.

Selain memvalidasi keaslian, sistem harus mengaitkan identitas yang telah diautentikasi dengan peran (*role*) yang didefinisikan secara jelas. Setiap peran hanya boleh diizinkan untuk menjalankan sekumpulan layanan diagnostik yang terbatas dan ditentukan secara eksplisit. Ketahanan terhadap serangan *replay* (pengulangan pesan) dan *downgrade* (penurunan versi) juga merupakan persyaratan krusial. Proses pertukaran autentikasi harus menyertakan *nonce* (angka acak sekali pakai) untuk mencegah penggunaan ulang pesan yang sempat ditangkap sebelumnya. Fungsionalitas diagnostik juga harus bersifat sadar konteks (*context-aware*). Banyak ECU telah membatasi operasi pemrograman berdasarkan status kendaraan seperti posisi kunci kontak atau pergerakan kendaraan dan pembatasan ini harus

diterapkan secara konsisten. Mekanisme pembatasan laju (*rate limiting*) dan penguncian (*lockout*) juga dapat diterapkan untuk mencegah serangan *brute force* terhadap layanan autentikasi.

Lebih lanjut, mekanisme pembaruan *firmware* baik yang dilakukan melalui dealer maupun secara *over-the-air* (OTA) harus menjamin keaslian sekaligus kerahasiaan citra (*image*) pembaruan tersebut. Penggunaan modul kriptografi yang Tangguh MCU otomotif modern kini semakin banyak menyertakan HPSE (*Hardware-Protected Security Element*) yang dirancang untuk memisahkan operasi kriptografi dan penyimpanan kunci dari fungsi aplikasi biasa. Modul-modul ini sebaiknya digunakan untuk melakukan verifikasi tanda tangan, autentikasi, dan enkripsi tanpa mengekspos kunci dalam bentuk teks polos (*plaintext*) kepada prosesor aplikasi utama. Bahkan jika perangkat lunak ECU utama berhasil disusupi, HPSE dapat mencegah ekstraksi dan penyalahgunaan kunci dengan membatasi operasi hanya melalui API kriptografi yang telah ditetapkan secara ketat.

Firmware HPSE harus menyertakan implementasi berwaktu konstan (*constant-time implementations*) guna mencegah kebocoran melalui saluran samping (*side channel leakage*). Selain itu, OEM harus merencanakan rotasi kunci secara berkala dan menghindari penggunaan ulang kunci di seluruh perangkat dalam jumlah besar. Risiko kebocoran kunci kriptografis akibat injeksi kesalahan (*fault injection*) juga harus diantisipasi dengan meminimalkan alur kendali yang bergantung pada data rahasia, serta memanfaatkan akselerator perangkat keras yang dirancang untuk menangkal analisis saluran samping. Faktanya, menyimpan kunci dalam bentuk teks terang (*plaintext*) untuk jangka waktu lama di dalam memori *flash* atau RAM biasa sudah tidak dapat dibenarkan lagi.

Mengintegrasikan Langkah Penanggulangan Injeksi Kesalahan (*Fault Injection*)

Sebagaimana telah kita lihat dalam bab ini, serangan injeksi kesalahan bukan lagi sekadar teori. Perancang ECU harus memperhitungkan teknik manipulasi fisik seperti manipulasi tegangan, sinyal jam (*clock*), elektromagnetik, dan metode lainnya. Pemasok cip tidak boleh membebankan penanganan seluruh serangan tersebut hanya kepada perangkat lunak. Hal ini terutama berlaku untuk antarmuka perangkat keras seperti pemrograman serial bawaan (*built-in serial programming*) dan JTAG, di mana perangkat lunak memiliki kendali yang sangat terbatas untuk melawan injeksi kesalahan.

Banyak mikrokontroler (MCU) otomotif menyediakan pemantau bawaan untuk mendeteksi anomali tegangan, frekuensi, dan suhu. Perangkat lunak dapat memanfaatkan fitur-fitur ini, khususnya pada tahap awal proses booting. Pada tahap tersebut, teknik pemrograman defensif harus diterapkan untuk mengurangi tingkat keberhasilan serangan *glitch* (gangguan sesaat), misalnya dengan menggunakan pemeriksaan kondisional bertingkat serta penggunaan konstanta dengan jarak Hamming (*Hamming distance*) yang besar. Langkah ini memperkecil kemungkinan bahwa satu kesalahan tunggal dapat mengubah hasil otorisasi. *Bootloader*, khususnya, harus dirancang sedemikian rupa sehingga untuk melewati pemeriksaan keamanan, diperlukan beberapa kesalahan yang presisi, bukan sekadar satu glitch tunggal.

Perlindungan tambahan mencakup sensor pendeteksi gangguan (*tamper-detection sensors*), lapisan pelindung (*coatings*), atau jaring pelindung (*meshes*) yang dapat mendeteksi serangan invasif, seperti injeksi menggunakan laser atau gelombang elektromagnetik (EM). Jika gangguan terdeteksi, HPSE (*Hardware-based Secure Element*) dapat diinstruksikan untuk menghapus kunci sensitif atau beralih ke status terkunci (*lockdown*). Meskipun respons semacam ini membawa risiko operasional, tindakan tersebut dapat dibenarkan untuk komponen yang memiliki dampak besar, seperti *gateway* atau ECU yang berfungsi mencegah pencurian kendaraan.

7.7 KERENTANAN DAN ANCAMAN KEAMANAN ECU BERBASIS MIKROKONTROLER

ECU berbasis mikrokontroler dulunya dianggap terlalu terbatas dan terlalu tertanam (*embedded*) sehingga tidak menarik perhatian serius dari penyerang. Seperti yang ditunjukkan sepanjang bab ini, asumsi tersebut tidak lagi berlaku. ECU berbasis mikrokontroler kini menjadi target yang rentan sekaligus menarik; serangan yang berhasil bahkan telah terjadi dalam praktik nyata.

Serangan-serangan ini mencakup spektrum yang luas, mulai dari eksploitasi perangkat lunak melalui jaringan di dalam kendaraan (seperti CAN) hingga teknik injeksi kesalahan yang canggih. Hasilnya adalah lanskap ancaman yang melibatkan penyerang lokal dengan akses fisik, penyerang dalam jarak dekat yang memanfaatkan OBD-II atau akses ke jaringan internal, hingga penyerang jarak jauh yang merangkai berbagai kerentanan untuk menjangkau ECU yang tertanam jauh di dalam sistem kendaraan.

Dampak potensial dari kompromi pada ECU sangatlah besar. Karena pengontrol ini secara langsung mengatur fungsi-fungsi kendaraan yang krusial bagi keselamatan, serangan yang berhasil dapat menyebabkan hilangnya kendali, penurunan kinerja atau penonaktifan mekanisme keselamatan, hingga memicu perilaku yang tidak aman selama kendaraan beroperasi. Bahkan ketika keselamatan tidak langsung terpengaruh, penyerang dapat mengekstrak data sensitif seperti materi kunci immobilizer, memintas batasan emisi atau pengaturan performa (*tuning*), atau dengan sengaja merusak unit kontrol elektronik (ECU) hingga tidak dapat berfungsi sama sekali (*bricking*), yang pada akhirnya menimbulkan kerugian finansial dan biaya operasional bagi produsen peralatan asli (OEM). Fakta-fakta ini menegaskan pentingnya pendekatan keamanan berlapis. Pertahanan di tingkat perangkat keras, perangkat lunak, dan arsitektur harus saling terintegrasi agar kegagalan dapat dibatasi dan tidak merembet menjadi masalah yang lebih luas.

Bagi para insinyur keamanan ofensif, ranah mikrokontroler (MCU) menawarkan ruang serangan yang luas dan terus menarik untuk dipelajari. Memahami susunan perangkat keras, perangkat lunak, dan arsitektur sistem-sistem ini merupakan cara terbaik untuk menemukan kelemahan keamanan serta menyusun eksploitasi. Untungnya, teknik-teknik yang dipelajari untuk menyerang ECU berbasis MCU sebagian besar dapat diterapkan saat mencoba menembus *High-Performance Computers* (HPC). Hal ini akan menjadi topik pembahasan kita di bab selanjutnya.

BAB 8
KEAMANAN DAN PENGUJIAN PENETRASI PADA HPC KENDARAAN

Kendaraan modern kini semakin banyak dirancang dengan mengandalkan beberapa komputer terpusat berkinerja tinggi atau *High-Performance Computer* (HPC). Komputer ini mengonsolidasikan fungsi dari berbagai *Electronic Control Unit* (ECU) berbasis *Microcontroller Unit* (MCU) yang telah kita pelajari pada bab sebelumnya. Berbeda dengan ECU konvensional yang bergantung pada *Real-Time Operating System* (RTOS) terbatas dengan satu tenant (pengguna/lingkungan eksekusi), HPC memanfaatkan perangkat *System-on-Chip* (SoC) berkinerja tinggi. SoC ini memuat inti CPU heterogen yang dapat dipartisi ke dalam beberapa domain perangkat lunak atau tenant yang berbeda. Dalam satu cip yang sama, beberapa inti dapat menjalankan sistem operasi yang kompleks seperti Linux, QNX, atau Android sementara inti lainnya menjalankan perangkat lunak berbasis RTOS untuk fungsi kontrol yang sangat bergantung pada ketepatan waktu (*time-critical*).

Basis kode yang beragam, antarmuka perangkat keras yang luas, serta posisinya dalam arsitektur kelistrikan/elektronik (*E/E architecture*) kendaraan menciptakan permukaan serangan (*attack surface*) yang jauh lebih besar dibandingkan ECU tradisional, sehingga menjadikannya target serangan yang lebih menarik. Biasanya, HPC menangani berbagai fungsi kendaraan seperti sistem infotainment, klaster instrumen, ADAS, dan konektivitas dengan memanfaatkan virtualisasi untuk isolasi. Kompleksitas ini, ditambah dengan sifat dinamis dari *Software-Defined Vehicle* (SDV) tempat HPC diimplementasikan, membuat ketahanan keamanan menjadi sulit untuk dicapai.

Dalam bab ini, kita akan mendalami aspek keamanan ofensif pada HPC, dengan melanjutkan pembahasan dari bab sebelumnya. Kita akan mulai dengan tinjauan mengenai arsitektur perangkat keras HPC dan software stack yang umum digunakan. Selanjutnya, kita akan mengkaji kelemahan-kelemahan yang sering ditemukan, baik pada perangkat keras maupun perangkat lunak.

Kemudian, kita akan menguraikan metodologi untuk menyusun rantai serangan (*attack chain*) terhadap perangkat berkinerja tinggi ini mulai dari pengintaian awal (*reconnaissance*) dan ekstraksi *firmware*, hingga eksploitasi dan upaya mempertahankan akses (*persistence*). Beberapa studi kasus akan disajikan untuk mengilustrasikan insiden peretasan HPC di dunia nyata beserta dampaknya. Terakhir, kita akan membahas praktik terbaik keamanan dan langkah pertahanan untuk menangkal serangan yang paling umum, dengan menekankan pentingnya pendekatan keamanan berlapis. Di akhir bab ini, Anda akan memperoleh wawasan mengenai teknik-teknik yang digunakan untuk meretas HPC serta cara untuk meminimalkan efektivitas serangan tersebut.

Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

- Pengenalan ECU berkinerja tinggi (*High-Performance ECU*/HPC)
- Memahami anatomi perangkat keras dan perangkat lunak

- Mengidentifikasi kelemahan umum pada HPC
- Menjelajahi teknik pengujian penetrasi (*penetration testing*)
- Studi kasus mengenai insiden keamanan yang diungkapkan ke publik yang melibatkan HPC
- Praktik terbaik untuk mengamankan HPC

8.1 KONSEP DAN ARSITEKTUR HIGH-PERFORMANCE COMPUTER (HPC)

Istilah ECU berkinerja tinggi (*High-Performance ECU* atau HPC) muncul seiring dengan hadirnya kendaraan berbasis perangkat lunak (*Software-Defined Vehicle*/SDV) dan tren transisi dari arsitektur kelistrikan/elektronik (E/E) yang sangat terdistribusi menuju arsitektur berbasis zona dan terpusat. Perubahan ini menyebabkan pergeseran fungsionalitas dari desain berbasis mikrokontroler (MCU) tradisional ke sistem berbasis *System-on-Chip* (SoC).

Didorong oleh meningkatnya permintaan akan daya komputasi tinggi untuk sistem kemudi otonom dan sistem kokpit yang memerlukan antarmuka jaringan yang lebih luas serta aplikasi konektivitas canggih HPC telah mendapatkan penetrasi yang signifikan di pasar otomotif dan tren ini terus berlanjut. Awalnya hanya digunakan secara terbatas untuk telematika dan sistem hiburan dalam kendaraan (*infotainment*), penggunaan HPC kini meluas ke central *gateway*, sistem bantuan pengemudi tingkat lanjut (ADAS), dan yang terbaru, sebagai komputer pusat kendaraan. HPC dibedakan dari ECU berbasis MCU melalui sistem operasinya yang kaya fitur dan antarmuka jaringan yang menyerupai komputer pribadi (PC). Berbeda dengan sistem berbasis MCU yang sangat mengutamakan kinerja *real-time*, HPC dapat menoleransi variasi dalam kinerja *real-time*. Bukan berarti determinisme tidak penting bagi HPC, namun persyaratannya tidak seketat sistem hard *real-time* seperti pada pengontrol mesin atau ECU kantung udara (*airbag*).

Selain itu, HPC semakin banyak mengonsolidasikan berbagai domain kendaraan ke dalam satu desain berbasis perangkat keras, seperti sistem infotainment, kluster instrumen, konektivitas eksternal, hingga sistem bantuan pengemudi tingkat lanjut. Konsolidasi ini menciptakan kompleksitas yang sangat besar, sehingga memerlukan isolasi yang kuat untuk memastikan bahwa gangguan pada satu domain misalnya sistem *infotainment* tidak merambat ke kontrol yang krusial bagi keselamatan.

Untuk mencapai hal tersebut, desain HPC biasanya menggunakan *hypervisor* tipe-1 (*bare-metal*) guna menjalankan beberapa instansi sistem operasi tamu (*guest OS*) secara berdampingan pada SoC yang sama. *Hypervisor* ini menjamin isolasi yang kuat di antara partisi dengan tingkat kekritisan yang berbeda; contohnya, kluster instrumen dengan standar ASIL-B dapat berjalan berdampingan dengan mesin virtual (*Virtual Machine*/VM) Android yang menangani fungsi *infotainment* (IVI). *Hypervisor* Tipe-1 (QNX *Hypervisor*, Xen, Green Hills Integrity Multivisor, dll.) menyediakan CPU virtual, partisi memori, dan perangkat virtual bagi setiap VM.

Hypervisor ini juga dapat menengahi komunikasi antar-VM serta akses ke perangkat keras guna mencegah interferensi. Sebagai contoh, QNX *Hypervisor* dari BlackBerry mendukung perangkat paravirtualisasi VirtIO untuk berbagi jaringan dan penyimpanan antar-

VM, serta mengalokasikan perangkat keras fisik (seperti GPU, pengontrol CAN, dll.) kepada VM sesuai kebutuhan. Arsitektur perangkat lunak semacam ini diilustrasikan dalam Gambar 8.2, yang akan kita bahas lebih mendalam pada bagian mengenai arsitektur perangkat lunak HPC di bab ini.

Mengingat sebagian besar HPC menjalankan sistem operasi Linux atau Android, para penguji penetrasi (*penetration tester*) yang terbiasa mengevaluasi sistem perusahaan maupun perangkat seluler dapat menerapkan keahlian mereka dengan relatif mudah. Namun, sebelum mempelajari teknik-teknik pengujian penetrasi tersebut, kita perlu menelaah lebih dalam anatomi perangkat keras dan perangkat lunak dari HPC, yang akan kita bahas pada bagian selanjutnya.

8.2 MEMAHAMI ANATOMI PERANGKAT KERAS DAN PERANGKAT LUNAK HPC

Untuk mendapatkan perspektif yang lebih baik mengenai permukaan serangan (*attack surface*) pada HPC (*High-Performance Computer*), kita harus terlebih dahulu memahami arsitektur perangkat keras dan perangkat lunak yang lazim digunakan dalam sistem semacam itu.

Komponen Perangkat Keras HPC

Saat memeriksa papan sirkuit cetak (PCB) pada sebuah HPC, komponen yang biasanya paling mencolok adalah SoC (*System-on-Chip*) canggih, beragam teknologi memori dan penyimpanan, serta berbagai antarmuka konektivitas, sensor, dan port untuk keperluan debugging, sebagaimana diilustrasikan pada Gambar K.1.

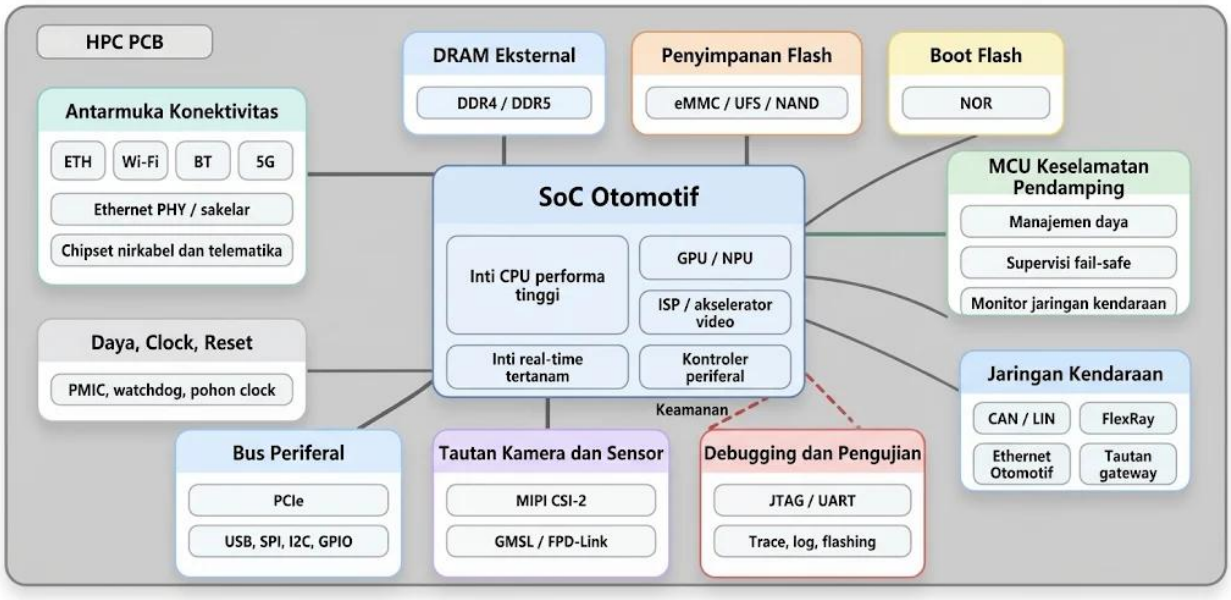
SoC Otomotif dan MCU Pendamping

HPC dibangun di seputar perangkat *System-on-Chip* (SoC) berkinerja tinggi yang sering kali mengintegrasikan klaster CPU canggih. Sebagai contoh, Anda mungkin menemukan inti (*core*) berkinerja tinggi seperti ARM A78 yang disandingkan dengan inti tertanam (*embedded core*) berbasis arsitektur RISC-V atau ARM Cortex-R/M. SoC ini biasanya dipasangkan dengan *Microcontroller Unit* (MCU) pendamping yang berfungsi sebagai pemantau keselamatan (*safety monitor*). MCU tersebut (contohnya Infineon AURIX atau Renesas RH850) dikhususkan untuk manajemen daya, operasi yang aman saat terjadi kegagalan (*fail-safe*), serta dapat berfungsi sebagai antarmuka jaringan kendaraan, sehingga memastikan sistem secara keseluruhan tetap dapat beroperasi meskipun komponen SoC mengalami kegagalan.

Memori dan Penyimpanan Eksternal

Berbeda dengan sistem berbasis MCU di mana memori *flash* dan RAM tertanam di dalam cip, HPC mengandalkan memori volatil maupun non-volatil eksternal. HPC umumnya dilengkapi dengan DRAM DDR4/DDR5 berkecepatan tinggi, dengan kapasitas mencapai beberapa gigabyte yang dibagikan ke berbagai lingkungan komputasi di dalam SoC. Untuk penyimpanan persisten, HPC menggunakan berbagai teknologi *flash* termasuk NAND konvensional, eMMC/UFS, hingga *Non-Volatile Memory Express* (NVMe) yang antara lain berguna untuk mendukung perekaman data video dalam jumlah besar. Sementara memori *flash* NOR biasanya digunakan untuk menyimpan *bootloader* dan *firmware* RTOS, modul

memori eMMC umumnya digunakan untuk menampung citra sistem operasi (*OS image*) yang lebih besar serta sistem berkas (*filesystem*).



Gambar 8.1: Diagram blok yang disederhanakan dari perangkat keras HPC

Antarmuka Konektivitas dan Bus Periferal

Selain pengontrol Ethernet multi-Gigabit dan switch Ethernet otomotif terintegrasi, HPC modern dilengkapi dengan opsi konektivitas canggih seperti chipset Bluetooth/Wi-Fi, dan dalam beberapa kasus, bahkan modem seluler. Fitur-fitur ini memungkinkan komunikasi untuk telematika, pembaruan OTA (*Over-the-Air*), dan layanan V2X (*Vehicle-to-Everything*).

Sebagaimana telah dibahas pada bab 5, kerentanan pada antarmuka komunikasi apa pun baik akibat konfigurasi yang tidak tepat, kelemahan protokol, maupun cacat pada driver perangkat lunak dapat menjadi titik masuk awal untuk menyisipkan kode berbahaya ke perangkat target. Karena HPC biasanya merupakan ECU pertama yang terpapar kerentanan semacam ini, maka HPC menjadi target utama untuk serangan tersebut.

Di sisi lain, bus periferal seperti PCIe memungkinkan penghubungan beberapa SoC dalam konfigurasi multi-chip, serta antarmuka dengan perangkat NVMe atau bahkan MCU pendamping. Antarmuka lain seperti USB, CAN, dan LIN menyediakan konektivitas ke perangkat pengguna dan protokol jaringan di dalam kendaraan. Selain itu, antarmuka untuk keperluan debug dan pengujian, seperti UART dan JTAG, sering kali disertakan untuk kebutuhan pengembangan dan pemeliharaan; antarmuka ini pun kerap menjadi sasaran untuk kegiatan pengintaian (*reconnaissance*), ekstraksi *firmware*, dan rekayasa balik (*reverse engineering*).

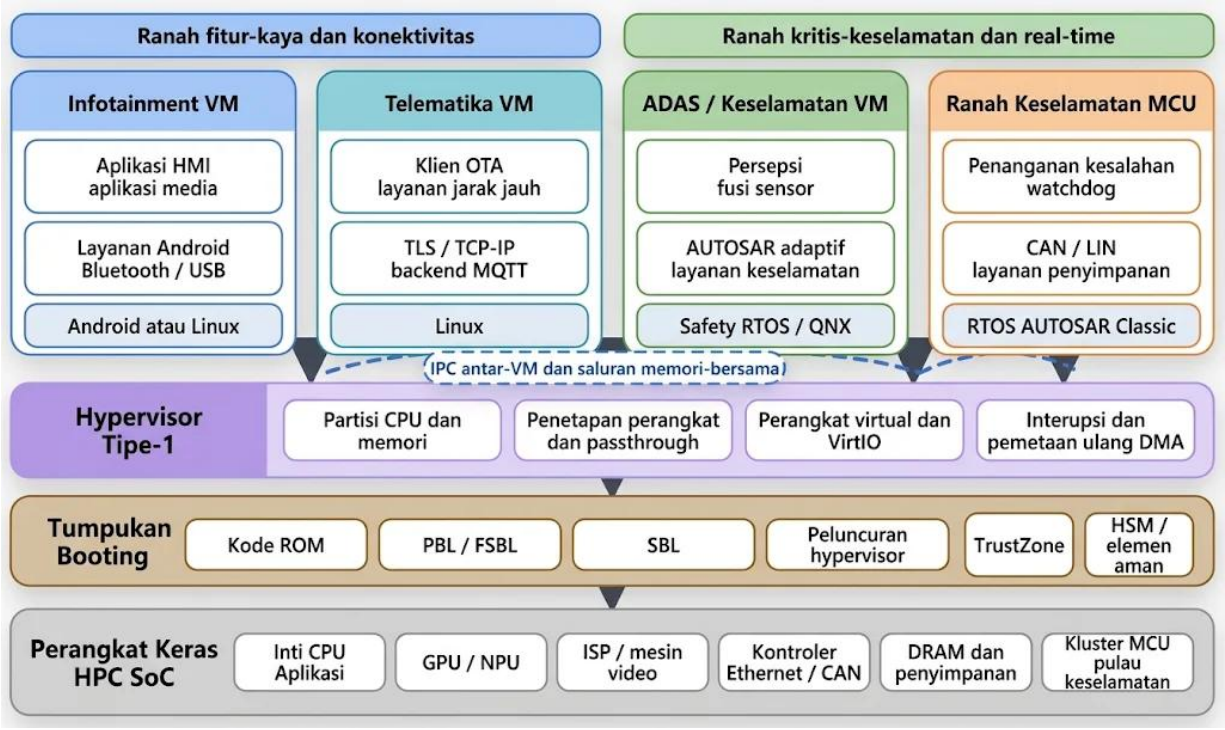
Antarmuka Kamera dan Sensor

Untuk aplikasi yang menggabungkan sistem kokpit dan ADAS (*Advanced Driver Assistance Systems*), HPC menyertakan antarmuka seperti MIPI (*Mobile Industry Processor Interface*) CSI-2 untuk kamera, serta antarmuka serial seperti GMSL (*Gigabit Multimedia Serial Link*) atau FPD-Link (*Flat Panel Display-Link*). Aliran data dengan bandwidth tinggi ini diproses

oleh ISP (*Image Signal Processor*) terintegrasi atau unit penangkap data khusus, yang sering kali ditempatkan dalam domain RTOS (*Real-Time Operating System*) di dalam HPC guna memenuhi persyaratan waktu nyata (*real-time*). Setelah memahami gambaran singkat mengenai komponen perangkat keras HPC, mari kita beralih ke aspek perangkat lunaknya.

Arsitektur Perangkat Lunak HPC

Tumpukan perangkat lunak dalam HPC bersifat berlapis dan beragam, mencerminkan berbagai lingkungan eksekusi waktu nyata yang hidup berdampingan dalam satu platform komputasi, sembari tetap mematuhi persyaratan keselamatan dan keamanan yang ketat.



Gambar 8.2: Diagram blok perangkat lunak tipikal pada HPC

Selanjutnya, kita akan membahas berbagai komponen dalam tumpukan perangkat lunak (*software stack*) HPC.

Bootloader dan Hypervisor

Saat perangkat dinyalakan, kode ROM melakukan pengaturan awal cip dan menjalankan *bootloader* tahap pertama. *Bootloader* ini dapat menginisialisasi satu atau beberapa inti (*core*) waktu nyata (*real-time*) yang mengelola berbagai mesin perangkat keras, sebelum mencapai tahap pemuatan *hypervisor*. *Hypervisor* kemudian mengatur partisi memori, menetapkan inti CPU dan perangkat keras ke mesin virtual (VM), lalu menjalankan setiap VM secara berurutan sembari tetap mempertahankan tingkat hak akses (*privilege level*) yang lebih tinggi. Sistem otomotif sering menggunakan *hypervisor* Tipe-1, seperti QNX *Hypervisor*, Xen, Green Hills INTEGRITY Multivisor, atau Wind River Helix. *Hypervisor* juga menyediakan lapisan pemantau (*monitor layer*) minimal untuk menangani komunikasi antar-VM serta mengelola akses perangkat melalui teknik seperti para-virtualisasi.

Sistem Operasi Tamu dan Partisi Perangkat Lunak

HPC (*High-Performance Computer*) biasanya menampung beberapa sistem operasi tamu yang dipisahkan ke dalam partisi perangkat lunak, dengan tetap menyesuaikan diri terhadap persyaratan fungsional, keselamatan, dan keamanan. Setiap sistem operasi tamu dijalankan di dalam mesin virtualnya sendiri dan diberi alokasi khusus berupa inti CPU, wilayah memori, serta akses ke perangkat keras oleh hipervisor. Pemisahan ini umumnya ditegakkan melalui dukungan virtualisasi perangkat keras (seperti penerjemahan alamat tahap kedua, penugasan perangkat, dan pemetaan ulang interupsi) dan menjadi batasan utama antara beban kerja yang krusial bagi keselamatan (*safety-critical*) dengan beban kerja yang tidak krusial bagi keselamatan.

Domain yang Krusial Bagi Keselamatan (Safety-Critical Domains)

Fungsi-fungsi yang menuntut integritas keselamatan otomotif tingkat tinggi biasanya ditempatkan dalam lingkungan sistem operasi tamu khusus yang bersifat real-time atau telah tersertifikasi untuk aspek keselamatan. Domain-domain ini umumnya menjalankan sistem operasi yang tersertifikasi keselamatan seperti QNX dan dikonfigurasi dengan isolasi temporal (waktu) serta spasial (ruang) yang ketat.

Sebagai contoh, sistem ADAS atau stack fusi persepsi dapat dijalankan di dalam mesin virtual (VM) khusus keselamatan, yang dilengkapi dengan inti CPU tersendiri, batasan waktu eksekusi, serta akses yang sangat dibatasi terhadap perangkat periferal. Selain itu, banyak platform menyertakan "pulau keselamatan" (*safety island*) atau klaster mikrokontroler lockstep di dalam SoC (*System-on-Chip*). Komponen ini beroperasi secara independen dari inti aplikasi utama dan berfungsi mengawasi aspek-aspek terkait keselamatan, seperti deteksi kesalahan, penanganan watchdog, serta transisi ke kondisi aman (*fail-safe*).

Domain Dengan Fitur Lengkap dan Konektivitas

Domain yang tidak krusial bagi keselamatan biasanya menjalankan sistem operasi seperti Linux atau Android, serta mendukung fungsi-fungsi seperti sistem hiburan (*infotainment*), telematika, konektivitas, antarmuka pengguna, dan komunikasi ke sistem *backend*. Sebuah VM telematika dapat mengelola konektivitas seluler, pembaruan perangkat lunak secara nirkabel (*over-the-air*), dan pertukaran pesan dengan *backend*; sementara VM infotainment dapat menjalankan aplikasi Android, pemrosesan multimedia, layanan Bluetooth, dan kerangka kerja antarmuka pengguna. Dalam beberapa desain, terdapat partisi berbasis Linux tambahan yang dikhususkan untuk pemrosesan data kamera, penyusunan tampilan sekeliling (*surround-view*), atau penyajian visual pada klaster instrumen. Domain-domain ini harus dipisahkan dari beban kerja yang krusial bagi keselamatan karena biasanya memiliki permukaan serangan (*attack surface*) yang jauh lebih luas akibat adanya antarmuka nirkabel, pengurai media (*media parsers*), dan kerangka kerja aplikasi yang kompleks.

Model Eksekusi Dengan Tingkat Kekritisan Campuran (Mixed-Criticality Execution Model)

Desain yang melibatkan banyak mesin virtual (VM) ini menciptakan lingkungan eksekusi dengan tingkat kekritisan campuran, di mana domain-domain yang memiliki model ancaman dan asumsi sertifikasi yang sangat berbeda dapat berjalan bersama-sama pada satu perangkat keras (silikon) yang sama. Domain yang kaya fitur biasanya berkembang lebih cepat,

menerima pembaruan lebih sering, serta bergantung pada tumpukan perangkat lunak sumber terbuka (*open-source*) yang besar.

Sebaliknya, domain yang krusial bagi keselamatan (*safety-critical*) berubah secara perlahan dan divalidasi melalui proses keselamatan yang ketat. Dari sudut pandang penyerang, asimetri ini merupakan elemen kunci dalam penyusunan rantai serangan (*attack-chain*). Akses awal jauh lebih mungkin diperoleh pada domain dengan kompleksitas tinggi seperti sistem hiburan dalam kendaraan (*infotainment*) atau telematika di mana langkah-langkah selanjutnya akan berupaya menembus batasan isolasi untuk masuk ke partisi yang lebih tepercaya atau yang berkaitan dengan aspek keselamatan.

Dengan demikian, partisi sistem operasi tamu (*guest operating system*) dalam HPC tidak boleh dipandang sekadar sebagai penerapan virtualisasi sederhana, melainkan sebagai strategi isolasi berlapis yang menggabungkan penegakan berbasis perangkat keras, kebijakan *hypervisor*, penugasan perangkat, serta antarmuka layanan yang dibatasi secara ketat. Keamanan platform secara keseluruhan tidak hanya bergantung pada ketangguhan *hypervisor*, tetapi juga pada ketepatan dan prinsip minimalisme dalam model partisi yang mengatur bagaimana sistem operasi tamu berinteraksi dengan perangkat keras maupun satu sama lain. Prinsip minimalisme berarti setiap sistem operasi tamu hanya diberikan akses ke memori, periferal, sumber daya bersama, dan jalur komunikasi yang memang diperlukan untuk menjalankan fungsi yang dituju. Prinsip ini dibahas lebih lanjut dalam bagian mengenai partisi dalam arsitektur HPC berbasis *hypervisor*.

Domain Perangkat Lunak MCU

Sebagaimana telah disebutkan sebelumnya, arsitektur perangkat lunak MCU dapat ditempatkan sepenuhnya di dalam HPC untuk menangani fungsi-fungsi seperti penanganan kesalahan keselamatan (*safety error handling*) dalam suatu *functional safety island*, guna mencapai target ASIL (*Automotive Safety Integrity Level*) yang lebih tinggi. Oleh karena itu, sering kali ditemukan satu atau lebih instansi *AUTOSAR Classic* yang berjalan di dalam HPC. Namun, terdapat beberapa perbedaan dalam cara perangkat lunak tersebut dimuat (*booting*) dan dioperasikan dibandingkan dengan ECU yang berbasis MCU murni.

Sebagai contoh, perangkat lunak MCU dalam HPC dimuat dari penyimpanan eksternal ke dalam DRAM atau SRAM yang telah dipartisi khusus untuk lingkungan eksekusi MCU tersebut. *Bootloader stack* yang bertanggung jawab untuk memuat seluruh sistem HPC juga akan menangani proses booting untuk perangkat lunak MCU. Alih-alih mengambil instruksi dari memori *flash* internal, perangkat lunak MCU akan mengambil instruksi dari partisi RAM yang telah dialokasikan untuknya.

Terkait penyimpanan data persisten, alih-alih mengandalkan data*flash* tertanam, perangkat lunak MCU harus menggunakan layanan penyimpanan dari HPC untuk membaca dan menulis data ke media penyimpanan yang lebih tangguh, seperti memori *flash* tipe NOR. Selain itu, domain MCU akan berkomunikasi dengan bagian lain dari kendaraan layaknya MCU pada umumnya, yakni dengan memanfaatkan protokol CAN, LIN, dan Ethernet. Domain ini juga akan mendukung protokol IPC (*Inter-Process Communication*) untuk berkomunikasi dengan

domain aplikasi di dalam HPC itu sendiri misalnya, untuk mengeluarkan perintah tertentu jika terjadi suatu kesalahan (*fault*).

Komunikasi Antarproses dan Antar-VM

Dengan banyaknya sistem operasi dan prosesor yang berbagi sumber daya perangkat keras, mekanisme komunikasi antarproses (*inter-process communication*/IPC) yang tangguh menjadi sangat penting. *Hypervisor* biasanya mengimplementasikan saluran komunikasi virtual, menggunakan metode seperti penyangga memori bersama (*shared memory buffer*) untuk memfasilitasi pertukaran data yang aman antar-VM. Perangkat lunak driver tambahan di dalam VM menangani pengaturan saluran logis serta autentikasi layanan guna menghasilkan solusi komunikasi menyeluruh (*end-to-end*). Akan tetapi, jika saluran-saluran ini tidak dirancang dan diisolasi dengan cermat, saluran tersebut dapat menjadi jalur serangan yang memungkinkan pergerakan lateral melintasi domain fungsional (misalnya, dari sistem infotainment ke sistem ADAS).

Protokol Komunikasi Jaringan

HPC (*High-Performance Computer*) memiliki serangkaian protokol jaringan yang dirancang untuk menangani berbagai kebutuhan komunikasi. Untuk komunikasi di dalam kendaraan, biasanya digunakan protokol SOME/IP, yang memungkinkan komunikasi dengan ECU berbasis sinyal melalui Ethernet. Selain itu, protokol standar seperti CAN dan FlexRay beserta lapisan komunikasi berbasis sinyal yang menyertainya juga tetap digunakan. Untuk konektivitas eksternal, baik ke sistem *backend* maupun ke HPC lain seperti sistem *In-Vehicle Infotainment* (IVI), biasanya digunakan protokol TLS dengan tumpukan perangkat lunak (*software stack*) TCP/IP. Jika MACsec diterapkan, maka akan terdapat komponen WP-supplicant serta driver Ethernet. Perangkat lunak komunikasi lain yang sering ditemui mencakup klien MQTT untuk berkomunikasi dengan *backend*, dan terkadang juga DDS.

Akses Perangkat, Passthrough, dan Alat Bantu Debugging

Perangkat lunak pada HPC sering kali menggunakan mekanisme device passthrough untuk periferal yang membutuhkan performa tinggi; contohnya, memberikan akses langsung kepada VM Linux ke pengontrol Ethernet, atau memberikan kendali eksklusif atas antarmuka USB kepada VM Android. Meskipun metode ini meminimalkan *overhead*, kerentanan pada driver di dalam sistem tamu (*guest*) dapat dimanfaatkan untuk mengeksploitasi perangkat keras dasar tanpa banyak campur tangan dari sistem pemantau seperti *hypervisor*. Selain itu, HPC sering kali menyediakan antarmuka untuk keperluan *flashing, debugging, profiling*, dan pencatatan log (seperti SSH, Telnet, GDB, atau *verbose logging*). Meski sangat berguna selama tahap pengembangan, jika alat atau antarmuka ini tetap aktif pada tahap produksi, maka alat tersebut dapat menjadi vektor serangan untuk melakukan pengintaian (*reconnaissance*) dan mendapatkan akses awal.

Firmware untuk Mesin Perangkat Keras Khusus

HPC mengelola berbagai perangkat keras khusus yang diperlukan untuk aplikasi-aplikasi tertentu, seperti *Image Signal Processor* (ISP), dekoder/enkoder video, mesin inferensi deep learning, sistem manajemen daya/suhu, serta GPU tertanam (*embedded GPU*). Biasanya, mesin perangkat keras ini dipasangkan dengan inti pemroses tertanam (seperti arsitektur RISC-

V) yang bertugas mengatur konfigurasi perangkat keras dan mengoordinasikan eksekusi tugas. Periferal-periferal ini, beserta *firmware* khususnya, memperluas permukaan serangan (*attack surface*) dengan cara yang sering kali tidak disadari hal yang akan kita bahas lebih lanjut di bab ini. Setelah meninjau komponen perangkat keras dan perangkat lunak yang umum terdapat pada HPC, kini kita akan beralih membahas kelemahan-kelemahan yang sering ditemukan pada sistem-sistem tersebut.

8.3 MENGIDENTIFIKASI KELEMAHAN UMUM PADA HPC

Tidaklah mungkin untuk mencantumkan seluruh kelemahan keamanan yang mungkin dimiliki oleh sebuah HPC dalam satu bab saja. Oleh karena itu, sebagai pengantar topik ini, kami akan menyoroti beberapa kelemahan keamanan yang paling umum yang perlu dipertimbangkan oleh seorang penguji penetrasi (*penetration tester*) saat mengevaluasi sebuah HPC. Tentu saja, beberapa kelemahan akan tumpang tindih dengan kelemahan yang ditemukan pada sistem berbasis MCU. Jika hal ini terjadi, kami tidak akan mengulangi aspek-aspek dasar yang telah dibahas pada bab sebelumnya, melainkan hanya akan berfokus pada aspek-aspek yang unik.

Kelemahan Perangkat Keras

Sebagaimana akan kita lihat sepanjang bagian ini, beberapa kelemahan keamanan perangkat keras berguna selama fase pengintaian (*reconnaissance*) karena membantu dalam mengefiltrasi data dan membangun akses awal, sementara kelemahan lainnya berguna saat menyusun rantai serangan selama eksploitasi aktif. Anda disarankan untuk mempertimbangkan kedua kategori ini saat mengevaluasi kegunaan dari suatu kelemahan keamanan yang ditemukan.

Akses Debug dan Periferal yang Tidak Aman

Seperti yang dibahas pada bab sebelumnya, selama fase pengintaian, penyerang dapat mengeksploitasi fitur perangkat keras yang terbuka untuk mengekstrak *firmware* atau rahasia kriptografis tanpa harus mengompromikan sistem sepenuhnya. Salah satu kelemahan mendasar dalam fase ini adalah keberadaan antarmuka debug dan pengujian, seperti JTAG atau porta debug khusus vendor. Antarmuka ini, yang ditujukan untuk keperluan pengembangan dan manufaktur, memberikan akses tingkat rendah ke bagian internal pengontrol (*controller*).

Pada *High-Performance Computer* (HPC) otomotif, pemasok *Electronic Control Unit* (ECU) diharapkan untuk menonaktifkan atau membatasi antarmuka debug pada perangkat produksi, biasanya dengan memutus fuse bit atau memprogram memori yang hanya dapat diprogram sekali (*One-Time Programmable* atau OTP). Namun, perlindungan berbasis fuse dan OTP tidaklah sempurna. Penelitian sebelumnya telah menunjukkan berbagai jenis serangan terhadap mekanisme tersebut, termasuk *voltage glitching* atau *clock glitching* untuk melewati pemeriksaan fuse, serta serangan semi-invasif terhadap struktur anti-fuse. Akibatnya, keberadaan antarmuka debug meskipun secara resmi telah dinonaktifkan tetap harus dianggap sebagai potensi permukaan serangan (*attack surface*) bagi penyerang yang gigih.

Kelemahan lain muncul akibat kesalahan dalam pengelolaan siklus hidup (*lifecycle management*) dan perlindungan debug: Saat platform beralih dari tahap manufaktur ke tahap produksi, kapabilitas debug seharusnya menjadi lebih terbatas. Namun, jika pembatasan tersebut tidak diterapkan dengan benar, fitur debug dengan hak akses tinggi (*privileged*) mungkin tetap dapat diakses atau dibuka kembali, sehingga secara signifikan mengurangi upaya yang diperlukan untuk mengekstrak *firmware* atau melakukan introspeksi sistem.

Porta USB yang tidak terlindungi merupakan vektor pengintaian lain yang menarik. Meskipun antarmuka USB sering kali diperlukan selama proses manufaktur atau servis untuk memasang *firmware*, membiarkannya tetap aktif atau tidak dibatasi dengan memadai selama operasi normal merupakan sebuah kelemahan keamanan. Jika digabungkan dengan saluran *Direct Memory Access* (DMA) yang salah konfigurasi, perangkat USB berbahaya dapat melakukan akses memori tanpa izin atau injeksi kode, yang menjadi pijakan untuk ekstraksi informasi lebih lanjut atau kompromi sistem. Pada sistem yang memerlukan penggunaan USB selama operasi normal seperti platform *In-Vehicle Infotainment* (IVI) isolasi yang lemah terhadap transaksi DMA melalui USB juga dapat disalahgunakan sebagai primitif serangan.

Kelemahan lain yang memfasilitasi kegiatan pengintaian terletak pada cara HPC menangani penyimpanan eksternal dan antarmuka memori. Karena ketergantungannya pada penyimpanan non-volatil dan RAM yang berada di luar cip (*off-chip*), penyerang yang memegang fisik HPC dapat memeriksa perangkat *flash* SPI atau eMMC secara langsung, ataupun menyadap bus memori DDR menggunakan perangkat perantara (*interposer*), guna menangkap citra *firmware* atau data yang sedang ditransmisikan. Jika kode dan informasi rahasia dikirimkan melalui bus-bus tersebut dalam bentuk teks polos (*plaintext*), pihak lawan yang mencegatnya dapat dengan mudah mengeksfiltrasi materi sensitif tersebut. Dalam praktiknya, jika konten memori di luar cip serta bus antarmuka tidak dienkripsi dan diautentikasi, penyerang dapat melepas perangkat memori (dengan menyolder ulang) atau memantau bus memori untuk memperoleh data dan *firmware* dalam jumlah besar.

Isolasi Perangkat Keras yang Lemah

Kelemahan perangkat keras yang merusak asumsi isolasi perangkat lunak menempati urutan teratas dalam daftar hal yang coba dieksploitasi oleh penyerang saat menargetkan HPC (*High-Performance Computer*/Komputer Berkinerja Tinggi). Salah satu kelemahan umum muncul dari konfigurasi MMU (*Memory Management Unit*) yang bersifat kasar (*coarse-grained*) atau pemetaan memori yang terlalu longgar. Meskipun HPC modern biasanya mengandalkan MMU (dengan penerjemahan alamat bertahap di bawah virtualisasi) untuk mengisolasi mesin virtual dan proses di dalamnya, kesalahan konfigurasi dapat dengan mudah menyebabkan cakupan akses melampaui apa yang sebenarnya dibutuhkan oleh beban kerja tertentu.

Contohnya termasuk pemetaan wilayah memori fisik yang luas dan berkesinambungan alih-alih menggunakan penyangga (*buffer*) dengan cakupan terbatas, penggunaan kembali tabel halaman (*page table*) umum di berbagai domain, atau penyediaan wilayah memori bersama untuk kemudahan selama pengembangan yang kemudian tetap dibiarkan aktif dalam lingkungan produksi. Kesalahan konfigurasi semacam ini memperluas permukaan serangan

dengan memberikan akses kepada mesin virtual atau komponen di ruang pengguna (*user-space*) untuk melihat wilayah memori milik domain dengan hak akses lebih tinggi atau tingkat kepentingan yang lebih krusial.

Selain isolasi memori, banyak HPC otomotif menerapkan mekanisme *firewall* perangkat keras atau filter akses register untuk melindungi register yang dipetakan ke memori (*memory-mapped registers*) serta antarmuka kontrol yang kritis. Filter ini bertujuan untuk membatasi pihak mana baik itu bus master, tingkat hak akses, maupun lingkungan eksekusi yang dapat mengakses rentang register tertentu, misalnya register yang mengontrol sistem pewaktuan (*clocking*), logika reset, mekanisme keselamatan, atau konfigurasi periferal. Jika dikonfigurasi dengan benar, filter register tersebut menjadi pertahanan terakhir yang krusial untuk mencegah konfigurasi ulang perangkat keras kritis, baik yang terjadi secara tidak sengaja maupun akibat tindakan berbahaya.

Dalam praktiknya, menerapkan kontrol akses dengan prinsip hak akses paling minimal (*least-privilege*) secara konsisten di seluruh mekanisme ini merupakan tantangan besar dalam skala yang luas. HPC otomotif dapat memiliki ribuan register yang dipetakan ke memori di berbagai CPU, GPU, akselerator, dan pengontrol periferal, yang masing-masing memiliki kebutuhan akses berbeda di berbagai tahapan booting, sistem operasi, *hypervisor*, dan pemantau keselamatan (*safety monitor*). Mengoordinasikan izin akses tersebut di antara berbagai tim perangkat lunak dan tahapan siklus hidup sistem menghadirkan tantangan signifikan yang sering kali berujung pada kesalahan. Jika prinsip hak akses paling minimal tidak ditegakkan secara ketat, filter register dan kebijakan kontrol akses bisa menjadi jauh lebih longgar daripada yang seharusnya, sehingga melemahkan batasan isolasi yang ada.

Begitu kelemahan semacam itu ada, kelemahan tersebut menjadi elemen dasar yang sangat berharga setelah sistem berhasil disusupi. Penyerang yang mendapatkan kemampuan eksekusi kode di dalam mesin virtual atau lingkungan *user-space* dengan tingkat kepentingan (*criticality*) lebih rendah dapat memanfaatkan pemetaan MMU (*Memory Management Unit*) yang terlalu luas atau kontrol akses register yang salah konfigurasi untuk meningkatkan hak akses mereka.

Tindakan ini bisa mencakup modifikasi register konfigurasi periferal, penonaktifan pemeriksaan keamanan, pemrograman ulang mesin DMA (*Direct Memory Access*), atau bahkan gangguan terhadap mekanisme isolasi itu sendiri. Yang terpenting, serangan ini tidak mengharuskan eksploitasi kerentanan perangkat lunak dalam kode isolasi; serangan ini hanya menyalahgunakan akses perangkat keras yang sah namun dikonfigurasi secara keliru.

Dari perspektif desain sistem, isolasi perangkat keras yang lemah akan meruntuhkan asumsi dasar yang menopang virtualisasi, penerapan sistem dengan tingkat kepentingan campuran (*mixed-ASIL*), serta strategi pertahanan berlapis (*defense-in-depth*). Bahkan ketika *hypervisor* dan sistem operasi telah diimplementasikan dengan benar, MMU atau firewall perangkat keras yang salah konfigurasi dapat meruntuhkan batasan isolasi di tingkat silikon. Oleh karena itu, kegagalan isolasi yang disebabkan oleh pemetaan memori yang terlalu luas (*coarse-grained*) atau kontrol akses register yang terlalu longgar harus dipandang sebagai

risiko keamanan arsitektural, bukan sekadar bug implementasi terutama pada platform komputasi berkinerja tinggi (HPC) di sektor otomotif yang krusial bagi keselamatan.

Kelemahan Pada GPU

GPU sering diintegrasikan ke dalam sistem komputasi berkinerja tinggi (HPC) untuk mempercepat beban kerja yang berkaitan dengan persepsi, visualisasi, dan kecerdasan buatan (AI), yang biasanya mencakup berbagai domain dengan tingkat kepentingan yang berbeda-beda. GPU dirancang untuk menjalankan beban kerja yang sangat paralel menggunakan shader yang dapat diprogram atau compute kernel; eksekusinya sebagian besar diatur oleh driver yang kompleks dan *runtime stack*. Hal ini berbeda dengan CPU, yang menjalankan program tujuan umum secara langsung dengan kontrol alur eksekusi yang sangat terperinci (*fine-grained*). Kemampuan untuk diprogram ini membuat kontrol eksekusi pada GPU sangat berbeda dari model CPU konvensional yang menggunakan izin eksekusi berbasis halaman (*page-based*). Penelitian sebelumnya menunjukkan bahwa proses di *user-space* dapat menyalahgunakan jalur pengiriman perintah GPU untuk memuat dan menjalankan beban kerja GPU yang berbahaya.

Tanpa adanya perlindungan eksekusi yang ditegakkan oleh perangkat keras, setiap wilayah memori yang digunakan GPU untuk mengambil instruksi akan secara otomatis menjadi area yang dapat dieksekusi. Akibatnya, jika penyerang mampu merusak atau memengaruhi memori GPU baik secara langsung maupun tidak langsung melalui API *user-space* mereka dapat menyisipkan kernel GPU berbahaya yang kemudian dieksekusi oleh GPU tanpa melanggar aturan perlindungan arsitektural apa pun.

Risiko ini diperparah oleh fakta bahwa memori GPU sering kali digunakan bersama oleh berbagai konteks eksekusi dan diakses melalui *driver stack* yang kompleks. Isolasi yang lemah antar-konteks GPU, mekanisme berbagi penyangga (*buffer*) yang terlalu longgar, atau kerentanan pada API GPU di ruang pengguna (*user-space*) dapat memungkinkan penulisan yang tidak diinginkan ke dalam wilayah memori GPU yang dapat dieksekusi. Penelitian sebelumnya telah menunjukkan bahwa proses di ruang pengguna dapat menyalahgunakan antarmuka pengiriman perintah GPU untuk mengunggah dan menjalankan kode GPU arbitrer, bahkan di lingkungan di mana kontrol eksekusi di sisi CPU tetap utuh. Dalam kasus seperti itu, ketiadaan perlindungan serupa NX (*No-Execute*) pada memori GPU secara langsung memungkinkan eksekusi kode, bukan sekadar kerusakan data.

Dari perspektif keamanan sistem, eksekusi kernel GPU yang berbahaya sangat mengkhawatirkan karena GPU sering kali beroperasi dengan hak akses yang lebih tinggi terhadap memori sistem dan periferal. GPU dapat memiliki akses DMA (*Direct Memory Access*) yang luas ke RAM sistem, akses ke wilayah memori yang dibagi dengan CPU, atau visibilitas terhadap framebuffer dan aliran data sensor. Dengan demikian, kernel GPU yang berbahaya dapat bertindak sebagai konteks eksekusi terselubung yang sebagian besar tidak terlihat oleh alat pemantau keamanan berbasis CPU konvensional.

Studi akademis telah menunjukkan bahwa perangkat lunak berbahaya (*malware*) berbasis GPU dapat bertahan tanpa terdeteksi, melakukan akses memori secara diam-diam, dan bahkan membantu mengekstrak informasi rahasia dari memori CPU melalui DMA atau

teknik saluran samping (*side-channel*). Jika memori GPU dapat diakses dari ruang pengguna yang tidak kritis terhadap keselamatan (*non-safety-critical*) dan tidak memiliki perlindungan eksekusi yang dapat ditegakkan, beban kerja (*workload*) yang tidak kritis namun telah disusupi berpotensi menyuntikkan kode GPU yang memanipulasi data milik fungsi-fungsi yang krusial bagi keselamatan.

Hal ini merusak asumsi mengenai pemisahan antara domain ASIL (*Automotive Safety Integrity Level*) dan non-ASIL, terutama pada arsitektur di mana GPU diperlakukan sebagai akselerator bersama (*shared accelerator*) alih-alih periferal yang diisolasi secara ketat (*sandboxed*). Akibatnya, kelemahan terkait perlindungan eksekusi pada GPU harus dipandang sebagai risiko struktural dalam sistem yang mengandalkan GPU untuk beban kerja kritis terutama jika dikombinasikan dengan isolasi driver yang lemah, akses DMA yang longgar, atau arsitektur memori bersama.

Kebocoran Saluran Samping (Side-Channel Leakage)

Seperti sistem komputer pada umumnya, HPC (*High-Performance Computer*) rentan terhadap berbagai serangan saluran samping yang membantu pihak lawan dalam tahap pengintaian dengan cara mengekstrak informasi rahasia melalui metode tidak langsung. Serangan ini memanfaatkan emisi sistem yang tidak disengaja, seperti durasi waktu (*timing*), konsumsi daya, dan radiasi elektromagnetik. Salah satu contoh klasik adalah *Differential Power Analysis* (DPA).

Dengan mengukur konsumsi daya HPC saat melakukan operasi kriptografi, penyerang dapat menyimpulkan kunci kriptografi yang digunakan melalui analisis statistik. Teknik ini terbukti mampu mengekstrak kunci enkripsi AES dengan upaya pengukuran yang relatif sederhana dalam kondisi laboratorium. Melalui pengambilan jejak daya (*power traces*) dan penerapan analisis statistik, kunci rahasia dapat disimpulkan secara bertahap, sebagaimana ditunjukkan dalam berbagai penelitian akademis terhadap perangkat nyata.

Risiko menjadi jauh lebih serius ketika kunci kriptografi digunakan secara global di seluruh unit HPC. Penyerang yang mendapatkan akses fisik ke unit HPC misalnya dari kendaraan yang sudah tidak terpakai atau melalui suku cadang yang dibeli di pasar sekunder dapat melakukan analisis tersebut secara *offline* untuk mengungkap rahasia yang digunakan bersama di seluruh armada kendaraan. Rahasia ini kemudian dapat dimanfaatkan dalam tahap eksploitasi selanjutnya terhadap kendaraan lain yang menggunakan materi kriptografi yang sama.

Kelemahan umum yang mempermudah ekstraksi rahasia melalui analisis saluran samping mencakup tidak adanya langkah pencegahan fisik maupun logis seperti penggunaan pelindung (*shielding*) atau pembangkitan derau (*noise generation*) yang seharusnya dapat meningkatkan jumlah jejak data yang diperlukan untuk memulihkan sebuah kunci secara signifikan. Dengan perancangan perangkat lunak yang cermat, pengumpulan jejak data dapat dibuat menjadi tidak praktis, misalnya dengan melakukan rotasi kunci atau membatasi periode penggunaan kunci (*cryptoperiod*), sehingga mempersempit rentang waktu yang tersedia untuk memperoleh pengukuran yang memadai.

Kelemahan lain yang berkontribusi adalah kurangnya akselerasi perangkat keras untuk algoritma kriptografi tertentu, yang memaksa sistem untuk bergantung pada implementasi berbasis perangkat lunak. Implementasi tersebut seringkali lebih rentan terhadap serangan *side-channel* berbasis waktu, terutama ketika dieksekusi pada inti CPU tujuan umum, karena variasi waktu eksekusi atau pola akses memori dapat membocorkan informasi tentang operasi yang bergantung pada rahasia. Ketergantungan pada implementasi kriptografi perangkat lunak bahkan lebih umum di HPC daripada ECU berbasis MCU karena lingkungan eksekusi yang kaya yang bergantung pada layanan canggih seperti TLS yang membutuhkan *throughput* tinggi untuk enkripsi dan otentikasi data sesi.

Penanganan Kunci Kriptografi

Kelemahan dalam penanganan kunci kriptografi seringkali berasal dari bagaimana dan di mana rahasia disimpan, diturunkan, dan diekspos di dalam SoC. Array fuse, misalnya, umumnya digunakan untuk konfigurasi chip dan sebagai akar untuk penurunan kunci. Namun, array tersebut rentan terhadap eksfiltrasi melalui serangan fisik invasif. Jika rahasia global disimpan langsung di eFuse, risiko paparannya sangat tinggi, karena penyerang dengan kemampuan yang cukup dapat mengekstrak isi fuse menggunakan teknik seperti mikroskop elektron. Setelah rahasia global semacam itu berhasil direbut, hal itu akan merusak keamanan semua perangkat yang bergantung padanya. Selain itu, implementasi manajemen siklus hidup berbasis fuse yang lemah dapat mengakibatkan penurunan keamanan jika transisi status tidak cukup membatasi akses ke kunci kriptografi. Misalnya, transisi dari status normal ke RMA seharusnya membuat kunci kriptografi tertentu tidak dapat digunakan.

Kelemahan terkait adalah ketergantungan pada kunci tingkat RTL bersama untuk mendapatkan rahasia khusus chip. Kunci-kunci ini tertanam dalam silikon dan merupakan target bernilai tinggi: penyerang yang berhasil merebutnya seringkali dapat merusak keamanan seluruh kelas perangkat daripada hanya satu unit. Hal ini menciptakan dampak yang tidak proporsional dari satu ekstraksi yang berhasil dan secara signifikan melemahkan jaminan keamanan di seluruh armada.

Pola kelemahan umum lainnya melibatkan penyimpanan kunci kriptografi dalam memori *flash* tujuan umum daripada dalam penyimpanan aman khusus. Kunci yang disimpan dalam *flash* biasa rentan tidak hanya terhadap ekstraksi tetapi juga terhadap perusakan, penggantian, atau pengembalian. Selama pengintaian, penyerang secara rutin memindai isi *flash* untuk kredensial atau materi kriptografi. Bahkan ketika kunci tersebut dienkripsi, kontrol versi *firmware* yang lemah atau kurangnya perlindungan *rollback* masih dapat memungkinkan penyerang untuk memulihkan keadaan yang lebih lama dan lebih lemah, seperti citra pra-produksi atau yang diaktifkan untuk debugging, kecuali jika mekanisme anti-rollback yang kuat diberlakukan.

Meskipun kemajuan telah dibuat dalam mewajibkan penggunaan teknologi *Hardware Platform Security Engine* (HPSE) dalam SoC tertanam, kelemahan masih dapat muncul dari partisi arsitektur yang buruk. Dalam beberapa desain, kunci kriptografi di-buffer atau ditangani langsung oleh *firmware* HPSE yang berjalan di dalam lingkungan eksekusi yang aman daripada dibatasi secara ketat di dalam mesin kriptografi khusus dan hanya diekspos ke *firmware*

melalui akses berbasis handle. Hal ini memperluas basis komputasi tepercaya dan meningkatkan risiko: jika penyerang dengan seseorang memiliki akses fisik, mereka mungkin dapat mengekstrak kunci-kunci sensitif dengan memanfaatkan kerentanan pada *firmware* HPSE. Bergantung pada peran kunci tersebut, hal ini dapat memungkinkan dekripsi kekayaan intelektual (IP) eksklusif atau pembuatan tanda tangan digital palsu. Pada bagian-bagian selanjutnya, kita akan melihat bagaimana desain *firmware* HPSE yang buruk dapat memperparah risiko keamanan ini.

Kerentanan CPU

Jika Anda rutin mengikuti pengungkapan keamanan CPU, Anda mungkin tidak akan terkejut mengetahui bahwa kerentanan yang berkaitan dengan CPU sering kali dipublikasikan oleh vendor-vendor utama CPU. Hal ini bukanlah suatu kebetulan. Desain CPU modern terus-menerus dituntut untuk menghasilkan performa yang lebih tinggi melalui teknik spekulasi yang agresif, penggunaan cache, dan paralelisme. Ketika berbagai optimasi ini melampaui atau melanggar asumsi keamanan mendasar, terciptalah celah yang rentan untuk dieksploitasi. Meskipun kerentanan jenis Spectre dan Meltdown kini telah dipahami secara umum, keduanya hanyalah sebagian kecil dari cakupan masalah yang lebih luas dan terus berkembang. Jenis-jenis kerentanan baru pun kerap muncul, yang berpotensi mengubah model keamanan menyeluruh dari tumpukan perangkat lunak (*software stack*).

Banyak dari masalah ini berakar pada kelemahan halus dalam mekanisme mikroarsitektur yang diandalkan secara implisit oleh perangkat lunak. Contohnya mencakup jalur eksekusi spekulatif yang mampu melewati pemeriksaan *pointer authentication code* (PAC), proses pembatalan (*invalidation*) entri *translation lookaside buffer* (TLB) yang tidak lengkap atau tertunda saat terjadi pergantian konteks (*context switch*), serta isolasi status spekulatif yang tidak memadai di antara batasan hak akses (*privilege boundaries*). Tema yang berulang di sini adalah keterbukaan (*exposure*): data rahasia yang dikelola oleh CPU menjadi dapat diamati secara arsitektural, atau dapat disimpulkan secara tidak langsung melalui analisis saluran samping (*side-channel analysis*).

Dalam kasus lain optimasi performa memungkinkan kode yang berjalan pada tingkat hak akses CPU tertentu untuk mengakses, menyimpulkan, atau memengaruhi kode maupun data milik tingkat hak akses yang lebih tinggi. Perilaku cache menjadi faktor penyebab yang cukup umum; cache yang digunakan bersama memungkinkan serangan seperti Flush+Reload dan Prime+Probe teknik yang dapat membocorkan informasi lintas lingkungan eksekusi yang seharusnya terisolasi satu sama lain.

Kerentanan mikroarsitektur menjadi sangat relevan ketika penyerang telah berhasil mengeksekusi kode pada platform komputasi berkinerja tinggi (*high-performance compute*/HPC) dan berupaya memperluas dampaknya dengan mengamati atau memengaruhi konteks eksekusi lainnya. Serangan jenis Spectre dan Meltdown sering kali diperkenalkan sebagai teknik pengintaian (*reconnaissance*), namun dalam skenario pasca-kompromi, dampaknya menjadi lebih langsung. Serangan-serangan ini dapat memintas pemisahan hak akses (*privilege separation*) yang diterapkan oleh CPU, sehingga memungkinkan kode yang tidak memiliki hak akses khusus untuk membaca atau menyimpulkan data milik konteks

dengan hak akses lebih tinggi seperti kernel atau, pada arsitektur dengan cache atau antarmuka memori bersama, mengakses lingkungan eksekusi tepercaya (*trusted execution environments*) maupun koprosesor aman di sekitarnya.

Meltdown menggambarkan risiko ini dengan jelas. Dengan mengeksploitasi eksekusi di luar urutan (*out-of-order execution*), penyerang dapat memuat memori kernel yang memiliki hak akses khusus ke dalam status register mikroarsitektur yang seharusnya tidak dapat diakses oleh kode tanpa hak akses khusus kemudian memulihkan data tersebut menggunakan cache side channel setelah CPU membatalkan operasi ilegal tersebut. Meskipun Meltdown tidak memberikan kemampuan untuk mengeksekusi kode di tingkat kernel atau peningkatan hak akses (*privilege escalation*) yang sesungguhnya, serangan ini menyediakan sarana ampuh untuk membocorkan informasi dengan memintas batasan hak akses: penyerang mendapatkan visibilitas ke dalam memori kernel tanpa perlu mengeksploitasi satu pun cacat perangkat lunak dalam sistem operasi, melainkan hanya mengandalkan kelemahan pada desain CPU. Hal ini menjadi sangat relevan bagi sistem HPC, di mana mesin virtual (*virtual machine*/VM) dengan tingkat kritikalitas yang beragam, serta aplikasi di dalam VM tersebut, sangat bergantung pada isolasi VM dan proses untuk memenuhi persyaratan keamanan dan keselamatan.

Kerentanan Terhadap Injeksi Kesalahan (Fault Injection)

Sebagaimana dibahas sebelumnya dalam konteks ECU berbasis MCU, serangan injeksi kesalahan dapat sangat efektif untuk memintas mekanisme *secure boot* (boot aman), membuka antarmuka debug, atau mengekstrak rahasia kriptografis. Pada platform HPC (*High-Performance Computing*), permukaan serangan untuk injeksi kesalahan sering kali diperumit oleh kompleksitas perangkat keras yang lebih tinggi, rantai boot yang lebih panjang dan terfragmentasi, serta keberadaan berbagai subsistem debug. HPC modern mungkin memiliki beberapa infrastruktur debug independen, seperti ARM CoreSight atau modul debug RISC-V, yang masing-masing terkait dengan domain hak akses dan lingkungan eksekusi *runtime* yang berbeda. Keberagaman ini menambah jumlah pemeriksaan keamanan dan transisi status yang berpotensi menjadi target bagi kesalahan yang disuntikkan dengan pengaturan waktu yang presisi.

Dengan interaksi terbatas terhadap HPC seperti kemampuan untuk mengeluarkan perintah, memengaruhi input, atau menjalankan payload tingkat pengguna penyerang dapat menyinkronkan injeksi kesalahan dengan operasi-operasi yang krusial bagi keamanan. Salah satu target yang sangat menarik adalah proses *secure boot*, baik karena nilai pentingnya maupun karena relatif mudahnya melakukan gangguan (*glitching*) selama tahap awal boot, saat hanya satu atau beberapa inti (*core*) yang aktif dan alur eksekusinya telah ditentukan sebelumnya. Penelitian terkini menunjukkan bahwa injeksi kesalahan yang disinkronkan dengan cermat dapat memintas verifikasi *secure boot* dan menyebabkan platform memuat image yang telah disusupi.

Begitu penyerang berhasil memuat *firmware* berbahaya selama proses boot, mereka akan mendapatkan tingkat hak akses tertinggi yang tersedia pada HPC tersebut. Pada titik ini, jaminan keamanan yang ditegakkan pada lapisan perangkat lunak yang lebih tinggi menjadi tidak berlaku. Penyerang dapat menanamkan *backdoor* yang persisten, mengonfigurasi ulang

mekanisme isolasi, mengaktifkan fungsi debug tersembunyi, atau melemahkan perlindungan kriptografis.

Kelemahan ROM

Sebagaimana dibahas dalam bagian sebelumnya, perangkat lunak yang terlibat dalam penerapan *secure boot* (proses booting yang aman) merupakan target yang menarik untuk dieksploitasi. Hal ini juga berlaku untuk HPC (*High-Performance Computing*), dengan perbedaan utama terletak pada kompleksitas dan keragaman tumpukan *bootloader* yang lebih tinggi. Dalam arsitektur HPC, satu atau lebih komponen ROM yang tidak dapat diubah (*immutable*) biasanya menginisiasi proses booting untuk berbagai lingkungan eksekusi runtime, seperti CPU dan GPU aplikasi. ROM ini memuat *firmware* yang dapat diubah sering kali di dalam lingkungan eksekusi tepercaya (*Trusted Execution Environment* atau TEE) seperti HPSE yang kemudian bertanggung jawab untuk melakukan autentikasi dan memuat perangkat lunak di berbagai lingkungan eksekusi lainnya, termasuk *firmware* tingkat rendah, *safety island* (area komputasi yang aman untuk fungsi kritis), dan klaster komputasi tujuan umum. Mengingat banyaknya jumlah dan heterogenitas komponen-komponen tersebut, proses *secure boot* sering kali dijalankan secara berurutan (*pipelined*) atau paralel untuk mempersingkat waktu *booting* secara keseluruhan. Meskipun langkah ini meningkatkan kinerja saat memulai sistem, hal tersebut juga menambah jumlah batasan kepercayaan (*trust boundaries*) dan status perantara yang harus diamankan.

Salah satu kelemahan mencolok dalam desain semacam ini adalah tidak adanya dukungan untuk mencabut (*revocation*) kunci publik akar (*root public key*) yang digunakan selama verifikasi tanda tangan digital. Karena kunci akar ini biasanya tertanam di dalam ROM yang tidak dapat diubah atau di dalam *fuse* (sekering elektronik), maka jika kunci privat yang bersesuaian berhasil dikompromikan misalnya melalui pembobolan infrastruktur *backend* atau sistem penandatanganan penyerang dapat membuat citra *firmware* yang akan terus dianggap sah untuk selamanya. Dalam kasus seperti ini, perangkat yang terdampak mungkin tidak dapat dipulihkan hanya melalui pembaruan perangkat lunak, karena jangkar kepercayaan (*trust anchor*) itu sendiri tidak dapat diganti.

Kelemahan lain yang berkaitan dengan perangkat keras muncul dari kerentanan dalam kode ROM itu sendiri. Konten ROM bersifat tidak dapat diubah setelah proses manufaktur, sehingga tidak dapat diperbaiki (*patched*) melalui pembaruan perangkat lunak konvensional. Ambiguitas mengenai apakah ROM harus diperlakukan sebagai perangkat keras atau perangkat lunak sering kali menyebabkan penerapan praktik pengodean yang aman menjadi tidak konsisten, analisis statis yang terbatas, serta tinjauan keamanan yang tidak memadai selama tahap pengembangan.

Meskipun beberapa SoC menyediakan mekanisme terbatas untuk memperbaiki ROM menggunakan memori OTP (*One-Time Programmable*) atau jenis memori serupa, mekanisme ini memiliki keterbatasan dalam hal kapasitas penyimpanan dan jumlah perbaikan yang dapat ditampung. Akibatnya, kerentanan dalam kode ROM cenderung bertahan sepanjang masa pakai produk tersebut, sehingga validasi keamanan sejak dini terhadap implementasi ROM menjadi sangat penting.

Kelemahan Perangkat Lunak

Kelemahan perangkat lunak pada HPC (*High-Performance Computer*) sebagian besar bersumber dari tumpukan proses boot (*boot stack*) yang kompleks, lingkungan tervirtualisasi, dan sistem operasi yang kaya fitur. Karena arsitektur MCU (*Microcontroller Unit*) biasanya tertanam di dalam HPC, sistem ini juga mewarisi kelemahan perangkat lunak yang telah dibahas pada bab sebelumnya. Di bagian ini, kita akan menelusuri berbagai kelemahan perangkat lunak yang umum terjadi.

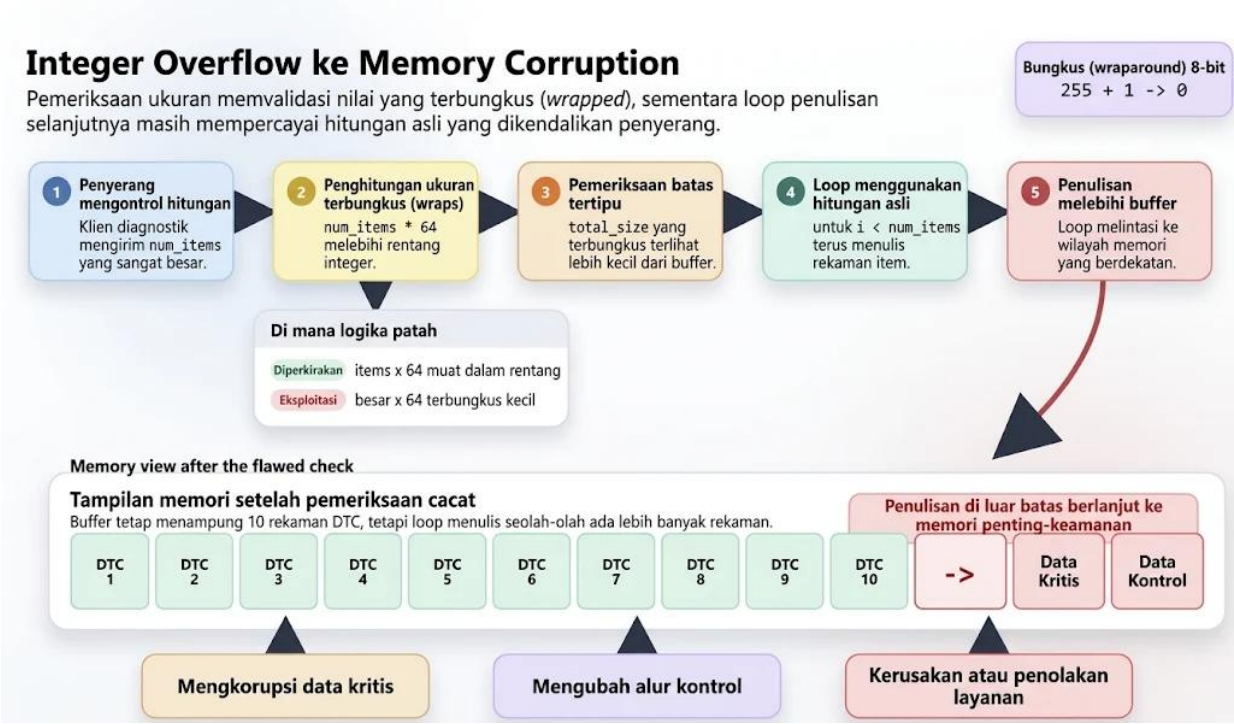
Cacat Implementasi Dasar

Sebelum membahas kelemahan yang spesifik pada perangkat lunak, penting untuk memahami cacat implementasi perangkat lunak yang mendasar. Kelemahan-kelemahan ini sering dieksploitasi untuk mendapatkan akses awal dan membangun rangkaian serangan (*attack paths*) di seluruh sistem yang kompleks. Setiap tahun, MITRE menerbitkan daftar peringkat kategori kelemahan yang paling sering ditemukan dalam kerentanan yang dilaporkan.

Dua jenis cacat implementasi yang secara konsisten muncul di urutan teratas daftar tersebut adalah *buffer overflow* (peluapan penyangga) dan *integer overflow* (peluapan bilangan bulat). Kelemahan-kelemahan ini bukanlah hal yang asing; fenomena ini sudah dipahami dengan baik dan didokumentasikan secara luas, namun tetap saja muncul dalam perangkat lunak otomotif kelas produksi. Untuk mengilustrasikan cacat implementasi perangkat lunak, kita akan mempelajari kelemahan integer overflow yang dapat berujung pada *buffer overflow*. Anda disarankan untuk mempelajari lebih dalam mengenai kelemahan lain yang tercantum dalam daftar "Top 25", karena konsep-konsep dasarnya sering muncul berulang kali di hampir setiap lapisan tumpukan perangkat lunak HPC.

Memahami Integer Overflow dan Buffer Overflow

Integer overflow terjadi ketika suatu operasi aritmetika menghasilkan nilai numerik yang melampaui batas maksimum yang dapat direpresentasikan oleh tipe data tertentu. Sebagai contoh, sebuah integer 8-bit tanpa tanda (*unsigned integer*) hanya dapat merepresentasikan nilai dari 0 hingga 255. Jika kita mencoba menambahkan 1 ke angka 255, hasilnya akan kembali ke 0 (bukan 256) karena keterbatasan rentang nilai pada tipe data tersebut.



Gambar 8.3: Ilustrasi kerusakan memori akibat cacat implementasi integer *overflow*

Perhatikan sebuah program yang dirancang untuk menghitung memori yang diperlukan guna menyimpan beberapa *Diagnostic Trouble Code* (DTC) ke dalam memori *flash* data, sebagaimana ditunjukkan pada Gambar K.3. Setiap data DTC menempati ruang sebesar 64 byte, dan program tersebut meminta klien diagnostik untuk menentukan jumlah data yang ingin disimpan. Selanjutnya, program akan menghitung total ukuran yang dibutuhkan dan mengalokasikan memori yang diperlukan.

Berikut adalah cuplikan kode sederhana yang menunjukkan bagaimana cacat tersebut mengakibatkan kerusakan memori:

```
#define ITEM_SIZE 64
#define MAX_ITEMS 10
#define BUFFER_SIZE (ITEM_SIZE * MAX_ITEMS)

// Buffer for storing item data
char buffer[BUFFER_SIZE];

int main() {
    unsigned int num_items;
    printf("Enter the number of items: ");
scanf("%u",#_items);

    // Integer overflow vulnerability
    unsigned int total_size = num_items * ITEM_SIZE;

    // Check to avoid writing beyond the buffer - but the check is flawed due to
integer overflow
    if (total_size > BUFFER_SIZE) {
        printf("Too many items!\n");
```

```
        return -1;
    }

    // Process items - this is where the overflow can be exploited
    for (unsigned int i = 0; i < num_items; i++) {
        // Simulating processing of each item and filling the buffer
        // Vulnerable code: writes data beyond the buffer if num_items is too
large due to overflow
        printf("Processing item %u\n", i);
memset(&buffer[i*ITEM_SIZE],0,ITEM_SIZE); // Fill item slot with zeroes
    }

    return 0;
}
```

Dalam kode ini, seorang penyerang dapat memasukkan angka yang sangat besar untuk `num_items`. Jika `num_items` cukup besar, perkalian `num_items ITEM_SIZE` dapat menyebabkan integer *overflow* (luapan bilangan bulat), yang mengakibatkan nilai `total_size` menjadi lebih kecil dari yang seharusnya. Akibatnya, pemeriksaan `if (total_size > BUFFER_SIZE)` gagal mendeteksi *buffer overflow* (luapan penyangga), sehingga program tetap berjalan dengan ruang penyangga yang tidak memadai.

Eksploitasi

Dengan memanfaatkan *integer overflow* ini, penyerang dapat membuat program menulis data di luar batas penyangga yang telah dialokasikan. Di dalam perulangan `for`, program menulis data ke dalam penyangga berdasarkan nilai `num_items` yang diberikan pengguna, yang berpotensi menimpa wilayah memori di sekitarnya. Inilah yang dikenal sebagai *buffer overflow*, yang sering digunakan untuk menyisipkan payload berbahaya ke dalam memori serta meluapi stack atau heap. Penulisan di luar batas (*out-of-bounds write*) ini dapat menimbulkan berbagai dampak buruk, antara lain:

- **Merusak data kritis**: Menimpa variabel penting atau struktur kontrol, yang menyebabkan perilaku sistem menjadi tidak terduga.
- **Mengubah alur kontrol (*control flow*)**: Memodifikasi penunjuk fungsi (*function pointer*) atau alamat kembalian (*return address*), yang berpotensi memungkinkan eksekusi kode yang tidak diinginkan.
- **Menyebabkan sistem crash**: Memicu pelanggaran akses (*access violation*) yang berujung pada penolakan layanan (*denial of service*).

Selain kelemahan di atas, kurangnya validasi input yang memadai serta adanya *race condition* (kondisi balapan) tetap menjadi celah yang dapat dimanfaatkan untuk menyisipkan kode ke dalam memori dan menyalahgunakan layanan yang memiliki hak akses khusus (*privileged services*). Memahami setiap jenis kelemahan dan cara mengeksploitasinya merupakan keterampilan wajib bagi seorang penetration tester yang ingin merangkai berbagai kelemahan untuk melancarkan serangan yang lebih canggih.

Bootloader dan Konfigurasi Boot Sistem Operasi yang Tidak Aman

Kelemahan dalam desain atau implementasi komponen apa pun dalam rangkaian proses boot (*boot stack*) dapat memutus rantai kepercayaan (*chain of trust*) dan

mengakibatkan dimuatnya kode berbahaya saat sistem dinyalakan. Salah satu kelemahan tersebut adalah tidak adanya perlindungan *read-only* (hanya baca) pada wilayah memori yang diinisialisasi oleh komponen boot tahap awal, yang tetap dapat ditulis (*writable*) bahkan setelah kendali dialihkan ke tahap berikutnya.

Hal ini menciptakan celah bagi penyerang yang memiliki akses terbatas ke memori selama proses boot awal untuk melakukan korupsi memori. Demikian pula, jika suatu komponen boot gagal membatasi akses periferal secara tepat kepada bus master atau lingkungan eksekusi yang dituju, sistem dapat beralih ke status operasional dengan konfigurasi perangkat keras yang tidak aman.

Pada banyak sistem, *bootloader* seperti U-Boot, UEFI, atau loader tahap kedua khusus bertanggung jawab untuk memuat sistem operasi tamu (*Guest OS*), misalnya Linux atau Android. Untuk mendukung pengembangan dan pengujian, banyak perangkat termasuk *High-Performance Computer* (HPC) pada kendaraan mengizinkan mekanisme *secure boot* untuk dilewati (*bypassed*) selama perangkat tersebut masih dalam tahap pra-produksi. Jika penyerang mendapatkan akses fisik ke HPC dan *bootloader* dibiarkan dalam kondisi tidak terkunci (atau dapat dibuka tanpa otorisasi yang sah), mereka dapat memuat sistem operasi tamu yang telah dimodifikasi. Tindakan ini secara efektif melangkahi mekanisme keamanan tingkat lebih tinggi yang diterapkan oleh sistem operasi asli. Begitu penyerang mampu mengeksekusi kode arbitrer selama proses boot, mereka dapat memasang modifikasi persisten, seperti bootkit atau rootkit, yang tetap ada bahkan setelah pembaruan perangkat lunak normal dilakukan.

Bootloader juga rentan terhadap cacat implementasi umum serta upaya untuk melewati logika perlindungannya melalui teknik *fault-injection* (injeksi kesalahan). Penelitian mengenai *bootloader* pada sistem tertanam (*embedded systems*) menunjukkan bahwa pola desain dan implementasi yang rentan dapat dieksploitasi melalui analisis biner dan teknik *glitching* untuk melumpuhkan mekanisme keamanan *bootloader* dan mengambil alih kendali atas memori yang terlindungi.

Serangan yang tidak terlalu canggih namun memiliki dampak yang sama besarnya dapat terjadi akibat kesalahan konfigurasi dalam rangkaian proses boot. Sebagai contoh, jika kunci publik akar (*root public key*) yang digunakan untuk memverifikasi tanda tangan digital selama proses boot secara keliru menggunakan kunci pengembangan (*development key*) alih-alih kunci produksi (*production key*), penyerang mungkin bisa mendapatkan kunci privat yang bersesuaian yang sering kali disebarluaskan secara luas untuk keperluan pengembangan dan pengujian. Akibatnya, citra (*image*) *firmware* yang tidak sah dapat diterima sebagai sesuatu yang sah. Oleh karena itu, penguji penetrasi (*penetration tester*) biasanya ditugaskan untuk meninjau perangkat lunak boot serta proses penyediaan kunci (*provisioning*) guna mengidentifikasi kelemahan semacam itu sebelum produk dipasarkan.

Bahkan ketika rantai kepercayaan (*chain of trust*) tetap utuh, kegagalan dalam menginisialisasi sistem ke dalam status runtime yang aman merupakan kelemahan lain yang kerap muncul pada platform HPC (*High-Performance Computing*). Banyak asumsi keamanan bergantung pada konfigurasi statis terkait partisi perangkat keras, wilayah memori, dan kontrol

akses periferal selama tahap awal proses booting. Konfigurasi yang terlalu longgar dapat membuat sistem rentan jika aplikasi pada tahap selanjutnya mengalami kompromi akibat kerentanan perangkat lunak.

Terakhir, penerapan *secure boot* (proses booting yang aman) yang tidak memadai serta kelemahan pada hardware trust anchor (jangkar kepercayaan perangkat keras) tetap menjadi masalah kritis dalam skenario pasca-kompromi. *Secure boot* dirancang untuk memastikan bahwa hanya *firmware* yang ditandatangani oleh produsen kendaraan atau pemasok ECU (*Electronic Control Unit*) yang dapat tetap ada setelah sistem dimulai ulang (*reboot*). Namun, cakupan *secure boot* yang tidak lengkap dapat membatalkan jaminan tersebut. Pada beberapa desain, tahap awal booting memang diverifikasi, tetapi komponen selanjutnya seperti sistem operasi atau citra aplikasi dimuat tanpa verifikasi tanda tangan, sering kali karena pertimbangan waktu mulai (*startup timing*) atau performa. Penyerang dapat menargetkan tahap-tahap yang tidak diverifikasi ini untuk memasang perangkat lunak berbahaya (*malware*) yang bersifat persisten (menetap).

Layanan yang Tidak Diperlukan dan Perluasan Permukaan Serangan (Attack Surface)

HPC sering kali menyertakan berbagai layanan, daemon, antarmuka debug, dan paket sistem operasi yang sebenarnya tidak dibutuhkan untuk fungsi utama kendaraan. Setiap komponen yang tidak diperlukan akan memperluas permukaan serangan dengan cara membuka port, menangani input eksternal, mengekspos operasi dengan hak akses khusus (*privileged operations*), atau menyediakan fungsi pengembangan yang seharusnya tidak diaktifkan pada tahap produksi.

Contoh umum mencakup server web bawaan, layanan SSH atau Telnet yang sudah usang, aplikasi contoh, kode demo, layanan pemeliharaan khusus vendor, serta daemon untuk keperluan manufaktur atau pengembangan. Setiap layanan tersebut mungkin memiliki kerentanan sendiri, autentikasi yang lemah, pengaturan bawaan yang tidak aman, atau hak akses yang berlebihan. Oleh karena itu, penyerang maupun penguji keamanan (*red-team testers*) akan secara sistematis mendata port yang terbuka, proses yang berjalan, paket yang terinstal, serta antarmuka manajemen yang terekspos pada HPC untuk mengidentifikasi titik masuk yang lemah.

Antarmuka debug dan konsol memerlukan perhatian khusus karena sering kali memberikan akses yang luas dengan perlindungan yang lebih minim dibandingkan layanan untuk tahap produksi. *Shell prompt* yang terekspos, konsol UART yang tidak diamankan, layanan Telnet yang terbuka, atau titik akhir debug dengan perlindungan lemah dapat memberikan akses langsung kepada penyerang untuk berinteraksi dengan sistem operasi. Demikian pula, antarmuka debug pada Linux seperti `/sys/kernel/debug` atau `debugfs`—dapat mengekspos kontrol tingkat kernel yang berbahaya jika izin akses (*permissions*) dikonfigurasi dengan keliru. Sebagai contoh, CVE-2019-14100 pada platform Snapdragon Automotive memungkinkan penulisan register melalui debugfs; pihak vendor kemudian menonaktifkan penulisan register melalui debugfs secara default untuk mengatasi masalah tersebut.

Layanan Dengan Hak Akses Berlebih (Overprivileged) dan Kesalahan Konfigurasi Izin

Prinsip hak akses minimum (*least privilege*) merupakan prinsip keamanan yang mendasar. Ketika komponen perangkat lunak HPC dijalankan dengan hak akses yang melebihi kebutuhan fungsinya, hal ini menciptakan peluang bagi penyerang untuk memperbesar dampak dari kompromi awal. Dalam praktiknya, sering ditemukan daemon, layanan, atau aplikasi pada sistem HPC yang berjalan dengan hak akses root atau tingkat sistem tanpa adanya pembenaran fungsional yang kuat.

Sebagai contoh, mengeksploitasi *buffer overflow* pada pemutar media *infotainment* mungkin hanya memungkinkan penyerang untuk membuat proses tersebut mengalami crash atau mengendalikannya. Namun, jika pemutar media tersebut berjalan dengan hak akses root, kerentanan yang sama dapat memberikan akses tingkat root di dalam VM (*virtual machine*) tamu. Jika aplikasi yang bersifat kritis terhadap keselamatan (*safety-critical*) atau layanan sensitif berbagi VM yang sama, dampak yang ditimbulkan oleh penyerang dapat meluas secara signifikan. Oleh karena itu, penguji penetrasi (*penetration tester*) sering menganggap layanan dengan hak akses tinggi sebagai target bernilai tinggi.

Pengaturan izin (*permission*) berkas dan direktori yang tidak tepat menciptakan risiko serupa. Linux dan Android sangat bergantung pada *discretionary access control* (DAC), dan dalam lingkungan pengembangan HPC (*High-Performance Computing*) yang terdistribusi, pengaturan izin dapat dengan mudah menjadi tidak konsisten. Akses baca tanpa batasan, izin tulis yang tidak diperlukan, atau akses eksekusi pada berkas sensitif dapat memfasilitasi pergerakan lateral (*lateral movement*) atau eskalasi hak akses setelah terjadi kompromi lokal. Sebagai contoh, berkas log yang dapat ditulis oleh siapa saja, data kalibrasi, direktori konfigurasi sistem seperti `/etc`, atau direktori aplikasi dengan izin `0777` dapat memungkinkan penyerang untuk mengubah perilaku sistem, menanam berkas yang dapat dieksekusi, atau menyisipkan konten yang nantinya akan diproses oleh layanan dengan hak akses tinggi.

Artefak pengembangan yang tertinggal merupakan jalur eskalasi umum lainnya. Citra (*image*) sistem HPC untuk lingkungan produksi mungkin secara tidak sengaja menyertakan alat debugging, binary pembantu, skrip, atau utilitas pengujian yang memiliki izin `setuid-root` atau daftar kontrol akses (*access control list*) yang terlalu longgar. Penyerang secara aktif mencari artefak-artefak ini karena dapat memberikan jalan langsung menuju hak akses yang lebih tinggi. Mekanisme *mandatory access control* (MAC) dirancang untuk mengurangi risiko ini dengan menerapkan isolasi yang lebih ketat antara proses dan sumber daya. Sistem HPC berbasis Linux dapat menggunakan SELinux yang juga umum digunakan dalam Android atau AppArmor untuk membatasi apa saja yang dapat diakses oleh setiap proses. Sebagai contoh, kebijakan SELinux dapat mencegah aplikasi infotainment membaca berkas konfigurasi sistem atau menghentikan layanan Bluetooth agar tidak mengakses data sensor, kecuali melalui antarmuka yang telah disetujui.

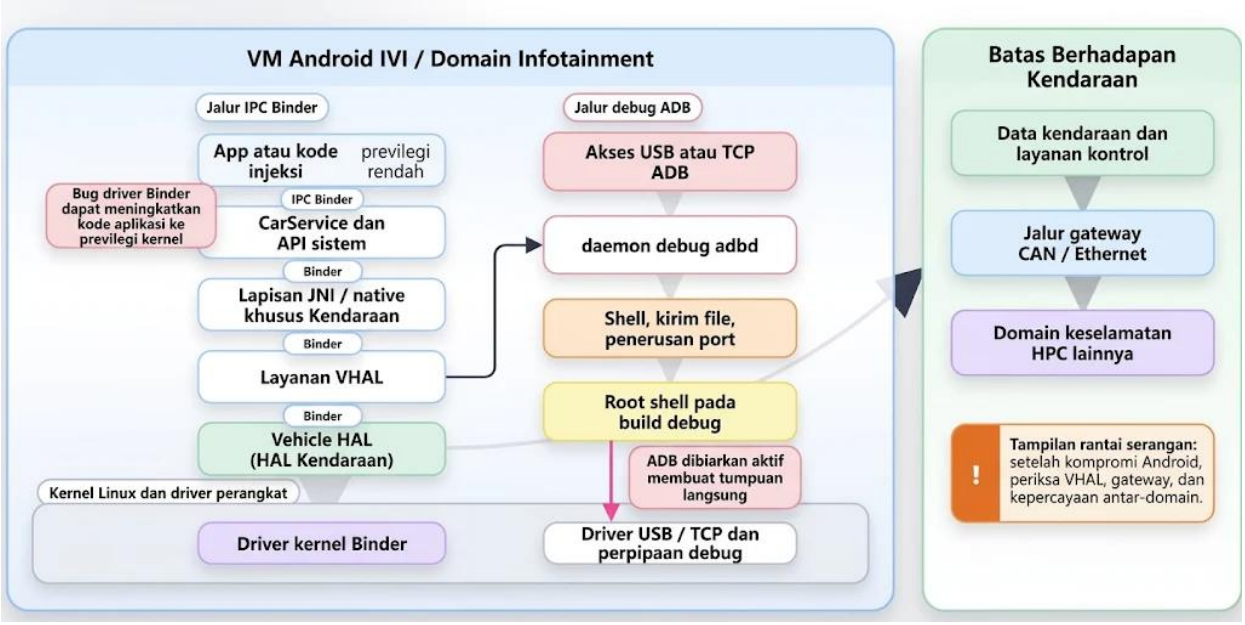
Namun dalam praktiknya, perlindungan MAC sering kali dikonfigurasi dengan keliru, terlalu longgar, atau bahkan dinonaktifkan demi kemudahan pengembangan. Sebagai contoh, mode permissive pada SELinux hanya mencatat pelanggaran kebijakan tanpa benar-benar

menegakkannya, sehingga menghilangkan lapisan pertahanan yang penting. Risiko serupa muncul ketika build pengembangan atau build untuk debugging pengguna (*userdebug*) diterapkan tanpa pengamanan yang memadai, karena build semacam ini memungkinkan penegakan MAC untuk dinonaktifkan atau dilewati dengan lebih mudah.

Kerentanan Pada Antarmuka IPC dan Debug yang Spesifik Untuk Android

Android, yang merupakan sistem operasi pilihan untuk sistem infotainment, memiliki kelemahan perangkat lunak tersendiri yang dapat dieksploitasi oleh penyerang, khususnya pada fitur komunikasi antarproses (*inter-process communication*/IPC) dan fitur debugging-nya. Salah satu vektor yang patut diperhatikan adalah mekanisme IPC Android Binder. Binder merupakan driver kernel sekaligus kerangka kerja yang memungkinkan layanan sistem tingkat tinggi dan aplikasi untuk saling berkomunikasi.

Mekanisme ini memegang peranan sentral dalam arsitektur Android karena hampir semua API sistem yang memiliki hak akses khusus (*privileged system APIs*) seperti layanan telepon, media, dan sebagainya disediakan melalui Binder. Akibatnya, setiap bug kernel pada driver Binder dapat menimbulkan konsekuensi yang serius.



Gambar 8.4: Binder mengekspos layanan kendaraan, sementara ADB mengekspos akses shell langsung

Contoh yang menonjol adalah CVE-2019-2215, sebuah kerentanan *use-after-free* pada driver Binder. Celah ini memungkinkan peningkatan hak akses (*privilege escalation*) dari aplikasi ke kernel Linux tanpa interaksi pengguna. Pada akhir 2019, bug Binder ini dieksploitasi di lapangan sebagai *zero-day* (terutama digunakan oleh perangkat mata-mata/ spyware untuk keluar dari sandbox Android).

Akibatnya, jika penyerang berhasil menginstal atau menyuntikkan aplikasi berbahaya ke dalam sistem IVI (*In-Vehicle Infotainment*) berbasis Android (melalui rekayasa sosial, kompromi rantai pasok, atau eksploitasi tahap awal lainnya), kerentanan Binder seperti CVE-

2019-2215 dapat memungkinkan aplikasi tersebut mengambil alih domain Android di dalam HPC (*High-Performance Computer*) dan melakukan gangguan berbahaya terhadap kendaraan melalui HAL (*Hardware Abstraction Layer*) kendaraan. Bahkan tanpa instalasi aplikasi, Binder dapat diakses oleh kode apa pun yang sudah berjalan di perangkat; dengan demikian, pijakan yang diperoleh melalui layanan lain dapat ditingkatkan (*escalated*) dengan memanfaatkan bug kernel semacam itu. Penguji penetrasi (*pen tester*) disarankan untuk mencoba eksploitasi Android yang sudah diketahui terhadap unit IVI otomotif, karena perangkat tersebut pada dasarnya adalah perangkat Android. Setelah lingkungan Android berhasil dikompromi, penting untuk menguji seberapa tangguh domain HPC lainnya terhadap gangguan berbahaya yang berasal dari lingkungan Android tersebut.

Seperti yang telah disebutkan sebelumnya terkait Linux, antarmuka debug dan pengembang dapat menjadi sarana yang sangat efektif untuk membangun pijakan dalam sistem; hal yang sama juga berlaku untuk Android. Pada Android, antarmuka debug utama adalah ADB (*Android Debug Bridge*). ADB memungkinkan pengembang untuk terhubung ke perangkat (biasanya melalui USB atau TCP) guna mendapatkan akses *root shell*, mengunggah berkas, dan sebagainya, terutama pada *build* pengembangan.

Dalam versi produksi, ADB biasanya dinonaktifkan atau memerlukan persetujuan serta pemasangan (*pairing*) dari pengguna. Oleh karena itu, sebagai penguji penetrasi, Anda harus selalu memeriksa apakah ADB secara tidak sengaja tertinggal dalam kondisi aktif. Koneksi ADB yang terbuka (misalnya, melalui port USB di dalam mobil atau ADB via Wi-Fi yang dibiarkan aktif) dapat memungkinkan penyerang yang memiliki akses fisik untuk langsung membuka shell pada HPC. Jika perangkat menjalankan build debug (yang sering terjadi pada tahap pra-rilis atau jika vendor lupa beralih ke *build* rilis), shell ADB tersebut mungkin akan berjalan sebagai root secara bawaan.

Kelemahan Firmware HPSE

Sebagaimana dibahas dalam Chauícu T, *Hardware Protected Security Environment* (HPSE) khusus dirancang untuk mengurangi risiko terungkapnya kunci kriptografis dengan menyimpan kunci secara aman dan menjalankan operasi sensitif di dalam lingkungan eksekusi yang terlindungi. Namun, berbeda dengan arsitektur berbasis MCU tradisional, HPSE dalam *High-Performance Computer* (HPC) sering kali harus melayani beberapa penyewa (*tenant*) yang saling tidak percaya. Akibatnya, kelemahan dalam cara VM tamu (*guest VM*) mengakses layanan HPSE dapat menyebabkan materi kriptografis milik satu penyewa terekspos ke penyewa lain. Kelemahan *firmware* HPSE paling sering muncul dalam logika pengendalian akses dan penanganan kunci. Jika *firmware* tidak menerapkan isolasi dengan benar di antara entitas yang mengajukan permintaan, kunci dapat diakses di luar cakupan yang seharusnya. Dengan demikian, keamanan HPSE tidak hanya bergantung pada perlindungan perangkat keras, tetapi juga pada kebenaran *firmware* yang menjadi perantara akses ke layanan kriptografis.

Seperti yang telah disebutkan sebelumnya, keberadaan HPSE tidak menjamin bahwa semua operasi kriptografis dilakukan di dalamnya. Pemasok komponen otomotif mungkin masih mengandalkan implementasi kriptografi berbasis perangkat lunak demi alasan kinerja

atau integrasi, terutama pada HPC yang menuntut *throughput* tinggi. Hal ini memunculkan kembali risiko terungkapnya kunci kriptografis melalui serangan terhadap memori sistem utama, termasuk melalui pemeriksaan memori fisik (*physical memory probing*) atau serangan berbasis DMA (*Direct Memory Access*). Dampak kinerja dari pengalihan seluruh operasi kriptografis melalui HPSE sering kali lebih terasa pada arsitektur HPC, sehingga meningkatkan kemungkinan digunakannya kriptografi perangkat lunak untuk melengkapi atau bahkan melewati perangkat keras yang aman yang pada akhirnya melemahkan postur keamanan secara keseluruhan.

Alasan lain mengapa kriptografi perangkat lunak masih terus digunakan adalah munculnya algoritma kriptografi baru yang mungkin belum didukung oleh perangkat keras, seperti algoritma PQC (*Post-Quantum Cryptography*) atau algoritma kriptografi standar Tiongkok. Ketika solusi kriptografi perangkat lunak tidak dapat dihindari, implementasi yang mengabaikan analisis side-channel berbasis waktu (*timing side-channel analysis*) dan eksekusi spekulatif (*speculative execution*) menjadi sangat rentan terhadap serangan yang bertujuan untuk mengungkap kunci kriptografis.

Isolasi Antar-VM yang Lemah dan Berbagi Sumber Daya

Selain skenario di mana VM tamu berhasil keluar sepenuhnya ke sistem *host*, terdapat jenis kelemahan yang lebih umum, yakni isolasi yang tidak lengkap atau tidak diterapkan dengan benar di antara mesin virtual (*virtual machine*). *Hypervisor* bertanggung jawab untuk memastikan pemisahan yang ketat terhadap memori, I/O, dan sumber daya eksekusi, sehingga satu VM tidak dapat mengganggu VM lainnya. Jika isolasi ini dikonfigurasi secara keliru, VM dengan tingkat kepercayaan rendah yang telah disusupi dapat berdampak langsung pada beban kerja yang memiliki tingkat kepentingan lebih tinggi.

Persyaratan mendasarnya adalah sebuah *guest* (sistem operasi tamu) tidak boleh dapat mengakses memori milik *hypervisor* maupun guest lainnya. Pemisahan ini biasanya ditegakkan melalui ekstensi virtualisasi perangkat keras dan MMU (*Memory Management Unit*). Akan tetapi, kesalahan konfigurasi atau jalur komunikasi memori bersama (*shared-memory*) yang tidak dibatasi dengan ketat dapat merusak jaminan tersebut. Sebagai contoh, ketika guest saling bertukar data melalui mekanisme memori bersama, model partisi harus membatasi secara ketat wilayah mana saja yang boleh dibagikan dan bagaimana mekanisme pemberian sinyal dilakukan. Bahkan tanpa adanya kerentanan langsung, saluran terselubung (*covert channels*) dapat muncul melalui pengamatan terhadap inti CPU (*CPU cores*) dan cache yang digunakan bersama.

Dalam sistem yang krusial bagi keselamatan (*safety-critical systems*), isolasi temporal sama pentingnya dengan isolasi spasial. Jika penjadwalan CPU tidak bersifat deterministik atau tidak dialokasikan dengan tepat, VM yang telah disusupi dapat menghabiskan siklus CPU yang seharusnya diperuntukkan bagi VM yang menangani fungsi keselamatan. Dari perspektif pengujian penetrasi (*penetration testing*), mengidentifikasi kesalahan konfigurasi semacam ini memberikan cara praktis untuk mendemonstrasikan kondisi penolakan layanan (*denial-of-service*) terhadap fungsi-fungsi keselamatan sebuah masalah serius, terutama mengingat target tingkat otonomi yang lebih tinggi dalam kendaraan otonom. Hipervisor otomotif

biasanya menyediakan mekanisme untuk isolasi temporal, seperti alokasi waktu CPU atau partisi penjadwalan. Namun, kesalahan konfigurasi tetap dapat menyebabkan pelanggaran terhadap persyaratan "bebas dari interferensi" (*freedom-from-interference*) dalam domain waktu, yang mengakibatkan kegagalan pemenuhan tenggat waktu (*deadline*) untuk tugas-tugas yang krusial bagi keselamatan.

Sumber daya bersama lainnya yang dapat menjadi celah untuk dieksploitasi adalah DMA (*Direct Memory Access*). Perlindungan DMA yang tidak memadai dapat memungkinkan VM yang telah disusupi untuk memprogram perangkat *pass-through* agar menimpa memori milik hipervisor atau VM lainnya. Tanpa pembatasan akses perangkat *pass-through* DMA terhadap memori fisik *host* yang seharusnya hanya diizinkan untuk area tertentu periferal yang memiliki kemampuan *bus-mastering* dapat menjadi mekanisme efektif untuk melanggar isolasi antar-VM.

Kerentanan Kernel dan Versi Perangkat Lunak yang Usang

Penggunaan perangkat lunak sumber terbuka (*open-source software* atau OSS) yang meluas dalam HPC (*High-Performance Computing*) otomotif terutama melalui ketergantungan pada komponen siap pakai (*off-the-shelf*) seperti kernel Linux atau Android serta pustaka sistem umum menciptakan permukaan serangan (*attack surface*) yang besar dan sudah dikenal luas. Ketika komponen-komponen tersebut diintegrasikan tanpa penilaian kerentanan yang ketat dan strategi pengelolaan *patch* (perbaikan perangkat lunak) yang baik, hal ini memberikan peluang eksploitasi yang siap pakai bagi penyerang.

Dalam praktiknya, memindai binary yang diperoleh dari sistem untuk mencari kerentanan yang sudah diketahui pada komponen OSS tetap menjadi salah satu teknik paling efektif yang digunakan oleh penguji penetrasi untuk mengidentifikasi jalur serangan yang mungkin dilakukan. Celah-celah ini sering kali dapat dieksploitasi karena penggunaan versi kernel yang tetap atau versi jangka panjang (*long-term*) yang tertinggal dari *patch* keamanan terbaru dari pengembang aslinya (*upstream*). Mengingat adanya kebutuhan untuk melakukan penilaian keselamatan terhadap setiap *patch*, sering kali ditemukan kerentanan yang belum diperbaiki karena pihak OEM tidak mampu menerapkan *patch* tersebut tepat waktu. Akibatnya, tidak jarang ditemukan puluhan CVE (*Common Vulnerabilities and Exposures*) yang sudah diketahui dalam tumpukan perangkat lunak HPC yang telah diterapkan, dengan eksploitasi yang tersedia untuk publik tetap dapat digunakan jauh setelah kerentanan tersebut diungkapkan. Kerentanan kernel yang terkenal, seperti Dirty COW (CVE-2016-5195) dan *Dirty Pipe* (CVE-2022-0847), menggambarkan risiko ini. *Dirty COW* merupakan *race condition* yang memengaruhi sistem operasi berbasis Linux termasuk Android dan memungkinkan eskalasi hak akses lokal hingga ke tingkat root.

Dirty Pipe, yang diungkap pada tahun 2022, memungkinkan pengguna lokal untuk mendapatkan hak akses root pada sistem Linux versi 5.8 ke atas dengan cara menyuntikkan data ke dalam berkas yang bersifat *read-only* (hanya baca) dan dimiliki oleh root. Kerentanan dalam kategori ini memungkinkan penyerang yang telah berhasil menjalankan kode secara lokal meskipun hanya sebagai pengguna atau aplikasi tanpa hak akses khusus untuk

meningkatkan hak aksesnya (*privilege escalation*) dan mengambil kendali penuh atas HPC, sehingga berhasil keluar dari lingkungan eksekusi yang terisolasi (*sandboxed environment*).

Utilitas sistem yang digunakan secara luas juga dapat mengandung kerentanan serius. Contoh nyata adalah CVE-2021-3156, yang dikenal sebagai "Baron Samedit" sebuah kerentanan *heap overflow* pada utilitas `sudo` yang tidak terdeteksi selama hampir satu dekade dan dapat dieksploitasi untuk memperoleh hak akses root. Meskipun eksploitasi memerlukan eksekusi kode awal, cacat semacam ini menunjukkan bagaimana CVE yang belum diperbaiki pada komponen standar Linux dapat langsung berujung pada eksploitasi berdampak tinggi dalam sistem otomotif.

Kerentanan Guest-To-Host Escape (Keluar Dari Guest ke Host)

Dalam HPC yang tervirtualisasi, penting untuk membedakan antara dua jenis kegagalan isolasi: *process isolation escape* (keluar dari isolasi proses) dan *guest-to-host escape* (keluar dari *guest* ke *host*). Process isolation escape terjadi ketika kode berbahaya berhasil keluar dari *sandbox* aplikasi atau batasan proses di dalam sistem operasi *guest* yang sama. Hal ini mungkin memberikan hak akses yang lebih luas kepada penyerang di dalam sistem operasi guest tersebut, namun tidak serta-merta mengompromikan *hypervisor* atau mesin virtual (VM) lainnya.

Guest-to-host escape yang terkadang disebut sebagai *GuestOS escape* atau *hypervisor escape* jauh lebih berbahaya. Dalam skenario ini, kode berbahaya yang dijalankan di dalam VM guest mengeksploitasi kerentanan pada model perangkat tervirtualisasi, antarmuka *hypercall*, mekanisme memori bersama (*shared memory*), jalur I/O virtual, atau komponen *hypervisor* untuk menembus batasan sistem operasi *guest*. Keberhasilan melakukan *escape* ini memungkinkan eksekusi kode pada tingkat hak akses *hypervisor* atau *host*, yang meruntuhkan model isolasi VM dan berpotensi memberikan akses ke VM *guest* lainnya.

Kerentanan *hypervisor escape* telah didokumentasikan dengan baik dalam dunia virtualisasi perusahaan dan memiliki dampak yang sama seriusnya dalam sistem otomotif. Sebagai contoh, pada awal tahun 2025, beberapa kerentanan dalam *hypervisor VMware* dieksploitasi secara berantai, yang memungkinkan administrator VM untuk keluar dari mesin *virtual guest* dan mengambil kendali atas *hypervisor* serta sistem *host*. Kerentanan seperti *heap overflow* memungkinkan penyerang yang memiliki hak akses lebih tinggi di dalam satu VM untuk mengeksekusi kode di dalam proses VMX *hypervisor*, yang menunjukkan bagaimana kompromi pada satu VM tamu dapat meningkat menjadi pengambilalihan sistem secara menyeluruh.

Setiap antarmuka yang disediakan oleh *hypervisor* kepada sistem operasi tamu merupakan potensi permukaan serangan (*attack surface*). Ini mencakup *hypercall*, mekanisme memori bersama (*shared memory*), dan logika emulasi perangkat virtual. Contoh terkini di sektor otomotif adalah CVE-2024-38420, sebuah kerentanan dengan tingkat keparahan tinggi pada perangkat input virtual. Dalam kasus ini, validasi yang tidak memadai terhadap data konfigurasi perangkat memungkinkan VM tamu memicu kerusakan memori (*memory corruption*) di dalam *hypervisor*. Cacat semacam ini berpotensi menyebabkan eksekusi kode pada tingkat hak akses *host*. Pada sistem komputasi berkinerja tinggi (*High-Performance*

Computing atau HPC) di kendaraan, di mana VM untuk sistem hiburan (*infotainment*) berbagi perangkat keras dengan VM untuk sistem bantuan pengemudi (*Advanced Driver Assistance Systems* atau ADAS), kerentanan serupa dapat memungkinkan aplikasi yang telah disusupi di domain yang tidak kritis terhadap keselamatan (*non-safety-critical*) untuk mengambil kendali atas domain yang kritis terhadap keselamatan (*safety-critical*).

Setelah berkas biner hipervisor diekstraksi dari HPC, analisis kerentanan terhadap perangkat lunak *hipervisor* menjadi langkah logis berikutnya. Penguji penetrasi (*penetration tester*) akan berfokus pada semua titik interaksi antara sistem tamu (*guest*) dan *hipervisor*, termasuk *hypercall*, jalur emulasi perangkat, penanganan VM exit, serta protokol memori bersama (*shared memory*). Meskipun masalah keamanan memori menjadi perhatian utama, kesalahan logika pun bisa sama berbahayanya. Sebagai contoh, *hipervisor PowerVM* dari IBM pernah memiliki celah di mana rangkaian *hypercall* yang dirancang khusus dari sistem tamu dapat menyebabkan sistem tersebut keluar dari partisinya (*escape*); hal ini menunjukkan bahwa kesalahan halus pada state machine pun dapat memungkinkan terjadinya pelarian dari sistem tamu ke sistem inang (*host*).

Pemeriksaan Autentikasi Tidak Ada Atau Lemah Selama Pembaruan

Dalam banyak desain HPC, SoC mengelola pembaruan untuk domain perangkat lunaknya sendiri serta subsistem lain yang terhubung dengannya. Jika pemeriksaan otentikasi tidak ada atau tidak lengkap, SoC yang disusupi dapat mendorong *firmware* berbahaya ke dirinya sendiri dan ECU lain yang bergantung padanya untuk menerima pembaruan perangkat lunak. Penguji penetrasi harus memberikan perhatian ekstra pada layanan pembaruan perangkat lunak di HPC untuk mencari kemungkinan penyalahgunaan pembaruan perangkat lunak untuk menginstal perangkat lunak berbahaya.

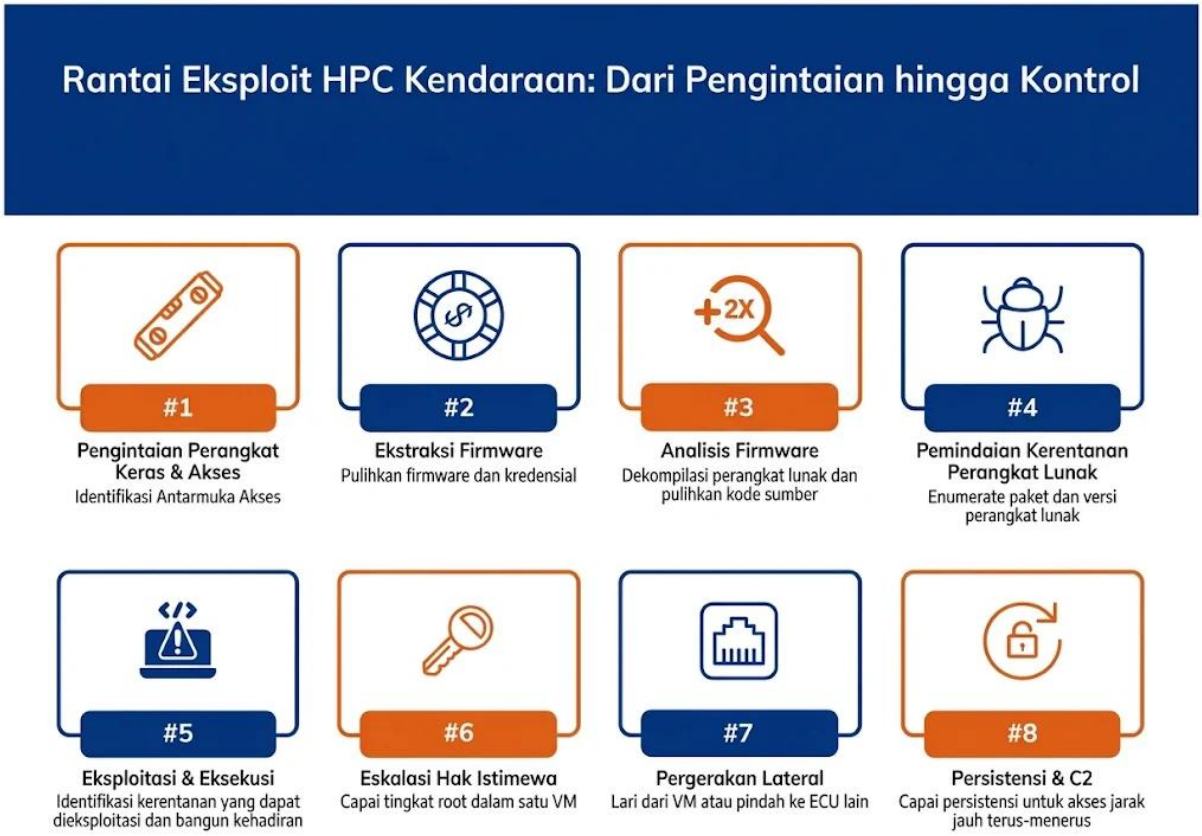
Kelemahan ini dapat menjangkau seluruh jalur komunikasi mulai dari *backend* hingga perangkat edge yang sedang diperbarui. Bidang investigasi yang umum meliputi hal-hal berikut:

- Cakupan otentikasi dan dekripsi paket pembaruan
- Masalah *Time-of-Check to Time-of-Use* (TOCTOU) antara memvalidasi keaslian blok pembaruan dan menulis gambar ke penyimpanan
- Kurangnya pemeriksaan kompatibilitas antar gambar dalam satu paket
- Kurangnya bootchain redundan untuk membantu pemulihan jika pembaruan gagal
- Kurangnya batasan pada kemampuan reboot perangkat lunak sebagai akibat dari selesainya pembaruan perangkat lunak
- Pembaca didorong untuk mempelajari kelemahan umum yang mengganggu mekanisme pembaruan perangkat lunak dengan memahami Model Ancaman Uptane

Sekarang kita sudah familiar dengan kelemahan keamanan umum yang ditemukan di HPC, sekarang saatnya menerapkan pengetahuan tersebut dan mengeksplorasi bagaimana kelemahan ini dapat disalahgunakan melalui teknik pengujian penetrasi.

8.4 MENJELAJAHI TEKNIK PENGUJIAN PENETRASI

Banyak teknik pengujian penetrasi yang disajikan untuk ECU berbasis MCU juga dapat diterapkan pada HPC. Meskipun metodenya hampir sama, terdapat beberapa taktik khusus untuk setiap jenis ECU yang akan kami soroti di sepanjang bagian ini. Meskipun HPC memiliki arsitektur perangkat keras dan perangkat lunak yang beragam, baik penyerang maupun penguji penetrasi hampir selalu mengikuti kerangka kerja yang serupa untuk menemukan kerentanan dan merancang eksploitasi, sebagaimana ditunjukkan pada Gambar 8.5.



Gambar 8.5: Alur tipikal untuk kampanye pengujian penetrasi (*pentest*) terhadap HPC

Dimulai dengan fase pengintaian (*reconnaissance*), penyerang akan mencari antarmuka yang terbuka atau tidak terlindungi untuk mengambil (*dump*) *firmware* dan mencuri data rahasia. Mengingat kompleksitas perangkat lunak serta banyaknya protokol konektivitas yang digunakan, penyerang kemudian akan mencari kerentanan perangkat lunak yang dapat dieksploitasi melalui antarmuka konektivitas tersebut untuk menyisipkan muatan berbahaya (*malicious payload*). Dalam beberapa kasus, mereka bahkan dapat mengeksploitasi kelemahan pada protokol komunikasi tanpa memerlukan adanya kerentanan perangkat lunak. Setelah itu, tahapan peningkatan hak akses (*privilege escalation*) dan pergerakan lateral (*lateral movement*) akan dilakukan, hingga penyerang mampu menjalankan kode arbitrer serta mempertahankan akses (*persistence*) guna mencapai tujuan serangan mereka. Mari kita cermati lebih dalam bagaimana penyerang dapat memanfaatkan sejumlah kelemahan yang telah dipaparkan sebelumnya untuk menyusun rangkaian serangan yang efektif.

Langkah 1: Pengintaian dan Akses Awal

Penguji penetrasi (*penetration tester*) memulai dengan meneliti perangkat keras HPC kendaraan untuk mencari celah masuk, terutama saat melakukan pengujian *black box*. Sering kali, langkah ini melibatkan pemeriksaan fisik pada papan sirkuit untuk menemukan antarmuka debug dan komunikasi, seperti port JTAG, port serial UART, serta konektor USB yang dapat diakses. Sebagaimana telah dibahas pada bagian mengenai kelemahan perangkat keras sebelumnya, JTAG merupakan port debugging standar di tingkat cip yang memungkinkan pembacaan atau penulisan langsung ke memori perangkat.

Penguji akan memeriksa dengan saksama keberadaan pad atau pin header JTAG pada papan HPC, menggunakan perangkat seperti JTAGulator untuk mengidentifikasi pinout yang tepat. Koneksi JTAG yang berhasil dapat memungkinkan penguji untuk menghentikan prosesor dan memeriksa isi memorinya. *Port serial* UART juga menjadi target menarik lainnya, karena menghubungkan perangkat ke konsol UART dapat menampilkan *log booting* atau perintah masuk (*login prompt*). Dalam beberapa kasus, produsen membiarkan shell pemeliharaan dapat diakses melalui UART. Dengan memasukkan kredensial bawaan yang diketahui atau menginterupsi urutan booting, penguji bisa mendapatkan pijakan awal, setidaknya dengan akses perintah yang terbatas.

Bahkan jika antarmuka ini seharusnya dinonaktifkan (misalnya, melalui kunci perangkat lunak atau dengan memutus fuse bit untuk menonaktifkan JTAG), penyerang yang terampil dapat mencoba melakukan *glitching* seperti yang telah dibahas pada bab sebelumnya untuk melewati kunci tersebut dan mengaktifkan mode debug atau pengujian. Melalui pengintaian perangkat keras yang cermat ini, tim penguji penetrasi menyiapkan landasan untuk eksploitasi lebih dalam guna mengambil *firmware* dan perangkat lunak bahkan mungkin kredensial serta membangun akses awal ke dalam HPC.

Teknik umum lainnya selama tahap pengintaian adalah melakukan enumerasi terhadap semua protokol komunikasi dan layanan jaringan yang mungkin memberikan akses awal. Idealnya, kita perlu menyusun peta jaringan HPC dalam kaitannya dengan arsitektur E/E (elektrik/elektronik) secara keseluruhan, guna menentukan input sensor apa saja yang diterima oleh HPC tersebut serta pengontrol mana saja yang berkomunikasi dengannya untuk keperluan aktuasi dan kendali.

Dalam tahap ini, kita akan mencari antarmuka standar seperti CAN, LIN, FlexRay, dan Ethernet, serta akses nirkabel misalnya melalui WiFi/Bluetooth atau modem seluler. Switch Ethernet sering menjadi target; dengan mengidentifikasi switch tersebut dan menelusuri jalur kabelnya, kita dapat mengungkap infrastruktur jaringan di dalam kendaraan. Dengan peta antarmuka jaringan HPC, penguji dapat mulai menyelidiki permukaan serangan HPC, seperti layanan jaringan yang mendengarkan pada antarmuka komunikasi, API dan daemon yang berjalan pada pengontrol, serta antarmuka pengguna apa pun (misalnya aplikasi hiburan dalam kendaraan atau penangan Bluetooth) yang memproses input eksternal. Dengan melakukan enumerasi terhadap semua antarmuka tersebut, penguji penetrasi (*pentester*) menyiapkan landasan untuk eksploitasi di tahap selanjutnya, setelah serangkaian kerentanan berhasil dikonfirmasi.

Langkah 2: Ekstraksi dan Analisis Kode

Setelah akses fisik atau jaringan diperoleh, tujuan berikutnya adalah mendapatkan *firmware* dan perangkat lunak HPC untuk dianalisis. Dengan menggunakan antarmuka JTAG atau *bootloader* yang telah dibuka, penguji penetrasi (*penetration tester*) akan menyalin (*dump*) isi memori *flash* yang menyimpan kombinasi antara biner *firmware*, berkas konfigurasi, skrip, dan konten sistem berkas seperti Linux RootFS.

Jika akses langsung tidak dimungkinkan, penguji penetrasi dapat mencoba ekstraksi fisik dengan melepas cip penyimpanan seperti cip NOR *flash* dan eMMC atau bahkan memasang interposer pada cip DRAM untuk menyalin perangkat lunak yang sedang berjalan. Tujuan akhirnya adalah ekstraksi kode, yang kemudian memungkinkan penguji untuk membedah basis kode perangkat lunak HPC tersebut di luar perangkat aslinya. Mereka sering menggunakan berbagai perangkat untuk membongkar sistem berkas dan berkas biner (misalnya, menggunakan binwalk dan skrip khusus) guna mengungkap komponen perangkat lunak HPC tersebut.

Pada tahap ini, fokus utama biasanya adalah mencari berkas konfigurasi yang mungkin memuat kata sandi atau kunci enkripsi, serta paket perangkat lunak yang mungkin mengandung kerentanan yang sudah diketahui. Di sini, tim melakukan *Software Composition Analysis* (SCA), yaitu audit mendalam terhadap komponen pihak ketiga dan sumber terbuka (*open-source*) di dalam HPC. Perangkat seperti Black Duck sangat efektif untuk mengidentifikasi paket perangkat lunak sumber terbuka yang ditemukan dalam data biner yang telah diekstraksi. Menemukan pustaka (*library*) atau berkas eksekusi yang sudah usang maupun rentan merupakan salah satu cara paling efektif dan mudah untuk mulai menyusun rantai serangan (*attack chain*).

Dalam praktiknya, hal ini berarti membandingkan konten kode dengan basis data kerentanan untuk mencari CVE (*Common Vulnerabilities and Exposures*) yang sudah diketahui. Proses ini sering kali mengungkap temuan seperti pustaka kriptografi yang kedaluwarsa, perangkat pengembangan yang tertinggal, skrip shell yang tidak dihapus, atau bahkan komponen kerangka kerja (*framework*) Android maupun Linux yang memiliki kerentanan. Kesalahan konfigurasi seperti kredensial debug yang tertanam secara permanen (*hardcoded*), layanan yang tidak diperlukan namun tetap aktif dengan izin akses terbuka, atau pengaturan bawaan yang tidak aman juga kerap ditemukan selama analisis *firmware* dan dapat berfungsi sebagai titik pijak (*pivot point*) untuk serangan lebih lanjut. Setiap temuan dalam data yang disalin tersebut kemudian dievaluasi potensi eksploitasinya, sementara tim merencanakan fase serangan berikutnya.

Langkah 3: Rekayasa Balik Untuk Mengungkap Kerentanan Perangkat Lunak

Selain kerentanan pada perangkat lunak sumber terbuka (*open-source*), penguji penetrasi (*penetration tester*) akan melakukan rekayasa balik (*reverse engineering*) terhadap *firmware* untuk menemukan kerentanan *zero-day* yang muncul akibat kesalahan implementasi. Dalam tahap ini, kerusakan memori (*memory corruption*) seperti buffer *overflow* atau kesalahan *use-after-free* menjadi target utama dalam komponen kode natif.

Pertama, berkas biner yang diperoleh harus didekompilasi untuk menghasilkan kode yang lebih mudah diperiksa guna mencari kerentanan keamanan. Alat seperti Ghidra dapat digunakan untuk mencapai tujuan tersebut. Setelah mendapatkan kode sumbernya, penguji penetrasi dapat menerapkan *Static Analysis Security Testing* (SAST) untuk memindai kelemahan keamanan. Penggunaan agen kecerdasan buatan (AI) sangat bermanfaat pada tahap ini untuk memilah informasi penting dari data yang tidak relevan, dengan memfokuskan analisis hanya pada kelemahan yang benar-benar dapat dieksploitasi. Selain SAST, membangun lingkungan yang mendukung *fuzzing* juga merupakan teknik yang efektif untuk menemukan kerentanan.

Sebagai contoh, tim dapat melakukan *fuzz-testing* pada layanan perangkat lunak seperti dekoder video untuk melihat apakah input yang tidak sesuai format dapat menyebabkan layanan tersebut mengalami crash atau menimpa memori. Setiap kejadian crash akan diselidiki untuk menentukan apakah hal tersebut dapat diubah menjadi metode eksekusi kode yang andal. Celah injeksi kode merupakan jalur lain, di mana perangkat lunak HPC menerima input dan secara keliru menganggapnya sebagai kode atau perintah. Contoh sederhananya adalah perintah baris yang tidak disanitasi dengan baik sebelum diteruskan ke shell. *Fuzzing* juga dapat mengungkap *race condition*, seperti celah *time-of-check-to-time-of-use* (TOCTOU). Celah TOCTOU memungkinkan penguji untuk, misalnya, mengganti berkas atau mengubah izin akses di antara waktu perangkat lunak HPC memeriksa berkas tersebut dan waktu saat berkas itu digunakan, yang dapat berujung pada eskalasi hak akses. Tujuan akhir dari pencarian kerentanan ini adalah menemukan sebanyak mungkin celah yang dapat dieksploitasi, yang nantinya dapat dirangkaikan untuk mencapai tujuan serangan yang lebih besar.

Langkah 4: Eksploitasi dan Eksekusi Kode

Setelah mengidentifikasi kelemahan keamanan yang cukup, tim harus mengembangkan eksploit untuk menunjukkan bahwa tujuan serangan telah tercapai. Sering kali, eksploit awal mungkin hanya memberikan akses terbatas seperti menjalankan kode dalam konteks dengan hak akses rendah atau mengambil alih satu proses tertentu. Namun, keberhasilan dalam mengeksekusi perintah (meskipun bukan sebagai *root*) merupakan pencapaian yang krusial. Tahap selanjutnya adalah menyusun payload yang akan dijalankan oleh sistem komputasi berkinerja tinggi (HPC). Sebagai contoh, jika penguji menemukan kerentanan *buffer overflow* pada layanan jaringan HPC, mereka dapat menyusun *input* yang dirancang khusus (biasanya berupa rangkaian data yang panjang) untuk menimpa alamat pengembalian (*return address*) program, sehingga mengalihkan eksekusi ke *payload* milik penguji. *Payload* tersebut bisa berupa potongan kecil kode (*shellcode*) yang berfungsi membuka *remote shell* atau menambahkan akun pengguna baru.

Dalam skenario lain mungkin sebuah HPC berbasis Android memiliki aplikasi dengan celah injeksi kode yang memungkinkan penguji untuk mengeksploitasinya guna masuk ke *Android Debug Bridge* (ADB) atau untuk menjalankan shell Unix pada perangkat. Para penguji menyempurnakan eksploitasi mereka, sehingga berfungsi berulang kali tanpa menyebabkan sistem crash. Jika HPC memiliki mitigasi eksploitasi seperti ASL™ (*Address Space Layout*

™andomization), *Pointer Authentication Codes* (PAC), dan DEP (*Data Execution Prevention*), eksploitasi perlu dengan cerdik melewati mitigasi tersebut (seringkali memanfaatkan kelemahan mikroarsitektur yang memungkinkan kebocoran informasi). Anda akan melihat bahwa proses ini bersifat iteratif di mana penguji bergantian antara eksploitasi dan penemuan karena pertahanan biasanya menghalangi, sehingga penguji perlu menemukan cara untuk melewatinya sebelum melanjutkan eksploitasi lebih lanjut. Setelah berhasil, penguji penetrasi sekarang memiliki kode yang berjalan pada HPC target di bawah kendali mereka. Pada tahap ini, mereka sering kali memiliki akses setara dengan shell tingkat pengguna pada perangkat dengan kemampuan untuk mengeluarkan perintah dan mengamati respons.

OEM mungkin puas dengan laporan pentest yang menunjukkan bahwa kerentanan yang ditemukan memungkinkan eksekusi kode tingkat pengguna. Jika keterlibatan pentest telah menetapkan standar yang lebih tinggi, maka pekerjaan harus dilanjutkan untuk mencapai hak akses eksekusi yang lebih tinggi. Dari perspektif penyerang, HPC mobil telah berhasil dibajak, tetapi pekerjaan belum selesai karena kemampuan penyerang mungkin dibatasi oleh lapisan keamanan sistem operasi.

Langkah 5: Peningkatan Hak Akses di Lingkungan HPC

Seperti yang telah kita bahas di bagian tentang kelemahan, HPC yang menjalankan Linux atau Android biasanya menerapkan pemisahan hak akses, yang berarti eksekusi kode awal mungkin berada di aplikasi sandbox atau sebagai proses pengguna tanpa hak akses. Akibatnya, untuk sepenuhnya mengontrol HPC, dan secara ekstensi fungsi kendaraan penting yang dikelolanya, penyerang harus berupaya meningkatkan hak akses mereka ke tingkat yang lebih tinggi. Tim pengujian penetrasi beralih ke pencarian jalan menuju hak akses *root*. Kerentanan kernel yang terkenal menjadi alat yang ampuh pada titik ini. Jika tidak ada kerentanan yang diketahui, maka diperlukan analisis yang lebih lanjut untuk menemukan kerentanan *zero-day* di kernel itu sendiri.

Cara lain termasuk mengeksploitasi biner setuid yang salah konfigurasi (program yang berjalan sebagai *root*). Jika analisis *firmware* mengungkapkan adanya *executable* pada HPC yang memiliki bit SUID yang diatur dan dikodekan secara tidak benar, penguji dapat menyalahgunakannya untuk menjalankan *shell root*. Demikian pula, pada HPC Android, mereka mungkin mencari layanan sistem yang berjalan dengan hak istimewa tinggi yang mempercayai input dari konteks hak istimewa rendahnya. Setiap kelemahan di sini dapat mengakibatkan kompromi terhadap pengguna sistem Android atau bahkan kernel Linux yang mendasarinya. Terkadang peningkatan hak akses semudah menemukan kredensial default.

Sayangnya, tidak jarang ditemukan kasus di mana kata sandi root diatur ke "*root*" atau server SSH debugging dibiarkan aktif. Cara yang lebih halus termasuk menyalahgunakan bug driver mode kernel karena kerentanan dalam KMD akan memberi penyerang hak istimewa yang sama dengan kernel. Pada akhir fase ini, penguji penetrasi bertujuan untuk memiliki akses root tanpa batasan kendali atas sistem operasi HPC. Ini berarti mereka dapat melakukan hampir semua hal di dalam VM tamu (*Guest VM*), seperti mengubah berkas, membaca data sensitif, dan mengirim pesan jaringan apa pun.

Langkah 6: Pergerakan lateral antar mesin virtual (*virtual machine*/VM)

Tergantung pada seberapa baik aplikasi HPC diisolasi, mendapatkan kendali atas satu VM di dalam HPC mungkin tidak selalu cukup. Biasanya, untuk memengaruhi lapisan yang lebih dalam dari arsitektur E/E, diperlukan pergerakan lateral dari HPC ke VM lain atau ke ECU fisik yang terhubung ke jaringan di dalam kendaraan. Dalam kasus HPC dengan banyak VM, penyerang sering kali berupaya berpindah dari VM sistem hiburan (*infotainment*) ke VM ADAS (sistem bantuan pengemudi) guna mengganggu ECU di domain keselamatan.

Penyerang yang gigih dan memiliki akses root pada satu domain akan mencari celah konfigurasi untuk keluar dari *sandbox* (lingkungan terisolasi) tersebut. Hal ini dapat dilakukan dengan mengeksploitasi *hypervisor* atau menyalahgunakan sumber daya yang digunakan bersama antar-VM. Sebagai contoh, jika dua mesin virtual berbagi wilayah memori yang sama tanpa validasi input yang memadai, serangan yang dirancang khusus dapat memungkinkan domain yang telah disusupi untuk menyuntikkan kode ke domain lainnya. Jalur pergerakan lateral lainnya adalah menuju jaringan kendaraan. Bayangkan sebuah arsitektur di mana HPC (*High-Performance Computer*) dipisahkan dari jaringan internal kendaraan yang lebih dalam melalui sebuah gerbang (*gateway*) pusat. Untuk berkomunikasi secara langsung dengan ECU (*Electronic Control Unit*) tingkat rendah yang mengendalikan rem, sistem penggerak (*powertrain*), atau kemudi, penyerang harus memengaruhi ECU gerbang tersebut. Dengan memiliki akses root pada salah satu mesin virtual di dalam HPC, penyerang dapat mencoba mengonfigurasi ulang antarmuka jaringan untuk memalsukan komunikasi dengan ECU gerbang. Dalam praktiknya, pada tahap ini, penguji penetrasi (*penetration tester*) dapat menulis skrip atau perangkat lunak khusus di HPC untuk menghasilkan *frame* CAN, lalu menguji kemampuan mereka misalnya untuk membuka kunci pintu, memanipulasi pembacaan pada kluster instrumen, atau bahkan mengaktifkan rem maupun akselerasi.

Melalui pergerakan lateral yang dilakukan dengan cermat, satu HPC yang berhasil disusupi dapat berkembang menjadi kompromi terhadap seluruh kendaraan. Sering kali, produsen kendaraan (OEM) menggunakan arsitektur kelistrikan/elektronik (E/E) yang serupa, sehingga penguji penetrasi dapat dengan cepat memvalidasi rantai serangan mereka yang mencakup berbagai ECU. Jika informasi tersebut tidak tersedia, maka diperlukan pemeriksaan mendalam terhadap topologi jaringan internal kendaraan untuk memetakan hubungan kepercayaan (*trust relationships*) serta titik-titik transisi (*pivot points*), guna menunjukkan bagaimana pijakan awal dapat merembet hingga mengendalikan berbagai ECU sekaligus.

Langkah 7: Persistensi Serta Perintah dan Kendali Jarak Jauh (*Command and Control*)

Sebagai penguji penetrasi (*penetration tester*), jika Anda telah mencapai tahap ini, berarti Anda kemungkinan besar telah melampaui tujuan pengujian penetrasi yang ditetapkan dengan hasil yang sangat memuaskan. Namun, bagi penyerang, memastikan keberadaan mereka tetap bertahan meskipun sistem diatur ulang (*reset*) merupakan target krusial tambahan. Salah satu metode yang lugas adalah dengan memasang pintu belakang (*backdoor*) yang persisten pada HPC (*High-Performance Computer*/komputer berkinerja tinggi) kendaraan.

Sebagai contoh, penyerang dapat menambahkan kunci SSH milik mereka sendiri dan mengaktifkan layanan SSH agar dapat terhubung ke HPC dari jarak jauh kapan pun diinginkan, dengan autentikasi yang sudah terintegrasi. Selain itu, mereka mungkin memodifikasi skrip awal (*startup scripts*) atau *firmware* sehingga layanan berbahaya atau perangkat lunak sisipan (*implant*) mereka akan berjalan secara otomatis, bahkan setelah perangkat dimatikan dan dinyalakan kembali. Pada HPC berbasis Linux, hal ini bisa dilakukan dengan menambahkan skrip ke dalam direktori `/etc/init.d` atau membuat layanan `systemd` yang menjalankan pendengar (*listener*) pintu belakang. Pada sistem berbasis Android, mereka mungkin memanipulasi partisi boot atau mekanisme pembaruan OTA (*Over-The-Air*) untuk memasang kembali muatan berbahaya (*payload*) setelah pembaruan sistem dilakukan. Penguji juga dapat menyiapkan saluran perintah dan kendali (*command-and-control*/C2) yang sesuai dengan konektivitas kendaraan. Misalnya, penyerang dapat menjalankan agen ringan yang secara berkala menghubungi server C2 (menggunakan HTTPS atau bahkan menyamarkannya di dalam data telemetri biasa) untuk mengambil perintah.

Sebagai alternatif, HPC dapat diinstruksikan untuk membuka *reverse shell* ke *host* tertentu setiap kali perangkat tersebut daring (*online*). Tujuannya adalah untuk menyimulasikan skenario di mana penyerang dapat kembali lagi setelah beberapa hari atau minggu dan tetap memiliki akses masuk ke dalam kendaraan. Dengan kata lain, kendaraan tersebut tetap berada dalam kendali penyerang (*owned*) sampai seluruh sistem diperbaiki (*patched*). Dalam pengujian tingkat lanjut, penguji penetrasi memperluas cakupan mereka hingga ke luar kendaraan itu sendiri, dengan menyoroti risiko serangan rantai pasok (*supply chain attacks*). Alih-alih menyerang HPC saat sudah terpasang di kendaraan, tim penyerang (*red team*) mungkin mencoba memanipulasi perangkat lunak bahkan sebelum perangkat lunak tersebut diinstal. Kita akan membahas mengenai serangan rantai pasok dan cara menanganinya pada bagian berikutnya.

8.5 SERANGAN RANTAI PASOK DALAM EKOSISTEM KEAMANAN HPC KENDARAAN

Serangan rantai pasok merupakan cara yang sangat efisien untuk mengompromikan banyak sistem sekaligus dengan menyisipkan kode berbahaya ke dalam komponen yang digunakan secara luas. Hal ini mungkin memerlukan penyusupan ke dalam jaringan pengembangan atau *server build* milik pemasok tingkat pertama. Sebagai contoh, jika penyerang berhasil menginfeksi paket pembaruan untuk *firmware* HPC atau mengompromikan infrastruktur penandatanganan kode mereka berpotensi mendistribusikan pembaruan yang tampak sah namun sebenarnya mengandung backdoor ke seluruh kendaraan dari model tertentu. Pendekatan semacam ini mampu melewati berbagai mekanisme pertahanan di dalam kendaraan karena *firmware* berbahaya tersebut tiba dalam kondisi telah ditandatangani dan dianggap tepercaya.

Meskipun sebagian besar masih bersifat hipotetis, peningkatan berkelanjutan dalam keamanan defensif kemungkinan akan memaksa penyerang canggih terutama aktor negara untuk beralih ke serangan rantai pasok, misalnya dengan menempatkan pihak internal, alih-alih menghabiskan banyak sumber daya untuk menembus kontrol keamanan yang sudah

diterapkan. Taktik terkait lainnya adalah menargetkan dependensi pihak ketiga dalam rantai pasok perangkat lunak kendaraan. Sebagaimana dalam eksploitasi rantai pasok TI klasik, penyerang dapat menyumbangkan kode ke proyek sumber terbuka yang digunakan dalam HPC, atau mengeksploitasi pustaka yang telah dikompromikan yang kemudian disertakan ke dalam *firmware* kendaraan. Begitu komponen yang telah disusupi tersebut diintegrasikan, setiap kendaraan menjadi rentan secara desain.

Setelah mendalami berbagai teknik pengujian penetrasi, kini saatnya kita mengalihkan fokus ke peretasan di dunia nyata untuk melihat bagaimana metode-metode teoretis tersebut dapat diterapkan. Hal ini diharapkan dapat memperkuat premis awal kita bahwa serangan terhadap HPC (sistem komputasi berkinerja tinggi) adalah sesuatu yang mungkin terjadi, bukan sekadar skenario hipotetis belaka.

8.6 ANALISIS STUDI KASUS PERETASAN DAN EKSPLOITASI HPC KENDARAAN

Untuk memperkuat konsep-konsep yang telah dipaparkan sebelumnya, kita akan menelaah tiga studi kasus mengenai kompromi sistem yang diungkapkan ke publik yang melibatkan HPC (*High-Performance Computer*). Setiap kasus menyoroti vektor serangan, tahapan serangan, dan hasil akhir yang berbeda. Berdasarkan riset keamanan, studi kasus ini menggambarkan perkembangan peretasan kendaraan selama satu dekade terakhir. Sebagaimana telah dilakukan pada bab-bab sebelumnya, tahapan serangan dipetakan ke dalam kerangka kerja MITRE ATT&CK agar lebih mudah dipahami secara terstruktur. Seperti yang terlihat di bawah ini, setiap studi kasus memanfaatkan lebih dari satu kelemahan keamanan untuk mencapai tujuan serangan. Anda diajak untuk mengidentifikasi kelemahan keamanan dalam setiap studi kasus dengan menggunakan materi yang telah kita bahas sejauh ini.

Studi Kasus 1: Mazda IVI

Connectivity Master Unit (CMU) pada kendaraan Mazda merupakan sistem infotainment tipikal yang mengelola Bluetooth, navigasi, dan integrasi ponsel pintar. Pada HPC ini, aplikasi yang berinteraksi dengan pengguna dijalankan di atas sistem operasi (OS) yang kompleks, sementara konektivitas ke jaringan CAN ditangani oleh mikrokontroler (MCU) pendamping yang menjalankan sistem operasi waktu nyata (RTOS). Dalam studi kasus ini, para peneliti menemukan beberapa kerentanan yang memengaruhi model kendaraan, seperti Mazda 3 keluaran tahun 2014 hingga 2021. Kerentanan tersebut muncul akibat sanitasi input yang tidak memadai, yang memungkinkan penyerang untuk mengeksekusi kode arbitrer dengan hak akses root melalui penyambungan perangkat USB yang telah dimodifikasi secara khusus ke dalam sistem. Gambar di bawah ini mengilustrasikan rantai serangan tersebut:



Gambar 8.6 Rantai serangan dari pembaruan otomatis via USB hingga interferensi pada MCU kendaraan

Penemuan dan Pengintaian

Selama fase penemuan, para peneliti terlebih dahulu membangun pemahaman yang jelas mengenai tata letak perangkat keras CMU, termasuk SoC aplikasi utama, MCU pendamping, dan perangkat penyimpanan non-volatil eksternal. Pemetaan perangkat keras awal ini diikuti dengan ekstraksi *firmware* dan rekayasa balik untuk memahami peran setiap komponen perangkat lunak utama. Dengan menelusuri jalur eksekusi pembaruan perangkat lunak, para peneliti mengidentifikasi pustaka serta fungsi internal yang bertanggung jawab atas penanganan dan validasi pembaruan. Salah satu temuan penting adalah bahwa beberapa operasi dilakukan pada paket pembaruan sebelum verifikasi tanda tangan selesai. Urutan ini segera mengungkap titik pemeriksaan krusial untuk potensi kerentanan, karena logika apa pun yang memproses data pembaruan yang tidak tepercaya sebelum verifikasi kriptografis akan menjadi bagian dari permukaan serangan (*attack surface*) yang dapat dieksploitasi.

Kerentanan pertama ditemukan pada jalur komunikasi USB antara CMU dan perangkat seluler Apple. Perangkat berbahaya dapat mengirimkan string yang dirancang khusus dan tidak sesuai format, yang kemudian diteruskan langsung ke operasi basis data internal tanpa sanitasi yang memadai, sehingga mengakibatkan kerentanan injeksi SQL. Karena proses yang menangani antarmuka ini berjalan dengan hak akses root, pernyataan SQL yang disuntikkan akan dieksekusi dalam konteks yang memiliki hak akses istimewa. Hal ini memungkinkan manipulasi langsung terhadap basis data SQLite yang mendasarinya dan dapat dimanfaatkan untuk tindakan seperti modifikasi data, pembuatan berkas, serta perusakan status sistem. Meskipun kerentanan ini tidak secara langsung menjamin eksekusi kode arbitrer, ia menyediakan sarana (*primitif*) ampuh yang dapat dikombinasikan dengan kelemahan lain untuk meningkatkan skala serangan hingga mencapai kompromi sistem secara menyeluruh.

Pemeriksaan Perangkat Keras

CMU diimplementasikan sebagai satu papan sirkuit cetak (PCB) yang mengintegrasikan pengontrol Freescale dengan nomor komponen SCIMX06DAVT10AC, serta penyimpanan NAND *flash* dan eMMC eksternal perangkat tersebut mencakup mikrokontroler Renesas dengan nomor komponen R5F35MCEJFF serta modul komunikasi Bluetooth dari Murata.

Konektivitas sistem disediakan melalui beberapa antarmuka, termasuk sejumlah port USB, antarmuka CAN, dan konsol serial.

Dari perspektif arsitektur, tata letak papan (*board layout*) dan pembagian komponen menunjukkan desain modular, di mana pemrosesan tingkat aplikasi ditangani oleh SoC Freescale, sementara fungsi-fungsi tambahan tertentu serta tugas-tugas yang sensitif terhadap waktu didelegasikan ke mikrokontroler (MCU) Renesas. Pemisahan fungsional semacam ini lazim ditemukan pada unit kepala (*head unit*) otomotif dan sistem infotainmen, terutama yang melibatkan akses ke bus CAN serta tugas-tugas yang memerlukan ketepatan waktu. Meskipun pembagian seperti ini sering kali ditujukan untuk membatasi penyebaran kesalahan (*fault propagation*) dan menyederhanakan desain sistem, hal tersebut dengan sendirinya tidak membentuk batasan keamanan (*security boundary*). Sebagaimana akan kita lihat pada bagian-bagian selanjutnya, efektivitas pemisahan ini pada akhirnya bergantung pada bagaimana hubungan kepercayaan dan jalur komunikasi antara prosesor aplikasi dan MCU pendamping tersebut diimplementasikan.

Analisis Komposisi Perangkat Lunak

SoC aplikasi utama menjalankan sistem operasi berbasis Linux, sebagaimana dikonfirmasi melalui output konsol serial dan *log bootloader*. Selama boot awal, sistem memasang beberapa sistem file, banyak di antaranya dikonfigurasi dengan izin baca-tulis. Dari perspektif penyerang, hal ini secara signifikan memperluas permukaan persistensi dan perusakan pasca-kompromi, terutama ketika layanan sistem dan komponen pembaruan berada pada penyimpanan yang dapat ditulis. Penyimpanan persisten didistribusikan di seluruh partisi mtdblock, ffx01, dan mmcblk. Tata letak penyimpanan heterogen ini memungkinkan beberapa jalur untuk manipulasi tingkat sistem file, termasuk modifikasi artefak pembaruan, file konfigurasi, dan komponen yang dapat dieksekusi yang kemudian dikonsumsi oleh layanan istimewa.

Perilaku otentikasi konsol tidak didokumentasikan secara publik. Namun, log boot dan inisialisasi yang detail mengungkapkan informasi sistem yang rinci, termasuk perilaku startup layanan dan penamaan komponen internal. Tingkat pengamatan ini secara material mengurangi upaya rekayasa balik dan secara langsung mendukung pengembangan eksploitasi dengan mengungkapkan jalur eksekusi kandidat dan layanan istimewa. CMU mendukung pembaruan *firmware* melalui menu diagnostik dan menggunakan paket pembaruan dalam bentuk *file up* yang disediakan melalui perangkat penyimpanan massal USB. File pembaruan ini diimplementasikan sebagai arsip yang dilindungi kata sandi dan berisi komponen sistem penting, termasuk sistem file root, *bootloader*, dan data konfigurasi. Pipeline pembaruan melakukan validasi struktural, ekstraksi sertifikat, dan verifikasi tanda tangan. Namun, beberapa operasi penguraian dan penanganan dieksekusi sebelum verifikasi kriptografi selesai, yang merupakan petunjuk lain untuk kemungkinan eksploitasi. Seperti yang dibahas sebelumnya, logika apa pun yang memproses konten pembaruan yang tidak tepercaya sebelum validasi integritas menjadi bagian dari permukaan serangan pra-autentikasi yang efektif.

Rekayasa balik terhadap alur kerja pembaruan mengidentifikasi layanan pembaruan `svcjciupdatea.so` dan `svcjciupdates.so`, beserta pustaka pendukung seperti `libjcire*flash*ua.so` dan `libjcisecurity.so`, sebagai komponen utama yang bertanggung jawab untuk memuat, memvalidasi, dan mengeksekusi instruksi pembaruan. Para peneliti berfokus pada fungsi-fungsi yang menangani penemuan berkas, pengelolaan arsip, ekstraksi, dan pemanggilan perintah, karena jalur kode ini secara langsung menghubungkan konten pembaruan yang dikendalikan penyerang dengan operasi sistem yang memiliki hak akses istimewa.

Pilihan desain ini sangat krusial. Mekanisme pembaruan dijalankan dengan hak akses yang ditingkatkan serta memiliki akses langsung ke berkas sistem dan jalur eksekusi. Akibatnya, kelemahan dalam penguraian paket pembaruan atau penanganan perintah dapat langsung berubah menjadi sarana eksekusi kode yang berdampak tinggi. Kerentanan berikut ditemukan dalam tumpukan perangkat lunak CMU. Secara keseluruhan, kerentanan-kerentanan ini menunjukkan bagaimana akses periferal dan fungsionalitas pembaruan dapat dirangkaikan untuk mencapai kompromi sistem secara menyeluruh:

- **Injeksi SQL pada DeviceManager (CVE-2024-8355 / ZDI-24-1208)**: Kurangnya sanitasi input pada fungsi `eInsertDeviceEntry` di dalam `/jci/devicemanager/libdevicemanager.so` memungkinkan penyerang untuk menyisipkan pernyataan SQL dengan memalsukan nomor seri iAP. Input yang disisipkan tersebut ditulis langsung ke dalam *backend* SQLite (`DeviceDatabase.db`) tanpa melalui validasi. Hal ini memungkinkan manipulasi konten basis data dan dapat dimanfaatkan untuk pembuatan berkas, pengungkapan informasi, serta kerusakan basis data. Karena komponen DeviceManager beroperasi dengan hak akses yang ditingkatkan, dampak dari kelemahan ini melampaui sekadar integritas data dan secara langsung memengaruhi jaminan keamanan di tingkat sistem.
- **Injeksi perintah melalui `REFLASH_DDU_FindFile` (CVE-2024-8359 / ZDI-24-1191)**: Fungsi `RE*FLASH*_DDU_FindFile` dalam `libjcire*flash*ua.so` menyusun perintah sistem operasi menggunakan input yang dikendalikan oleh penyerang tanpa sanitasi yang memadai. Selama proses pembaruan *firmware*, hal ini memungkinkan penyerang untuk menyisipkan perintah *shell arbitrer* (sewenang-wenang). Dalam praktiknya, ini menyediakan sarana eksekusi kode jarak jauh yang andal di dalam jalur eksekusi pembaruan.
- **Injeksi perintah melalui ™EFLASH_DDU_ExtractFile (CVE-2024-8360 / ZDI-24-1192)**: Fungsi RE*FLASH*_DDU_ExtractFile menunjukkan jenis kelemahan penanganan input yang serupa. Parameter yang tidak dibersihkan (*unsanitized*) disertakan ke dalam eksekusi perintah sistem selama proses ekstraksi file, yang memungkinkan terjadinya injeksi perintah secara langsung dan berujung pada kompromi sistem secara menyeluruh.
- **Injeksi perintah melalui UPDATES_ExtractFile (CVE-2024-8358 / ZDI-24-1190)**: Di dalam `svcjciupdates.so`, fungsi UPDATES_ExtractFile memproses arsip pembaruan menggunakan jalur berkas (*file path*) dan parameter yang dapat dikendalikan oleh

penyerang tanpa validasi yang memadai. Hal ini memungkinkan manipulasi jalur serta injeksi perintah selama proses ekstraksi pembaruan, yang pada akhirnya berujung pada eksekusi kode dengan hak akses istimewa (*privileged code execution*).

- **Tidak adanya root of trust pada SoC aplikasi (CVE-2024-8357 / ZDI-24-1189)**: Urutan bootstrap CMU pada SoC aplikasi tidak menerapkan autentikasi terhadap komponen-komponen boot. Akibatnya, penyerang yang memiliki akses tulis ke penyimpanan terkait dapat memodifikasi kode *bootloader*, mempertahankan keberadaan perangkat lunak berbahaya (*malware*) meskipun sistem telah dimulai ulang, serta memperoleh kendali eksekusi sebelum sistem operasi dimuat. Hal ini secara mendasar merusak mekanisme keamanan perangkat lunak tingkat lebih tinggi yang diterapkan di dalam lingkungan Linux.
- **Eksekusi kode tanpa tanda tangan digital (*unsigned code*) pada VIP MCU (CVE-2024-8356 / ZDI-24-1188)**: VIP MCU tambahan yang bertanggung jawab atas komunikasi CAN dan LIN menerima citra (*image*) *firmware* tanpa verifikasi tanda tangan. Dengan demikian, penyerang dapat mengganti *firmware* VIP dengan citra yang telah dimodifikasi dan mendapatkan kendali langsung atas komunikasi jaringan di dalam kendaraan.

Kelemahan ini meruntuhkan pemisahan yang dirancang antara domain pemrosesan aplikasi dan fungsi komunikasi kendaraan. Dampaknya, kompromi yang bermula dari domain infotainment atau aplikasi dapat merembet ke bus CAN, sehingga secara langsung memperluas cakupan dampak (*blast radius*) hingga ke sistem kendaraan yang berkaitan dengan keselamatan.

Penjabaran Langkah Demi Langkah Rantai Serangan

Eksploitasi terhadap kerentanan injeksi perintah yang dijelaskan di atas dimungkinkan karena instalasi pembaruan perangkat lunak dapat dipicu secara otomatis saat perangkat penyimpanan massal USB dihubungkan; ini merupakan "fitur yang diketahui" (*known feature*) yang dapat diaktifkan hanya dengan menempatkan sebuah berkas kosong bernama `jci-autoupdate` di dalam perangkat penyimpanan tersebut.

Persiapan dan Eksploitasi Awal

Eksploitasi kerentanan injeksi perintah sistem operasi pada CMU Mazda tergolong sangat mudah; hanya diperlukan akses fisik ke port USB kendaraan. Penyerang menyiapkan perangkat penyimpanan massal USB berformat FAT32 yang berisi berkas yang dirancang secara berbahaya, dengan nama yang memuat perintah sistem operasi (OS) untuk dieksekusi. Nama berkas tersebut harus diakhiri dengan ekstensi up agar dapat dikenali oleh mekanisme pembaruan perangkat lunak.

Untuk mempermudah eksploitasi, penyerang dapat memanfaatkan kerentanan ZDI-CAN-23420, yang tidak memerlukan persyaratan tambahan seperti penyusunan berkas pembaruan yang valid. Selain itu, pembaruan perangkat lunak dapat dipicu secara otomatis dengan menempatkan sebuah berkas kosong bernama jci-autoupdate pada perangkat USB, yang akan mengaktifkan fitur "pembaruan otomatis" (*auto-update*) saat perangkat tersebut dihubungkan.

Eksekusi dan Persistensi

Setelah perangkat USB dihubungkan ke kendaraan dan sistem membaca berkas yang telah dimanipulasi, perintah yang tertanam di dalamnya akan dijalankan, sehingga memberikan penyerang akses ke CMU. Akses ini memungkinkan manipulasi sistem berkas akar (*root filesystem*), seperti memasang komponen sistem yang telah disisipi *backdoor* agar tetap ada (*persistent*) meskipun sistem dimulai ulang (*reboot*). Perubahan ini memastikan penyerang tetap memegang kendali atas sistem yang telah disusupi untuk jangka waktu yang tidak terbatas.

Pergerakan Lateral (*Lateral Movement*)

Setelah berhasil masuk ke dalam CMU, penyerang dapat bergerak secara lateral menuju mikrokontroler VIP. Hal ini dilakukan dengan memasang *firmware* khusus pada VIP, yang memberikan akses tanpa batas ke jaringan kendaraan seperti CAN dan LIN. Meskipun penelitian ini tidak mengeksplorasi dampak spesifiknya, kendali tersebut secara teoretis dapat memungkinkan penyerang untuk memanipulasi operasional kendaraan, termasuk fungsi-fungsi yang krusial bagi keselamatan.

Studi kasus ini menunjukkan kepada kita bagaimana kelemahan-kelemahan umum dapat digabungkan dengan relatif mudah untuk melancarkan serangan yang sukses terhadap HPC, yang pada akhirnya dapat memberikan kendali langsung atas komponen-komponen yang lebih kritis dalam jaringan internal kendaraan. Selanjutnya, kita akan membahas studi kasus kedua yang menggunakan teknik serangan yang lebih canggih.

Studi Kasus 2: Tesla IVI

Studi kasus ini menunjukkan peretasan sistem IVI dari jarak jauh tanpa memerlukan interaksi pengguna (*zero-click*), dengan memanfaatkan celah keamanan memori pada *stack* Bluetooth eksklusif yang dapat diakses sebelum proses autentikasi. Dengan mengekstraksi dan melakukan rekayasa balik (*reverse engineering*) terhadap *firmware* perangkat, penyerang menemukan layanan Bluetooth dengan hak akses *root* yang tidak dilengkapi dengan mitigasi serangan standar. Mereka kemudian memicu kondisi *Use-After-Free* dalam logika segmentasi dan penyusunan ulang (*reassembly*) L2CAP saat memproses payload ACL yang dikirim melalui tautan HCI yang telah terbentuk. Melalui manipulasi heap yang terkendali, mereka memperoleh kemampuan untuk membaca dan menulis data secara bebas (*arbitrary read and write*), membocorkan alamat heap, pustaka (*library*), dan stack untuk menembus mekanisme ASLR, serta mengalihkan alur eksekusi menggunakan ROP chain yang dirancang khusus guna menjalankan kode dengan hak akses *root*. Peretasan ini memungkinkan akses yang bersifat persisten serta kendali jarak jauh (*command-and-control*), dengan dampak yang mencakup pengambilalihan sistem IVI, kebocoran data pengguna, penyadapan di dalam kendaraan, hingga potensi pergerakan lebih lanjut (*pivoting*) ke perangkat seluler yang terhubung.

Studi kasus ini mengilustrasikan bagaimana satu kerentanan yang dapat diakses dari jarak jauh dalam tumpukan (*stack*) nirkabel berhak akses tinggi (*privileged*) dapat meningkat menjadi kompromi sistem kelas HPC (*High-Performance Computing*) secara menyeluruh. Mari kita telusuri langkah-langkah yang mereka ambil untuk mencapai tujuan serangan tersebut.

Penemuan dan Pengintaian

Selama fase penemuan, penyerang melakukan pengintaian ekstensif terhadap sistem *In-Vehicle Infotainment* (IVI) Alpine Halo9 iLX-F509. Perangkat ini merupakan sistem IVI eksternal yang dilengkapi dengan layar sentuh dan berbagai opsi konektivitas, termasuk USB, Wi-Fi, dan Bluetooth. Perangkat ini juga mendukung Apple CarPlay dan Android Auto. Penyerang memfokuskan perhatian pada tumpukan Bluetooth perangkat tersebut, sebuah implementasi eksklusif yang mereka curigai mungkin kurang diperiksa secara mendalam terkait kerentanan.

Untuk memperoleh pemahaman mendalam tentang sistem tersebut, penyerang mendapatkan *firmware* dari chip *embedded MultiMediaCard* (eMMC) perangkat tanpa harus melepas solder komponen tersebut. Melalui rekayasa balik (*reverse engineering*) terhadap *firmware* khususnya pada berkas biner `btapp` yang bertanggung jawab atas fungsi Bluetooth mereka menemukan bahwa berkas tersebut merupakan executable ARM 32-bit yang berjalan dengan hak akses root. Biner ini tidak dilengkapi dengan beberapa mitigasi keamanan, seperti *stack canaries* dan dukungan *Position Independent Executable* (PIE), sehingga menjadikannya target yang menarik untuk dieksploitasi. Selain itu, keberadaan simbol-simbol (*symbols*) di dalam biner tersebut mempermudah proses rekayasa balik.

Penyerang menganalisis protokol Bluetooth yang diimplementasikan oleh perangkat, dengan fokus pada lapisan *Host Controller Interface* (HCI) serta *Logical Link Control and Adaptation Protocol* (L2CAP). Mereka menemukan kerentanan *Use-After-Free* (UAF) dalam penanganan fragmen data *Asynchronous Connection-Less* (ACL) HCI di dalam fungsi `prh_l2_sar_data_ind`. Kerentanan ini dapat dieksploitasi tanpa interaksi pengguna, yang mengklasifikasikannya sebagai kerentanan *zero-click*.

Penjabaran Langkah Demi Langkah Rantai Serangan

Mari kita telusuri setiap tahapan dalam rantai serangan tersebut secara mendetail, sebagaimana diilustrasikan dalam Gambar 8.7



Gambar 8.7 Rantai serangan dari *zero-click* hingga *reverse root shell*

Akses Awal

Akses awal diperoleh melalui kerentanan *zero-click* pada stack Bluetooth. Karena lalu lintas L2CAP diproses sebelum autentikasi atau interaksi pengguna apa pun terjadi, penyerang dapat mengirimkan paket Bluetooth yang dirancang khusus langsung ke perangkat target. Satu-satunya prasyarat untuk menargetkan perangkat tersebut adalah mengetahui alamat perangkat Bluetooth-nya (BDADDR). Dalam praktiknya, alamat ini bisa didapatkan dengan cara memantau (*sniffing*) lalu lintas Bluetooth di sekitar menggunakan perangkat seperti Ubertooth, menyimpulkan alamat tersebut dari alamat MAC modul Wi-Fi dalam desain yang menggunakan teknologi koeksistensi, atau melakukan *brute-force* pada byte bagian bawah alamat jika byte bagian atasnya sudah diketahui. Dari sudut pandang penyerang, hal ini menyediakan titik masuk yang sepenuhnya jarak jauh (*remote*) dan tanpa autentikasi ke dalam stack protokol yang memiliki hak akses istimewa (*privileged*).

Pemicu *Use-After-Free*

Kerentanan utamanya adalah kondisi *Use-After-Free* (UAF) yang terjadi selama proses penyusunan kembali (*reassembly*) data Bluetooth yang terfragmentasi. Sistem menggunakan objek tautan internal untuk melacak status koneksi yang telah terbentuk, termasuk penyangga data (*data buffer*) dan penunjuk posisi penulisan (*write-position pointer*) yang menandai lokasi

penyimpanan fragmen data berikutnya. Eksploitasi ini dipicu dengan mengirimkan urutan fragmen tertentu:

- Fragmen awal menyiapkan buffer data yang valid. Fragmen "awal" (*start*) kedua kemudian dikirimkan, yang sengaja dibuat agar gagal dalam pemeriksaan validasi panjang data.
- Akibat kesalahan logika, sistem membebaskan (*free*) buffer data tersebut namun gagal menghapus penunjuk posisi penulisan internal.

Hal ini menyisakan *dangling pointer* (penunjuk yang mengarah ke lokasi yang sudah dibebaskan) yang tetap tertuju ke suatu titik di tengah bagian heap yang telah dibebaskan. Saat penyerang mengirimkan fragmen data berikutnya, sistem terus menulis ke memori yang sudah dibebaskan tersebut.

Heap Overflow

Penyerang kemudian mengubah kondisi *Use-After-Free* (UAF) ini menjadi heap overflow. Dengan memberikan nilai panjang data yang berbahaya pada fragmen baru, mereka menipu sistem agar meyakini bahwa data yang sedang disusun ulang jauh lebih besar daripada ukuran buffer aslinya. Seiring datangnya fragmen tambahan, sistem menggunakan dangling pointer tersebut untuk menulis data melampaui batas alokasi heap yang asli, sehingga memungkinkan penyerang menimpa objek lain yang berada di dekatnya dalam memori.

Heap Spraying dan Manipulasi

Untuk memastikan overflow mengenai target yang diinginkan, penyerang melakukan heap spraying. Mereka membuat sekitar 50 objek saluran logis (*logical channel objects*) untuk mengisi heap dengan struktur yang dapat diprediksi. Langkah ini memastikan bahwa ketika *overflow* terjadi, objek saluran target akan rusak dengan cara yang dapat diperkirakan. Hasil dari tahapan ini adalah tata letak heap yang deterministik, yang dapat digunakan kembali dalam berbagai upaya eksploitasi.

Arbitrary Write Primitive

Penyerang mencapai kemampuan untuk menulis ke alamat memori mana pun dengan memanfaatkan *Enhanced Retransmission Mode* (ERTM) sejenis saluran kompleks yang mereka "palsukan" dengan menimpa objek saluran standar. Dengan merusak penunjuk penyusunan ulang data internal di dalam struktur ERTM palsu ini, mereka dapat mengarahkan penunjuk tersebut ke lokasi mana pun dalam memori sistem. Data apa pun yang dikirimkan ke saluran tersebut kemudian akan ditulis ke alamat yang dipilih oleh penyerang. Kemampuan ini menjadi fondasi utama bagi tahapan eksploitasi selanjutnya, karena memungkinkan modifikasi langsung terhadap memori yang mengendalikan alur eksekusi (*control-flow*) serta struktur data yang krusial.

Arbitrary Read Primitive

Kemampuan untuk membaca dari alamat memori mana pun diperoleh dengan menyalahgunakan logika transmisi ulang (*retransmission logic*) pada ERTM. Penyerang menyusun "permintaan transmisi ulang" (*retransmission request*) yang memaksa sistem untuk mengirimkan kembali data yang sebenarnya sudah diterima sebelumnya. Dengan menggunakan primitif tulis (*write primitive*) mereka untuk menimpa daftar transmisi ulang

internal dengan alamat memori target, sistem dikelabui agar "mengirimkan kembali" isi memori tersebut kepada penyerang. Primitif baca (*read primitive*) ini digunakan untuk membocorkan penunjuk heap, alamat dasar pustaka bersama (*shared library*), serta alamat stack.

Melewati ASLR

Dengan memanfaatkan kapabilitas AAR (*Arbitrary Address Read*), penyerang membocorkan alamat dari heap, libc, dan stack. Pengungkapan informasi ini cukup untuk menembus mekanisme *Address Space Layout Randomization* (ASLR). Setelah alamat dasar dari modul dan wilayah memori yang relevan diketahui, pengalihan alur kontrol (*control-flow redirection*) dapat dilakukan secara stabil dan konsisten. Tanpa kebocoran informasi ini, eksploitasi akan bersifat probabilistik dan tidak stabil.

Peningkatan Hak Akses (*Privilege Escalation*)

Berkas biner aplikasi Bluetooth (*btapp*) dijalankan dengan hak akses root. Akibatnya, setiap eksekusi kode yang berhasil dilakukan dalam konteks proses ini akan langsung memberikan kendali tingkat root atas sistem. Namun, eksekusi kode tersebut harus diarahkan ke logika yang dikendalikan oleh penyerang. Untuk mencapai hal ini, penyerang menyusun *return-oriented programming* (ROP) chain menggunakan gadget yang terletak di wilayah memori yang telah diketahui melalui kebocoran alamat tadi.

Eksekusi Kode

Penyerang menggunakan primitif AAW (*Arbitrary Address Write*) untuk menimpa stack pada thread pemrosesan Bluetooth dengan ROP chain yang telah dirancang khusus. Rantai ini disusun untuk memanggil fungsi `*system(payload*)`. Keberhasilan eksekusi rantai tersebut akan memicu terbentuknya *root reverse shell* melalui protokol TCP/IP pada perangkat target, yang memberikan kendali interaktif penuh atas sistem IVI (*In-Vehicle Infotainment*). Pada tahap ini, penyerang telah beralih dari kerentanan pada tingkat protokol nirkabel menuju kompromi sistem secara menyeluruh.

Dampak

Root reverse shell tersebut membentuk saluran komunikasi antara sistem IVI yang telah dikompromikan dan infrastruktur milik penyerang. Saluran ini memungkinkan eksekusi perintah jarak jauh serta interaksi *real-time* dengan perangkat. Secara operasional, sistem IVI tersebut berubah menjadi endpoint yang dikelola dari jarak jauh di dalam kendaraan, sehingga penyerang dapat mengubah tampilan layar, mencuri informasi buku telepon, atau menguping melalui mikrofon eksternal.

Studi kasus ini menunjukkan bagaimana penyerang mengombinasikan kerusakan memori (*memory corruption*), pengungkapan informasi, dan manipulasi alur kontrol untuk menjalankan rangkaian serangan. Dalam studi kasus berikutnya, kita akan melangkah lebih jauh dari skenario dasar ini dan menelaah skenario intrusi yang jauh lebih canggih, dengan menyoroti teknik-teknik tambahan serta kompleksitas operasional yang diperlukan untuk memperluas serangan dari batasan HPC (*High-Performance Computing*).

Studi Kasus 3: Tesla Pwn2Own 2023

Studi kasus ini menyajikan rantai serangan tiga tahap terhadap sistem *infotainment* Tesla, yang dimulai dengan *heap overflow* pada perangkat Bluetooth yang telah dipasangkan di dalam proses server Bluetooth, kemudian berlanjut melalui wilayah memori bersama (*shared memory region*) dengan chipset Wi-Fi dan Bluetooth, hingga akhirnya mencapai *firmware* Wi-Fi eksekusi, lalu menyalahgunakan kerentanan pada driver Wi-Fi Linux untuk mencapai eksekusi kode di tingkat kernel Linux pada sistem infotainment eksekusi, lalu menyalahgunakan kerentanan pada driver Wi-Fi Linux untuk mencapai eksekusi kode di tingkat kernel Linux pada sistem infotainmen.

Dari posisi akses penuh (*root*) yang tidak dibatasi ini, para penyerang beralih melalui jaringan Ethernet di dalam kendaraan menuju *Security Gateway* (gerbang keamanan). Mereka mengeksploitasi kelemahan logika *Time of Check to Time of Use* (TOCTOU) dalam alur pembaruan *firmware*-nya. Hal ini memungkinkan pembaruan yang sah (telah ditandatangani) pada media penyimpanan eksternal diganti dengan image berbahaya, yang kemudian diterapkan tanpa melalui verifikasi ulang.

Dengan memodifikasi *bootloader* pada *Security Gateway* untuk menonaktifkan *secure boot* berbasis perangkat lunak, penyerang dapat menjalankan kode secara persisten di dalam *gateway* tersebut. Mereka kemudian memanfaatkan layanan *firmware* yang sudah ada untuk mengirimkan pesan CAN arbitrer ke unit kontrol elektronik (ECU) hilir. Hasilnya adalah kompromi lintas domain secara menyeluruh dari *High-Performance Computer* (HPC) multimedia hingga jaringan kendali di dalam kendaraan. Hal ini menunjukkan bagaimana rangkaian kelemahan keamanan memori dan logika pembaruan dapat digabungkan untuk menembus mekanisme *sandboxing*, mematahkan asumsi keamanan *secure boot*, serta memungkinkan kendali yang bersifat permanen dan mencakup seluruh sistem kendaraan. Gambar 8.8 mengilustrasikan tahapan serangan ini dari awal hingga akhir.



Gambar 8.8 Tiga fase pembuatan eksploit

Penemuan dan Pengintaian

Selama fase penemuan, para peneliti keamanan melakukan analisis mendalam terhadap sistem Tesla Model 3 untuk mengungkap kerentanan potensial yang dapat dieksploitasi. Mereka berfokus pada sistem *infotainment* kendaraan, komponen krusial dalam domain multimedia yang terhubung dengan domain kendaraan melalui Se*curity Gateway* (gerbang keamanan) yang terletak pada papan sirkuit cetak (PCB) yang sama. Domain *infotainment* ini dijalankan menggunakan prosesor AMD Ryzen dengan sistem operasi Linux, serta dilengkapi dengan konektivitas Wi-Fi dan Bluetooth.

Para peneliti memperoleh perangkat keras untuk pengujian, termasuk papan infotainment dan layar, serta menggunakan perangkat analisis statis (seperti Ghidra dan IDA Pro) dan teknik instrumentasi dinamis. Mereka memanfaatkan program peneliti keamanan Tesla, yang memberikan akses SSH ke sistem infotainment, sehingga memungkinkan dilakukannya analisis dan pengujian secara menyeluruh. Permukaan serangan (*attack surface*) utama yang diidentifikasi meliputi:

- Bluetooth stack: *Proses Bluetooth Server* (bthserv), yang bertanggung jawab menangani komunikasi Bluetooth, dikompilasi tanpa dukungan *Position Independent Executable* (PIE). Luasnya permukaan serangan akibat penggunaan bahasa

pemrograman C dan *heap allocator* kustom menjadikannya target yang menarik untuk dieksploitasi.

- *Firmware* dan driver Wi-Fi: Modul Wi-Fi/Bluetooth yang digunakan bersama membuka peluang untuk beralih dari Bluetooth ke Wi-Fi, dengan potensi kerentanan pada *firmware* Wi-Fi serta interaksinya dengan driver kernel Linux (bcmdhd).
- *Security gateway*: Berfungsi sebagai jembatan antara domain multimedia dan domain kendaraan, Security *Gateway* bertanggung jawab untuk menyaring pesan *Controller Area Network* (CAN) serta mengelola pembaruan *firmware* untuk *Electronic Control Unit* (ECU) lainnya di dalam kendaraan.

Penjabaran Langkah Demi Langkah Rantai Serangan

Mari kita bahas setiap fase dari rantai serangan ini secara mendetail.

Fase 1: Akses Awal

Pada fase 1, penyerang memperoleh akses awal dengan mengeksploitasi kerentanan *heap overflow* pada proses Bluetooth Server. Hal ini mengharuskan penyerang memiliki perangkat yang telah dipasangkan (*paired*) dengan sistem Bluetooth kendaraan yang dapat dicapai melalui ponsel yang telah disusupi dan terhubung ke kendaraan, atau melalui akses fisik ke layar sentuh kendaraan tersebut. Berikut adalah hal yang terjadi pada fase ini:

- **Eksploitasi *heap overflow* pada Bluetooth**: Ditemukan sebuah *heap overflow* dalam fungsi penguraian (*parsing*) pada *Basic Imaging Profile* (BIP) di dalam Bluetooth Server. Profil ini memungkinkan ponsel untuk mengirimkan gambar sampul (*cover art*) agar ditampilkan di layar saat musik diputar. Akibat adanya bug implementasi pada fungsi penguraian tersebut, peneliti dapat mengirimkan berkas XML yang dirancang khusus dengan jumlah tag yang berlebihan (melebihi batas yang ditentukan). Hal ini memicu *heap overflow* yang menyebabkan penulisan data di luar batas buffer yang telah dialokasikan.
- **Arbitrary write primitive dan eksekusi kode di dalam Bluetooth Server**: Dengan mengeksploitasi *heap allocator* kustom yang digunakan oleh Bluetooth Server, penyerang memperoleh kemampuan untuk melakukan penulisan data ke lokasi memori mana pun (*arbitrary write primitive*). Karena tidak adanya dukungan ASLR (*Address Space Layout Randomization*) dan PIE (*Position Independent Execution*), alamat memori yang perlu ditimpa bersifat tetap, sehingga memudahkan pekerjaan penyerang. Mereka menimpa variabel global yang menyimpan penunjuk fungsi (*function pointer*) yang digunakan dalam *callback* pengatur waktu (*timer*) di dalam loop utama aplikasi. Dengan mengendalikan penunjuk fungsi ini, mereka mengalihkan alur eksekusi ke kode yang telah mereka sisipkan, sehingga berhasil menjalankan kode di dalam proses Bluetooth Server.

Fase 2: Eskalasi Hak Akses ke *Firmware* Wi-Fi

Pada fase 2, peneliti menemukan bahwa sistem infotainment tersebut telah diamankan dengan ketat menggunakan sandboxing: Kafel untuk penyaringan syscall, AppArmor untuk kontrol akses wajib (*mandatory access control*), dan Minijail untuk isolasi sistem berkas. Akibatnya, jika proses tersebut berhasil disusupi, penyerang hanya terbatas pada hak akses

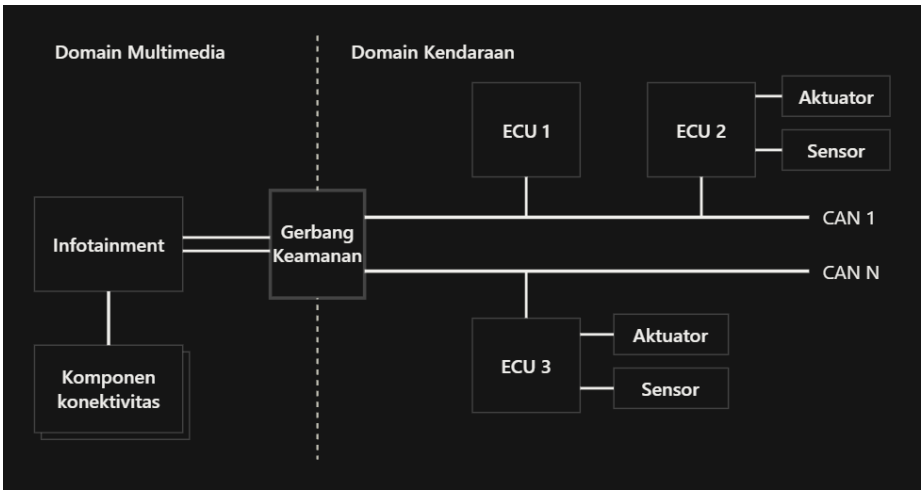
dan sumber daya yang dimiliki oleh proses tersebut. Mengingat proses Bluetooth Server beroperasi dengan hak akses terbatas dan telah diisolasi dengan baik, penyerang berupaya meningkatkan hak akses mereka hingga ke tingkat kernel. Mereka memanfaatkan memori DRAM yang digunakan bersama dengan chipset Wi-Fi/Bluetooth, serta menggunakan perintah *Host Controller Interface* (HCI) khusus dari vendor untuk menulis data secara langsung ke dalam memori *firmware* Wi-Fi. Hal ini memungkinkan mereka untuk menyisipkan kode ke dalam modul Wi-Fi melalui Bluetooth Server. Tindakan-tindakan berikut terjadi pada fase ini:

- **Eksekusi kode dalam *firmware* Wi-Fi**: Dengan menyuntikkan kode mereka ke dalam wilayah memori yang tidak terpakai pada *firmware* Wi-Fi dan memodifikasi (*patch*) tugas *idle* (diam) *firmware* tersebut agar beralih ke kode mereka, para penyerang berhasil menjalankan kode di dalam *firmware* Wi-Fi. Posisi ini memberikan pijakan yang lebih menguntungkan, karena *firmware* Wi-Fi berinteraksi secara erat dengan driver kernel sistem inang melalui antarmuka PCI Express, dengan memanfaatkan mekanisme *Direct Memory Access* (DMA) dan komunikasi mailbox. Bisa dikatakan, mereka harus bergerak secara lateral ke luar batasan SoC menuju chipset Wi-Fi, hanya untuk kemudian menemukan jalan kembali masuk ke dalam kernel Linux pada sistem infotainment tersebut!
- **Eksploitasi kerentanan driver Wi-Fi**: Penyerang menemukan kerentanan *out-of-bounds write* (penulisan di luar batas memori) dalam cara driver Wi-Fi Linux menangani struktur data tertentu yang digunakan bersama antara *firmware* dan driver. Dengan memanipulasi struktur-struktur tersebut melalui kode yang mereka jalankan di dalam *firmware* Wi-Fi, mereka memaksa driver (bcmdhd) untuk menulis data melampaui batas memori yang seharusnya di dalam ruang memori kernel.
- **Eksekusi kode kernel**: Kerentanan *out-of-bounds write* tersebut memungkinkan penyerang untuk menimpa data di dalam memori kernel, khususnya dengan menyasar *stack kernel* dari proses-proses yang mereka kendalikan. Dengan menimpa alamat kembalian (*return address*) pada stack, mereka menjalankan rangkaian *Return-Oriented Programming* (ROP) dalam konteks kernel, yang pada akhirnya memungkinkan eksekusi kode arbitrer (sewenang-wenang) di tingkat kernel.
- **Membangun root shell tanpa batasan**: Dengan kemampuan mengeksekusi kode di dalam kernel, penyerang meluncurkan proses tingkat root yang tidak terikat oleh batasan-batasan dari lingkungan sandbox awal. Hal ini memberi mereka kendali penuh atas sistem *infotainment*, termasuk kemampuan untuk menjalankan perintah apa pun dan mengakses data sensitif.

Tahap 3: Beralih ke Security *Gateway*

Sampai di titik ini, seorang peneliti keamanan mungkin sudah merasa puas dengan hasilnya, namun tidak demikian dengan kelompok penyerang ini! Oleh karena itu, mereka melanjutkan ke tahap 3 dari rangkaian serangan ini, di mana mereka berupaya untuk menyusupi *gateway* seperti yang ditunjukkan pada Gambar 8.9 guna menjangkau domain kendaraan melalui bus CAN. Dari *root shell* tanpa batasan pada sistem infotainment, penyerang berkomunikasi dengan Security *Gateway* melalui jaringan Ethernet internal.

Security Gateway merupakan sebuah MCU (*Microcontroller Unit*) yang menjalankan sistem operasi waktu nyata (*real-time operating system*/FreeRTOS) dan berfungsi sebagai perantara komunikasi antara sistem infotainment dengan komponen-komponen krusial kendaraan melalui bus CAN. Teknik-teknik serangan selanjutnya mungkin terdengar familier, karena kita telah membahas pola serangan serupa pada bab sebelumnya mengenai ECU berbasis MCU. Hal ini menunjukkan bahwa dalam kasus tertentu, penyerang harus menyusupi baik HPC (*High-Performance Computer*) maupun ECU berbasis MCU untuk mencapai tujuan serangan mereka.



Gambar 8.9 Gerbang keamanan (*security gateway*) memisahkan domain multimedia dari domain di dalam kendaraan (*in-vehicle domain*) tempat fungsi-fungsi yang krusial bagi keselamatan berada.

- **Eksploitasi kelemahan logika pada Gerbang Keamanan**: Penyerang mengeksploitasi kelemahan logika dalam mekanisme pembaruan *firmware* pada gerbang keamanan. Secara khusus, mereka memanipulasi proses pembaruan untuk melewati pemeriksaan verifikasi tanda tangan (*signature verification*). Mereka memulai proses pembaruan *firmware* yang sah yang berhasil lolos verifikasi tanda tangan lalu menjadwalkan pembaruan tersebut. Sebelum pembaruan diterapkan, mereka mengganti file *firmware* pada kartu SD di dalam Gerbang Keamanan dengan file berbahaya. Akibat adanya celah dalam proses pembaruan, tanda tangan dari pembaruan yang telah dijadwalkan tersebut tidak diverifikasi ulang sebelum diterapkan, sehingga memungkinkan *firmware* berbahaya untuk dimuat.
- **Eksekusi kode di dalam Gerbang Keamanan**: Dengan mengunggah *bootloader* yang telah dimodifikasi yang menonaktifkan mekanisme *secure boot* berbasis perangkat lunak penyerang berhasil mengeksekusi kode di dalam Gerbang Keamanan. Mereka menyisipkan kode mereka ke dalam binary pembaru (*updater*), serta memanfaatkan fungsi-fungsi yang sudah ada untuk berinteraksi dengan bus CAN (*CAN bus*) kendaraan.

- **Memahami mekanisme pembaruan *firmware* pada Gerbang Keamanan**: Gerbang Keamanan beroperasi dalam beberapa mode, dengan mode-mode utama sebagai berikut:
  - ***Mode Bootloader***: Bertanggung jawab untuk inisialisasi sistem saat pertama kali dinyalakan, menjalankan *secure boot* berbasis perangkat lunak, serta melakukan transisi ke mode lainnya.
  - ***Mode Updater* (Pembaru)**: Digunakan untuk memperbarui *firmware*, baik untuk dirinya sendiri maupun untuk unit kontrol elektronik (ECU) lain di dalam kendaraan. Mode ini dapat mengambil file *firmware*, memverifikasi tanda tangannya, dan menerapkan pembaruan.
  - **Mode aplikasi utama (*Main application mode*)**: Menangani operasional rutin, termasuk memfilter pesan CAN serta memfasilitasi komunikasi antara sistem infotainment dan berbagai komponen kendaraan.
- **Alur pembaruan *firmware***: Proses pembaruan *firmware* pada umumnya mencakup langkah-langkah berikut:
  1. **Inisiasi**: Sistem infotainment memulai pembaruan dengan mengunggah berkas biner pembaru (*updater binary*) ke kartu SD pada *Security Gateway* menggunakan protokol XMODEM.
  2. **Mulai ulang untuk pembaruan (*reboot for update*)**: Sistem *infotainment* mengirimkan perintah "mulai ulang untuk pembaruan" melalui UDP ke *Security Gateway*.
  3. **Verifikasi dan eksekusi**: Saat memulai ulang, *bootloader* pada *Security Gateway* memverifikasi tanda tangan digital dari berkas biner pembaru tersebut; jika valid, *bootloader* akan mengeksekusinya.
  4. **Pengambilan *firmware***: Dalam mode pembaru (*Updater Mode*), *Security Gateway* mengambil berkas *firmware* ECU dari sistem *infotainment* menggunakan protokol TFTP (*Trivial File Transfer Protocol*).
  5. **Verifikasi tanda tangan**: Setiap berkas *firmware* yang diambil disimpan di kartu SD, kemudian tanda tangan digitalnya diverifikasi.
  6. **Penerapan pembaruan**: Jika verifikasi tanda tangan berhasil, pembaruan dijadwalkan dan pada akhirnya diterapkan ke ECU terkait.
- **Memanfaatkan kelemahan logika**: Penyerang menemukan kerentanan *Time-of-Check to Time-of-Use* (TOCTOU) dalam proses pembaruan *firmware*:
  - ***Time-of-Check* (Waktu Pemeriksaan)**: *Security Gateway* memverifikasi tanda tangan digital berkas *firmware* segera setelah mengambilnya melalui TFTP dari sistem infotainment.
  - ***Time-of-Use* (Waktu Penggunaan)**: Berkas *firmware* tersebut kemudian digunakan pada tahap penerapan pembaruan.

Kelemahan logika ini muncul karena *Security Gateway* tidak memverifikasi ulang tanda tangan berkas *firmware* tepat sebelum menerapkan pembaruan. Hal ini menciptakan celah kesempatan di mana penyerang dapat mengganti berkas *firmware* yang sah dan bertanda

tangan dengan berkas berbahaya di kartu SD setelah verifikasi tanda tangan awal dilakukan, namun sebelum pembaruan diterapkan.

Langkah-Langkah Terperinci Eksploitasi

Setelah membahas fase-fase dalam rantai serangan, mari kita lihat bagaimana penyerang menjalankan eksploitasi tersebut serta langkah-langkah yang mereka ambil:

1. **Memulai pembaruan *firmware* yang sah**: Penyerang, yang telah memiliki akses root ke sistem *infotainment* dari langkah-langkah sebelumnya, memulai proses pembaruan *firmware* untuk sebuah ECU melalui *Security Gateway*. Sistem infotainment kemudian mengirimkan berkas *firmware* yang sah dengan tanda tangan digital yang valid ke *Security Gateway* menggunakan protokol TFTP.
2. **Verifikasi tanda tangan dan penjadwalan**: *Security Gateway* mengambil berkas *firmware*, menyimpannya di kartu SD, dan memverifikasi tanda tangannya. Setelah verifikasi berhasil, pembaruan dijadwalkan untuk diterapkan pada fase pembaruan berikutnya.
3. **Penggantian berkas *firmware***: Sebelum pembaruan diterapkan, penyerang mengganti berkas *firmware* yang sah di kartu SD dengan berkas berbahaya. Mereka memanfaatkan fakta bahwa *Security Gateway* tidak melindungi konten kartu SD dari modifikasi yang tidak sah oleh sistem infotainment. Penyerang menggunakan hak akses root mereka pada sistem infotainment untuk menimpa berkas *firmware* tersebut dengan versi mereka sendiri, yang berisi kode berbahaya.
4. **Penerapan *firmware* berbahaya**: Selama fase pembaruan, *Security Gateway* menerapkan pembaruan *firmware* ke ECU menggunakan berkas *firmware* yang ada di kartu SD. Yang krusial, sistem ini tidak memverifikasi ulang tanda tangan digital berkas *firmware* tersebut sebelum penerapannya. Akibatnya, *firmware* berbahaya diterapkan ke ECU tanpa terdeteksi.
5. **Penonaktifan secure boot berbasis perangkat lunak**: Penyerang menargetkan *firmware* milik *Security Gateway* itu sendiri, termasuk *bootloader* yang bertanggung jawab untuk melakukan pemeriksaan *secure boot* berbasis perangkat lunak. Dengan menggunakan metode yang sama, mereka mengunggah *bootloader* yang telah dimodifikasi untuk menonaktifkan mekanisme *secure boot*. *Bootloader* yang dimodifikasi ini memungkinkan eksekusi kode yang tidak ditandatangani atau kode berbahaya pada Security *Gateway*.
6. **Mendapatkan akses eksekusi kode pada Security *Gateway***: Dengan *secure boot* yang dinonaktifkan, penyerang kini dapat mengeksekusi kode *arbitrer* (sewenang-wenang) pada *Security Gateway*. Mereka menyisipkan kode mereka ke dalam binary pembaru (*updater*) atau membuat binary pembaru kustom yang menyertakan payload mereka. Saat *Security Gateway* melakukan reboot dan memuat binary pembaru dari kartu SD, sistem tersebut akan mengeksekusi kode milik penyerang.
7. **Memanfaatkan fungsi yang sudah ada**: Kode yang disisipkan oleh penyerang menggunakan kembali fungsi-fungsi yang sudah ada di dalam *firmware Security Gateway* untuk berinteraksi dengan bus CAN kendaraan. Dengan memanggil fungsi-

fungsi ini, mereka dapat mengirim pesan CAN arbitrer ke ECU lain di dalam kendaraan tersebut.

Dampak

Eksploitasi yang berhasil terhadap kerentanan-kerentanan ini menunjukkan rangkaian serangan canggih yang terdiri dari beberapa tahap berawal dari perangkat Bluetooth yang terhubung, berlanjut ke eksekusi di tingkat kernel, dan akhirnya mengompromikan sistem inti kendaraan. Implikasi dari rangkaian serangan ini sangat signifikan:

- **Kendali tidak sah atas fungsi kendaraan**: Dengan mengirimkan pesan CAN arbitrer, penyerang dapat memanipulasi sistem kendaraan yang krusial, yang berpotensi memengaruhi fitur keselamatan serta pengoperasian kendaraan.
- **Risiko privasi data**: Akses ke sistem infotainment dapat mengekspos data pribadi, termasuk kontak, pesan, riwayat panggilan, dan riwayat navigasi.
- **Kompromi sistem yang persisten**: Modifikasi pada komponen *firmware* seperti *bootloader* dan *Security Gateway* dapat memungkinkan penyerang untuk tetap berada di dalam sistem kendaraan secara permanen, bahkan setelah kendaraan dimulai ulang (*reboot*) atau melalui mekanisme pembaruan standar.

Jika di awal bab ini Anda masih meragukan kemungkinan teknis serangan jarak jauh yang dapat memengaruhi keselamatan kendaraan, kini Anda seharusnya telah memiliki pemahaman yang kuat mengenai apa saja yang mungkin dilakukan, serta bagaimana penyerang yang terampil merangkai berbagai kerentanan untuk mencapai tujuan mereka.

Studi Kasus Tambahan

Setelah membaca studi kasus kami, Anda dipersilakan untuk menganalisis sendiri rangkaian studi kasus berikut guna memahami bagaimana kerentanan ditemukan, bagaimana kerentanan tersebut dieksploitasi secara terpisah, serta bagaimana kerentanan-kerentanan itu digabungkan (dirantai) untuk mencapai tujuan penyerang. Dalam prosesnya, Anda juga diajak untuk memikirkan langkah-langkah pertahanan yang dapat mencegah masalah-masalah tersebut agar tidak bisa dieksploitasi sejak awal:

Tabel 8.1 Bahan bacaan dan materi studi tambahan

Topic	Links
bd-jb: Blu-ray Disc Java Sandbox Escape	https://www.youtube.com/watch?v=jyyCOoWksbg
Next-Gen Exploitation: Exploring the PS5 Security Landscape by Specter	https://www.youtube.com/watch?v=HBFDjfmIUis
XBOX One Cybersecurity Concept	https://www.youtube.com/watch?v=U7VwtOrwceo

Topic	Links
Penetration Testing of In-Vehicle Infotainment Systems in Connected Vehicles	https://ieeexplore.ieee.org/document/10575976
Hyundai Head Unit Hacking	https://docs.google.com/presentation/d/14B_Jrxjz8zHF_HcdfbCyitxiQVvrvYQKZ60USY33rY8/edit#slide=id.p
SIMurai: Slicing Through the Complexity of SIM Card Security Research	https://www.usenix.org/system/files/usenixsecurity24-lisowski.pdf
BarraCUDA: Edge GPUs do Leak DNNWeights	https://www.usenix.org/conference/usenixsecurity25/presentation/horvath

8.7 PRAKTIK TERBAIK DAN STRATEGI PENGUATAN KEAMANAN HPC KENDARAAN

Sebelum membahas langkah-langkah penanggulangan secara spesifik, penting untuk menetapkan suatu kondisi batasan yang krusial. Kumpulan praktik terbaik yang berdiri sendiri tidak akan mampu mengamankan HPC (*High-Performance Computing*) pada kendaraan secara efektif, kecuali jika didasari oleh proses analisis ancaman dan penilaian risiko yang sistematis. Platform komputasi berkinerja tinggi terus berkembang seiring dengan diperkenalkannya aplikasi baru, jalur konektivitas, dan titik integrasi; setiap fitur baru akan mengubah lanskap kerentanan (*attack surface*). Memandang keamanan hanya sebagai daftar periksa (*checklist*) yang statis sering kali justru menciptakan rasa aman yang semu.

Meskipun demikian, prinsip-prinsip keamanan yang telah mapan seperti pertahanan berlapis (*defense in depth*), prinsip hak akses minimum (*least privilege*), dan prinsip fungsionalitas minimum (*least functionality*) tetap sangat relevan jika diterapkan sebagai bagian dari proses rekayasa yang terstruktur. Mari kita simak beberapa praktik terbaik yang dapat Anda adopsi untuk mengamankan HPC.

Meminimalkan Permukaan Serangan Pada Perangkat Lunak dan Antarmuka

Mengurangi fungsionalitas yang terekspos tetap menjadi salah satu strategi pertahanan yang paling efektif dan paling mudah diterapkan. Platform HPC (*High-Performance Computing*) otomotif pada dasarnya sangat kompleks; setiap layanan, daemon, tumpukan protokol (*protocol stack*), dan port yang terbuka akan menambah potensi titik masuk bagi serangan. Sebagaimana ditunjukkan dalam kasus 1, keberadaan mekanisme pembaruan berbasis USB yang terpicu secara otomatis (diaktifkan melalui file penanda `jci-autoupdate`) secara signifikan memperluas permukaan serangan sebelum autentikasi dan memungkinkan eksploitasi langsung melalui media penyimpanan eksternal.

Menonaktifkan pemicu pembaruan otomatis tanpa pengawasan serta membatasi inisiasi pembaruan hanya pada alur kerja diagnostik yang terautentikasi akan dapat

menghilangkan vektor eksploitasi ini sepenuhnya. Para insinyur pertahanan harus melakukan audit sistematis terhadap perangkat lunak HPC dan menghapus komponen apa pun yang tidak benar-benar diperlukan untuk pengoperasian kendaraan. Dalam praktiknya, ini mencakup penghapusan alat pengembangan, aplikasi contoh, lingkungan skrip, dan layanan penelusuran kesalahan (*debugging*) dari citra perangkat lunak (*image*) versi produksi.

Prinsip fungsionalitas minimum harus dibarengi dengan prinsip hak akses minimum (*least privilege*). Dalam kasus 2, layanan Bluetooth eksklusif (`btapp`) dijalankan dengan hak akses root. Menjalankan tumpukan Bluetooth dalam proses khusus yang tidak memiliki hak akses istimewa dengan batasan kapabilitas yang ketat akan mencegah satu celah keamanan memori berkembang menjadi kompromi sistem secara menyeluruh. Layanan yang harus tetap aktif sebaiknya dijalankan dengan hak akses seminimal mungkin yang diperlukan untuk menjalankan fungsinya. Sarana pendukung pengembangan, seperti server SSH atau *Android Debug Bridge*, tidak boleh ada dalam citra perangkat lunak versi produksi kecuali jika memang sangat diperlukan; bahkan dalam kondisi tersebut, aksesnya harus dibatasi dengan sangat ketat.

Kredensial bawaan (*default*) dan pintu belakang (*backdoor*) pengembangan merupakan jenis kerentanan dengan tingkat risiko yang sangat tinggi. Semua kredensial yang digunakan selama tahap persiapan atau pengujian harus dihapus atau diganti dengan kredensial yang unik untuk setiap perangkat sebelum penerapan (*deployment*). Jika platform HPC dipasok oleh pihak ketiga, produsen peralatan asli (OEM) wajib memastikan secara tegas bahwa tidak terdapat pintu belakang dari vendor maupun jalan pintas diagnostik di dalam *firmware* produksi.

Selain menghapus perangkat lunak yang tidak diperlukan, layanan-layanan yang memang dibutuhkan harus dikonfigurasi dengan batasan seketat mungkin. Sebagai contoh, membiarkan tumpukan (*stack*) Bluetooth mengekspos mode protokol yang kompleks serta logika penyusunan ulang data kepada pihak lain yang tidak terautentikasi akan memperluas area serangan dan meningkatkan kerentanan terhadap eksploitasi. Konfigurasi keamanan harus secara eksplisit menonaktifkan ekstensi protokol opsional dan fitur-fitur eksperimental dalam versi produksi.

Lebih lanjut, jika layanan web diperlukan untuk antarmuka pengguna di dalam kendaraan, layanan tersebut tidak boleh menerima koneksi jarak jauh (sebaiknya hanya mendukung inisiasi komunikasi keluar). Jika akses dari luar tidak dapat dihindari, maka autentikasi timbal balik (*mutual authentication*) harus diterapkan. Semua port yang terbuka (*listening ports*) dan layanan jaringan yang tidak diperlukan harus dinonaktifkan, sehingga meskipun penyerang berhasil mencapai antarmuka jaringan HPC, tidak ada titik masuk yang tidak diinginkan yang dapat dieksploitasi. Antarmuka fisik maupun logis harus dikelola dengan tingkat kedisiplinan yang sama. Port USB harus dinonaktifkan atau dibatasi hanya untuk perangkat yang telah dikenali dan terautentikasi, sementara antarmuka perangkat keras yang tidak digunakan harus dinonaktifkan.

Daftar komponen perangkat lunak (*Software Bill of Materials*/SBOM) dan Manajemen Kerentanan

Platform komputasi berkinerja tinggi (*High-Performance Computing*/HPC) modern sangat bergantung pada komponen perangkat lunak pihak ketiga, terutama dalam lingkungan berbasis Linux dan Android. Mengidentifikasi serta menghilangkan paket perangkat lunak yang rentan tetap menjadi salah satu cara paling efektif untuk meningkatkan standar keamanan pada sistem HPC. Kerentanan dalam *firmware* Wi-Fi dari vendor dan driver Linux (sebagai contoh, driver `bcmdhd` beserta interaksinya dengan memori *firmware* bersama) dapat menjadi faktor penentu dalam rantai serangan multi-tahap. Oleh karena itu, SBOM harus mencakup *firmware* chipset dan driver kernel, bukan sekadar paket di tingkat pengguna (*user-space*).

Semua komponen eksternal beserta versi spesifiknya harus dilacak, dan pemindaian kerentanan harus dilakukan secara berkelanjutan untuk mendeteksi CVE (*Common Vulnerabilities and Exposures*) yang telah diketahui. Selain itu, kesiapan untuk menerapkan perbaikan (*patch*) tidak boleh berhenti pada tahap rilis produksi. Platform HPC harus dirancang untuk mendukung pembaruan di lapangan (*in-field updates*) sepanjang masa operasional kendaraan. Hal ini menuntut dukungan jangka panjang untuk lingkungan pengembangan (*build environments*), infrastruktur penandatanganan digital, dan alur pembaruan (*updating pipelines*). Menjaga kemampuan untuk membangun ulang dan memperbaiki platform HPC bertahun-tahun setelah dirilis merupakan komitmen organisasi dan teknis yang cukup berat.

Membangun *Hardware Root of Trust* dan *Secure Boot*

Keamanan perangkat keras menjadi fondasi bagi semua mekanisme pertahanan di tingkat yang lebih tinggi. Pertahanan perangkat lunak secanggih apa pun tidak akan memadai jika perangkat keras dasarnya sendiri penuh dengan kelemahan keamanan. Pada tingkat paling dasar, arsitektur *secure boot* (proses booting yang aman) yang tahan terhadap manipulasi (*tamper-resistant*) sangat diperlukan untuk membangun rantai kepercayaan (*chain of trust*) perangkat lunak. *Secure boot* menegakkan rantai kepercayaan tersebut mulai dari instruksi pertama yang dieksekusi setelah reset, melalui seluruh tahapan *bootloader*, hingga ke sistem operasi dan perangkat lunak aplikasi. Boot ROM yang tidak dapat diubah (*immutable*), yang tertanam di dalam SoC (*System-on-Chip*), berfungsi sebagai *Root of Trust* awal dan memverifikasi tanda tangan digital dari tahapan boot berikutnya sebelum dieksekusi.

Setiap tahapan selanjutnya memverifikasi integritas dan keaslian tahapan berikutnya. Jika ada komponen yang gagal dalam verifikasi, eksekusi harus segera dihentikan. Konsekuensi dari tidak diterapkannya *secure boot* pada SoC aplikasi dapat terlihat jelas: rantai bootstrap yang tidak melakukan autentikasi terhadap komponen boot memungkinkan instalasi perangkat lunak berbahaya (*malware*) yang bersifat persisten (menetap) begitu akses tulis ke penyimpanan berhasil diperoleh. Rantai *secure boot* yang berakar pada perangkat keras (*hardware-anchored*) akan mampu mencegah persistensi pasca-eksploitasi dengan cara memastikan integritas dari *bootloader* dan citra (*image*) sistem operasi.

Mekanisme *Device Identifier Composition Engine* (DICE) semakin banyak dipilih untuk mengimplementasikan *secure boot* (proses booting yang aman) dan penyediaan identitas

(*identity provisioning*). Alih-alih hanya mengandalkan kunci perangkat statis, DICE menghasilkan identitas kriptografis yang unik bagi setiap perangkat berdasarkan pengukuran status perangkat keras dan perangkat lunak pada setiap tahapannya. Konstruksi identitas berlapis ini memperkuat ketahanan terhadap serangan kloning maupun penggantian *firmware*.

Jika memungkinkan, *secure boot* sebaiknya dilengkapi dengan *remote attestation* (atestasi jarak jauh). Mengingat *platform High-Performance Computing* (HPC) pada kendaraan biasanya terhubung ke layanan *backend*, akses ke *backend* tersebut harus bergantung pada klaim yang dapat diverifikasi mengenai status keamanan platform. Atestasi membatasi kemampuan kendaraan yang telah disusupi untuk berinteraksi dengan infrastruktur *backend*, sekaligus mengurangi risiko dampak berskala luas yang dapat memengaruhi seluruh armada kendaraan.

Selain itu, antarmuka debug perangkat keras dan antarmuka pengembang harus dikunci sebelum perangkat diterapkan (*deployment*). Antarmuka seperti JTAG yang terbuka, konsol UART, atau porta debug USB akan sangat mempermudah ekstraksi *firmware* dan analisis secara offline. Oleh karena itu, antarmuka-antarmuka ini harus dinonaktifkan atau dilindungi menggunakan autentikasi kriptografis yang kuat serta kredensial yang unik untuk setiap perangkat.

Penggunaan Lingkungan Eksekusi Tepercaya (*Trusted Execution Environment*/TEE) dan Subsistem Keamanan Terisolasi

Meskipun telah memiliki rantai boot yang aman dan antarmuka perangkat keras yang terkunci, pihak pertahanan harus mengantisipasi bahwa kerentanan perangkat lunak pada akhirnya dapat memungkinkan penyerang untuk mengeksekusi kode di dalam sistem operasi utama. Oleh karena itu, layanan keamanan krusial dan data rahasia yang bersifat jangka panjang harus diisolasi dari lingkungan eksekusi utama. Platform komputasi berkinerja tinggi (*High-Performance Computing*/HPC) pada kendaraan umumnya mengandalkan *Trusted Execution Environment* (TEE), seperti Arm TrustZone, untuk menjalankan layanan sensitif termasuk operasi kriptografi, penyimpanan kunci, dan agen atestasi. Aplikasi yang berjalan di lingkungan umum (*normal world*) hanya dapat mengakses layanan ini melalui antarmuka yang telah divalidasi dan memiliki cakupan akses yang sangat terbatas.

Banyak *System-on-Chip* (SoC) otomotif juga mengintegrasikan subsistem keamanan khusus atau *Hardware Security Module* (HSM). Standar SAE J3101 meresmikan model ini sebagai *Hardware Protected Security Environment* (HPSE). Sebuah HPSE biasanya dilengkapi dengan prosesor, penyimpanan aman, dan *firmware* tersendiri, serta menjalankan fungsi-fungsi keamanan krusial seperti validasi *secure boot*, verifikasi tanda tangan, dan manajemen kunci secara independen dari prosesor aplikasi utama. Mengisolasi data rahasia di dalam HPSE akan mengurangi risiko secara signifikan jika sistem operasi utama berhasil disusupi. Kunci yang disimpan di dalam HPSE tidak dapat dibaca melalui akses memori CPU utama, penyalahgunaan DMA (*Direct Memory Access*), ataupun antarmuka debug.

TEE dan HSM juga memungkinkan layanan pemantauan dan atestasi yang terisolasi. Agen pengukuran integritas yang berjalan di lingkungan terlindungi dapat mendeteksi adanya

manipulasi pada kernel atau *firmware* sesuatu yang akan sulit disembunyikan jika sistem operasi telah dikuasai oleh penyerang. Desain berlapis ini merupakan perwujudan praktis dari strategi pertahanan berlapis (*defense in depth*).

Partisi Aman Dalam Arsitektur HPC Berbasis Hipervisor

Ketika hipervisor Tipe-1 digunakan untuk menampung berbagai domain kendaraan pada satu platform HPC, hipervisor tersebut menjadi batasan keamanan yang krusial. Hipervisor itu sendiri harus diminimalkan dan diperkuat (*di-hardened*). Hanya fungsi virtualisasi esensial yang boleh dijalankan pada tingkat hak akses tertinggi. Layanan opsional, fitur debug, dan komponen manajemen harus dinonaktifkan dalam versi produksi.

Selain itu, setiap sistem operasi tamu (*guest OS*) harus diisolasi secara ketat, baik dalam hal akses memori maupun akses perangkat. Mekanisme virtualisasi perangkat keras dan penerjemahan alamat tahap kedua (*second-stage address translation*) harus mencegah satu mesin virtual membaca atau memodifikasi memori milik mesin virtual lainnya. Isolasi periferal juga sama pentingnya; periferal tersebut harus ditetapkan secara statis kepada masing-masing tamu. Jika berbagi perangkat tidak dapat dihindari, IOMMU yang pada sistem Arm disebut sebagai SMMU harus diaktifkan dan dikonfigurasi dengan benar. Selain itu, perlindungan DMA sangat penting untuk mencegah tamu yang telah disusupi atau *firmware* perangkat mengakses memori di luar wilayah yang telah ditetapkan untuknya.

Konfigurasi hipervisor harus bersifat tidak dapat diubah (*immutable*) setelah proses booting, jika memungkinkan. Tamu tidak boleh diizinkan untuk memperluas alokasi memori secara dinamis atau memperoleh akses ke periferal tambahan. Batasan isolasi harus ditetapkan sepenuhnya sebelum tamu mana pun dijalankan, dan kebijakan hipervisor yang terperinci harus dikonfigurasi menggunakan pengaturan paling ketat yang tetap memenuhi persyaratan fungsional. Partisi yang kuat seperti ini memastikan bahwa jika domain yang kurang kritis seperti sistem infotainment atau mesin virtual Android disusupi, penyerang tidak dapat dengan mudah melakukan pergerakan lateral (*lateral movement*) ke domain yang kritis bagi keselamatan (*safety-critical*) atau domain kendali kendaraan.

Perlindungan Memori dan Mitigasi Risiko Mikroarsitektur

Di dalam setiap sistem operasi tamu, teknik penguatan (*hardening*) OS standar harus diterapkan untuk mencegah satu proses pengguna yang disusupi mengambil alih seluruh mesin virtual tersebut. Langkah-langkah yang umum dilakukan meliputi isolasi proses yang ketat, penetapan wilayah memori sebagai non-eksekusi (*non-executable*), serta pemetaan akses baca-saja (*read-only*) untuk struktur data kritis, guna meningkatkan kesulitan dalam melakukan eksploitasi.

Fitur penguatan pada tahap kompilasi maupun saat *runtime* seperti *Address Space Layout Randomization* (ASLR), *Pointer Authentication Codes* (PAC), *Branch Target Identification* (BTI), dan *stack canaries* harus diaktifkan untuk mempersulit serangan terhadap alur kendali (*control flow attacks*). 2. ASLR saja tidaklah cukup karena rangkaian eksploitasi tersebut menggabungkan kerusakan memori (*memory corruption*) dengan primitif pembacaan arbitrer (*arbitrary read primitive*) untuk mengungkap alamat heap, pustaka (*library*), dan stack. Hal ini menunjukkan mengapa ASLR harus dikombinasikan dengan perlindungan alur kendali (*control-*

flow) dan integritas penunjuk (*pointer-integrity*) yang lebih kuat seperti PAC dan BTI pada platform yang mendukungnya. Selain itu, untuk lingkungan berbasis Linux, mekanisme kontrol akses wajib (*mandatory access control*) seperti SELinux dan isolasi namespace dapat semakin membatasi akses sumber daya bagi aplikasi maupun kontainer.

Selain itu, beberapa SoC otomotif kini menyediakan enkripsi memori transparan untuk DRAM eksternal. Mengenkripsi konten memori pada pengontrol memori dapat memitigasi serangan fisik, seperti ekstraksi data melalui *cold-boot* atau pemeriksaan memori (*memory probing*), dan sebaiknya diterapkan kapan pun memungkinkan. Pihak pertahanan juga harus memperhitungkan serangan berbasis eksekusi spekulatif (*speculative execution attacks*), di mana risiko kebocoran informasi dianggap cukup kritis sehingga memerlukan beban pemrosesan tambahan untuk langkah-langkah mitigasinya. Dalam kasus seperti ini, pembaruan mikrokode CPU serta mitigasi pada sistem operasi untuk kerentanan kelas Spectre dan Meltdown harus diterapkan. Hal ini mencakup pengaktifan mekanisme seperti *Kernel Page Table Isolation* (KPTI) dan penerapan *speculation barrier* yang sesuai. Jika virtualisasi digunakan, *hypervisor* harus memastikan bahwa *side channel* spekulatif tidak dapat dieksploitasi untuk meloloskan diri dari *isolasi guest* (sistem tamu). Jika dampak terhadap performa terlalu berat, penilaian berbasis risiko harus dilakukan untuk menentukan jalur kode kritis mana yang wajib dilindungi. Jelas bahwa dampak terhadap performa memerlukan evaluasi yang cermat, namun prinsip utamanya tetaplah bahwa kode yang tidak tepercaya tidak boleh dapat menyimpulkan informasi rahasia melalui status mikroarsitektur yang digunakan bersama.

Memperkuat Penanganan Input

Selain memperkuat sistem operasi tamu (*Guest OS*), semua input yang berasal dari luar atau lintas domain harus dianggap sebagai input yang tidak tepercaya. Ini mencakup nilai-nilai yang digunakan untuk menyusun jalur berkas (*file paths*), kueri basis data, dan perintah sistem operasi. Contoh konkret dari prinsip ini dapat dilihat pada ilustrasi yang diberikan. Pengidentifikasi perangkat yang tidak valid serta parameter arsip yang dikendalikan oleh penyerang sempat diteruskan secara langsung ke dalam operasi basis data dan penyusunan perintah shell. Hal ini memicu kerentanan injeksi SQL (*SQL injection*) dan berbagai kerentanan injeksi perintah di dalam komponen pembaruan yang memiliki hak akses istimewa.

Penanganan *input* harus diimplementasikan dengan menggunakan API dan pustaka yang menerapkan batasan serta parameterisasi pada *input*, guna menghilangkan kerentanan jenis injeksi. Logika penguraian (*parsing*) apa pun yang menjadi masukan bagi eksekusi perintah atau operasi dengan hak akses istimewa harus diuji secara menyeluruh untuk mendeteksi adanya kerentanan injeksi kode.

Pembaruan Perangkat di Lapangan yang Aman dan Perlindungan Rantai Pasok

Sementara *secure boot* (boot aman) memverifikasi perangkat lunak saat dijalankan, mekanisme pembaruan yang aman mencegah masuknya perangkat lunak berbahaya sejak awal. Kasus 3 menunjukkan kelemahan kritis dalam alur kerja pembaruan di *Security Gateway*: citra *firmware* diverifikasi saat diunduh, tetapi tidak diverifikasi ulang saat dipasang. Semua artefak pembaruan harus diautentikasi dan diverifikasi secara kriptografis sebelum konten

pembaruan apa pun diproses. Perhatian khusus harus diberikan pada pencegahan *rollback* (pengembalian ke versi sebelumnya) serta serangan *mix-and-match* (penggabungan komponen yang tidak kompatibel) dengan menggunakan tabel versi partisi yang ditandatangani secara kriptografis, yang mengikat pembaruan perangkat lunak satu sama lain.

Ketelitian dalam validasi input juga berlaku untuk mekanisme pembaruan perangkat lunak. Celah *time-of-check to time-of-use* (perbedaan waktu antara pemeriksaan dan penggunaan) yang dieksploitasi dalam Kasus 3 menunjukkan bahwa keamanan pembaruan tidak hanya bersifat kriptografis, tetapi juga transaksional. Status pembaruan, artefak, dan metadata harus dilindungi dari modifikasi yang mungkin terjadi di antara tahap verifikasi dan eksekusi.

Selain itu, *pull request* untuk pembaruan OTA tetap menjadi metode yang lebih disukai karena dapat menghilangkan kemungkinan server berbahaya mengirimkan pembaruan perangkat lunak yang telah dimanipulasi. Faktanya, alur pembaruan harus menerapkan pemeriksaan integritas di setiap tahap, dan tidak ada perintah yang berasal dari konten pembaruan boleh dijalankan kecuali keaslian serta integritasnya telah dipastikan. Dalam kasus ini, sistem infotainment bertindak sebagai sumber pembaruan dengan hak akses khusus (*privileged source*) bagi Security *Gateway* melalui jaringan internal.

Memperhatikan seluruh sumber pembaruan sangatlah penting, karena kita tidak bisa berasumsi bahwa sistem pembaruan di dalam kendaraan (*onboard updater*) memiliki tingkat kepercayaan yang lebih tinggi hanya karena ia memulai pembaruan dari dalam kendaraan itu sendiri. Faktanya, pengiriman pembaruan harus menerapkan autentikasi dan otorisasi timbal balik antar domain terlepas dari lokasi fisiknya sehingga domain multimedia yang telah disusupi tidak dapat bertindak sebagai sumber pembaruan yang tepercaya. Selain itu, strategi pembaruan yang aman dari kegagalan (*fail-safe*), seperti penggunaan rantai boot ganda atau redundan, diperlukan untuk memastikan bahwa kegagalan pembaruan tidak menyebabkan HPC (komputer berkinerja tinggi) menjadi tidak dapat beroperasi.

Terkait keamanan rantai pasok, aspek ini harus diperhatikan setara dengan keamanan perangkat keras dan perangkat lunak. Komponen harus diperoleh dari pemasok tepercaya dengan proses manufaktur dan distribusi yang terkendali. Perangkat lunak yang dikirimkan oleh pemasok dan mitra harus diverifikasi integritas serta asal-usulnya menggunakan mekanisme kriptografi. Selain itu, perlindungan terhadap infrastruktur penandatanganan dan pengendalian ketat atas kunci penandatanganan adalah hal yang wajib dilakukan. Pada dasarnya, setiap artefak yang masuk ke dalam ekosistem perangkat lunak kendaraan harus dapat diverifikasi dan dilacak.

8.8 EVALUASI ANCAMAN DAN STRATEGI PERTAHANAN KEAMANAN HPC KENDARAAN

HPC (*High-Performance Computer*) merepresentasikan konvergensi antara fungsi kendaraan yang krusial, pengalaman pengguna yang kaya, dan layanan yang terintegrasi dengan cloud ke dalam satu platform komputasi tunggal. Sebagaimana telah kita bahas sepanjang bab ini, keputusan arsitektural yang membuat HPC begitu tangguh juga membuatnya jauh lebih sulit untuk diamankan. Dengan beragam antarmuka perangkat keras,

ketergantungan pada sistem operasi komoditas, serta kebutuhan untuk menampung berbagai domain perangkat lunak dengan batasan keselamatan dan keamanan yang berbeda-beda, HPC menghadirkan permukaan serangan (*attack surface*) yang jauh lebih luas.

Melalui analisis mendalam terhadap komponen perangkat keras dan tumpukan perangkat lunak (*software stack*) yang berlapis, kita telah melihat bahwa kelemahan di kedua area tersebut tidak hanya lazim ditemukan, tetapi sering kali saling berkaitan. Sebagai contoh, port JTAG yang tidak terlindungi dengan baik atau *bootloader* yang tidak aman dapat menjadi titik awal untuk eskalasi hak akses, pergerakan lateral (*lateral movement*), bahkan pengambilalihan kendali penuh atas kendaraan. Kerentanan perangkat lunak mulai dari CVE pada kernel hingga layanan dengan hak akses berlebih (*overprivileged*) dan partisi *hypervisor* yang lemah menjadi jauh lebih berbahaya jika digabungkan dengan isolasi perangkat keras yang buruk atau konfigurasi bawaan (*default*) yang tidak aman.

Studi kasus, seperti yang melibatkan Mazda dan Tesla, memberikan bukti nyata mengenai metode dan kelemahan yang telah dibahas. Setiap kasus menunjukkan bahwa kelalaian yang tampak sepele seperti jalur *firmware* yang tidak ditandatangani secara digital (*unsigned firmware path*), antarmuka debug yang salah konfigurasi, atau sanitasi input yang tidak memadai dapat dirangkaikan menjadi serangan yang canggih, persisten, dan berdampak besar. Dengan mengadopsi pola pikir seorang penyerang, kita telah menelaah tahapan-tahapan dalam sebuah kompromi sistem yang berhasil, mulai dari akses awal hingga upaya untuk mempertahankan keberadaan di dalam sistem (persistensi). Bagi para insinyur keamanan ofensif, hal ini dapat menjadi landasan untuk memahami dan menyusun rantai serangan (*attack chain*) yang canggih terhadap HPC di dunia nyata.

Dari sisi pertahanan, dengan menekankan pentingnya pendekatan yang didorong oleh analisis ancaman sistematis, kami telah menguraikan praktik-praktik terbaik untuk merancang dan memelihara HPC yang aman. Praktik-praktik ini mencakup penerapan *secure boot* dengan *hardware root of trust*, pemanfaatan TrustZone atau HPSE untuk mengisolasi materi kriptografis, penggunaan *hypervisor* yang diperkuat (*hardened hypervisor*) untuk memisahkan komponen dengan tingkat kekritisan yang berbeda, pengurangan permukaan serangan melalui prinsip hak akses minimum (*least privilege*), serta integrasi keamanan ke dalam siklus hidup pengembangan perangkat lunak. Yang terpenting, langkah-langkah pertahanan ini harus berfungsi sebagai lapisan keamanan berlapis (*defense-in-depth*), di mana setiap lapisan berkontribusi untuk mengurangi kemungkinan terjadinya pelanggaran keamanan yang merugikan. Mungkin wawasan yang paling menyadarkan dari bab ini adalah pemahaman bahwa penyerang sering kali berhasil bukan karena tidak adanya mekanisme keamanan, melainkan karena mekanisme tersebut tidak lengkap, salah dikonfigurasikan, atau diterapkan secara tidak konsisten.

Saat mengakhiri bab ini, kita memahami bahwa melindungi sistem komputasi berkinerja tinggi (HPC) berarti mengamankan sebuah platform yang kompleksitas dan paparan ancamannya setara dengan server tingkat perusahaan, namun dengan tantangan tambahan berupa akses fisik, dukungan sepanjang masa pakai yang panjang, serta integrasi keselamatan fungsional. Hal ini menuntut pola pikir lintas disiplin, pengujian berkelanjutan, dan yang

terpenting, pemahaman mendalam mengenai teknik-teknik serangan terkini, agar kita tidak lengah akibat pola pikir yang hanya berfokus pada pertahanan.

BAB 9
KEAMANAN SENSOR DAN AI PADA KENDARAAN MODERN

Kendaraan yang terhubung (*connected vehicles*) bergantung pada sejumlah besar sensor yang memungkinkan kesadaran waktu nyata (*real-time*) baik terhadap kondisi kendaraan maupun lingkungan sekitarnya. Kendaraan modern umumnya mengintegrasikan puluhan sensor, dengan platform kelas atas bahkan menggunakan lebih dari 100 sensor seiring dengan diperkenalkannya fitur-fitur yang lebih canggih. Sensor-sensor ini mencakup setiap aspek fungsionalitas kendaraan. Unit kendali mesin dan sistem penggerak (*powertrain*) mengandalkan sensor yang tertanam secara mendalam seperti sensor suhu, tekanan, dan aliran udara untuk mengoptimalkan kinerja, efisiensi, serta emisi. Sistem bodi dan fitur keselamatan menggunakan sensor untuk berbagai keperluan, mulai dari pengembangan kantung udara (*airbag*) hingga kenyamanan kabin.

Sistem Bantuan Pengemudi Lanjutan (*Advanced Driver Assistance Systems*/ADAS) menghadirkan serangkaian sensor persepsi eksternal termasuk kamera, radar, LiDAR, dan sensor ultrasonic untuk mendeteksi hambatan, lajur jalan, dan rambu lalu lintas di berbagai tingkat otomatisasi mengemudi. Selain sensor konvensional, kendaraan modern kini semakin banyak mengintegrasikan sistem berbasis kecerdasan buatan (AI) dan pembelajaran mesin (*machine learning*/ML), termasuk asisten dalam kendaraan berbasis model bahasa besar (*Large Language Model*/LLM) yang memproses perintah suara serta sumber data eksternal untuk menyediakan navigasi, komunikasi, dan kemampuan pengendalian kendaraan.

Mengingat sensor dan sistem AI merupakan fondasi bagi keselamatan dan fungsionalitas kendaraan, setiap gangguan atau manipulasi terhadap input maupun output-nya dapat menimbulkan konsekuensi serius. Sensor ADAS yang dimanipulasi (*spoofed*) dapat menyebabkan kendaraan salah menilai hambatan atau kondisi jalan, sementara asisten AI dalam kendaraan yang telah disusupi dapat dipicu untuk menjalankan fungsi kendaraan tanpa otorisasi dari pengguna yang sah. Seiring dengan pergerakan kendaraan menuju tingkat otonomi yang lebih tinggi, integritas input sensor dan perilaku sistem AI menjadi semakin krusial.

Bab ini membahas sensor kendaraan dan sistem AI dari perspektif keamanan ofensif. Kami menelaah arsitektur berbagai jenis sensor yang terdapat pada kendaraan modern, menganalisis kelemahan yang muncul dari sifat fisik, desain protokol, serta integrasinya ke dalam kendaraan, dan mendiskusikan teknik pengujian penetrasi (*penetration testing*). Lebih lanjut, kami menyajikan tiga studi kasus untuk mengilustrasikan konsep-konsep ini dalam praktik. Bab ini diakhiri dengan praktik terbaik dan panduan pertahanan untuk memperkecil permukaan serangan (*attack surface*) pada sensor, guna memastikan komponen-komponen vital ini tetap tangguh dalam menghadapi pihak yang berniat jahat. Dalam bab ini, kita akan membahas topik-topik utama berikut:

- Gambaran umum mengenai sensor kendaraan dan AI
- Memahami sensor kendaraan serta antarmuka AI

- Kelemahan umum pada sensor kendaraan dan sistem AI
- Teknik pengujian penetrasi (*penetration testing*) terhadap sensor kendaraan dan AI
- Studi kasus serangan nyata terhadap sensor kendaraan dan sistem AI/ML
- Praktik terbaik untuk mengamankan sensor kendaraan dan sistem AI

9.1 ARSITEKTUR DAN ANTARMUKA SENSOR KENDARAAN SERTA SISTEM AI

Sensor kendaraan memiliki tingkat kompleksitas yang sangat beragam. Beberapa sensor hanyalah transduser sederhana yang dihubungkan secara elektrik ke *Electronic Control Unit* (ECU) terdekat, sementara sensor lainnya merupakan modul lengkap yang dilengkapi dengan prosesor internal, penyimpanan, dan jalur data berkapasitas tinggi. Kendaraan penumpang yang lengkap dapat memuat puluhan elemen sensor mulai dari radar, kamera, LiDAR, ultrasonik, hingga teknologi berbasis tekanan yang masing-masing berkontribusi pada lapisan pemahaman lingkungan yang berbeda. Selain persepsi fisik, kendaraan kini semakin banyak mengintegrasikan asisten suara berbasis AI dan kecerdasan di dalam kabin yang mampu memproses perintah bahasa alami, mempersonalisasi pengalaman berkendara, serta berkomunikasi dengan layanan cloud secara waktu nyata (*real-time*). Setiap kemampuan ini memperluas fungsionalitas kendaraan, namun di saat yang sama juga memperluas celah kerentanan (*attack surface*) kendaraan tersebut.

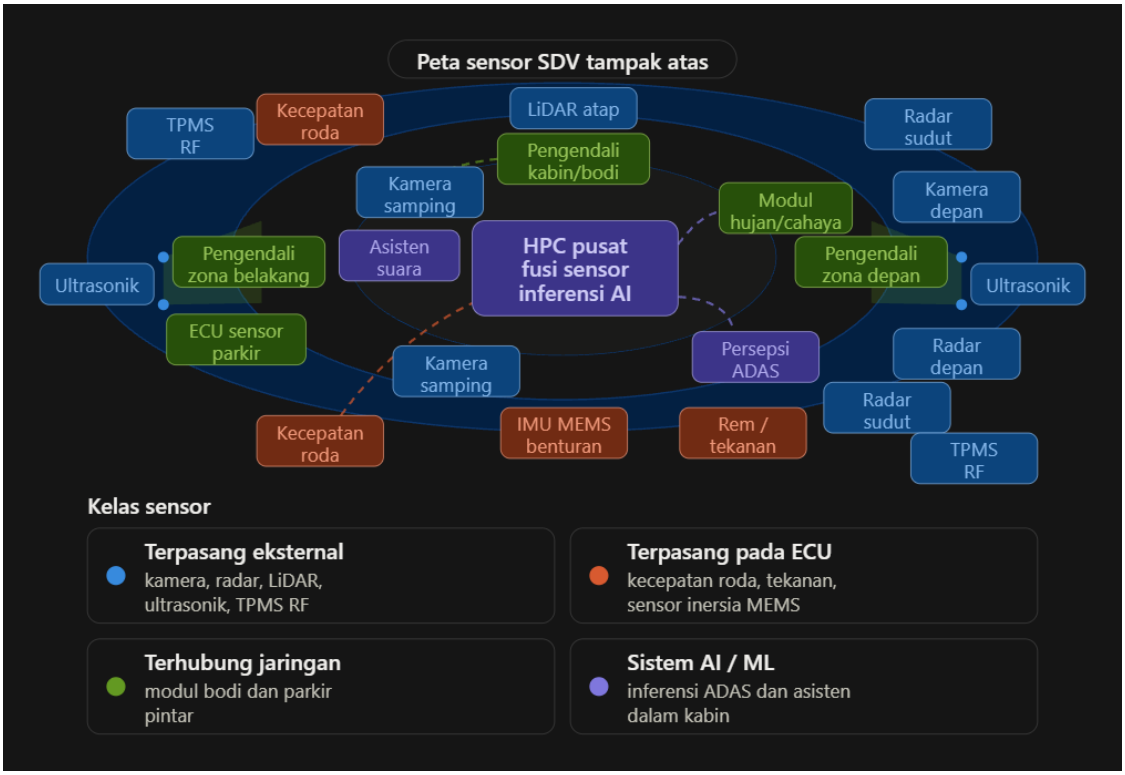
Terlepas dari tingkat kompleksitasnya, saat menganalisis keamanan sensor kendaraan, pertanyaan arsitektural yang muncul tetap sama: di mana proses penginderaan terjadi, di mana data diproses, dan bagaimana data tersebut ditransmisikan melalui arsitektur kelistrikan/elektronik (E/E) menuju fungsi-fungsi kontrol yang memanfaatkannya? Meskipun ECU biasanya menjadi target utama dalam pengujian penetrasi (*penetration testing*), dalam beberapa tahun terakhir, produsen sensor dan OEM (*Original Equipment Manufacturer*) mulai melibatkan penguji penetrasi. Tujuannya adalah untuk mengidentifikasi kerentanan sensor sebelum produk diluncurkan ke pasar, serta untuk mengungkap dampak yang mungkin terjadi pada bagian lain dari kendaraan jika satu atau lebih sensor berhasil disusupi. Perlu dicatat bahwa sensor kendaraan dan sistem AI menghadirkan tantangan keamanan unik yang melampaui apa yang biasanya ditemukan pada protokol jaringan internal kendaraan maupun ECU konvensional. Serangan terhadap sensor umumnya beroperasi pada lapisan fisik menggunakan sinyal elektromagnetik, cahaya, atau suara untuk menyuntikkan data palsu ke dalam sistem yang sejak awal tidak dirancang untuk menghadapi input yang bersifat jahat (*adversarial inputs*).

Sementara itu, serangan terhadap AI menargetkan proses inferensi itu sendiri, dengan merusak atau menyesatkan model yang menjadi dasar pengambilan keputusan otomatis. Secara keseluruhan, kategori-kategori serangan ini mewakili salah satu ranah yang berkembang paling pesat dalam keamanan siber otomotif. Sebelum mendalami metodologi pengujian penetrasi sensor, penting untuk menguraikan jenis-jenis sensor utama serta antarmuka kecerdasan buatan (AI) yang lazim ditemukan pada kendaraan modern yang terhubung dan otomatis, beserta anatomi perangkat keras dan perangkat lunak yang umum digunakan, serta pola konektivitas jaringannya.

9.2 KLASIFIKASI ARSITEKTUR DAN KARAKTERISTIK SENSOR KENDARAAN SERTA SISTEM AI

Sensor dan sistem AI dalam kendaraan modern mencakup beragam teknologi yang sangat berbeda dalam hal prinsip kerja fisik, karakteristik konektivitas, serta sifat celah keamanan (*attack surface*) yang ditimbulkannya. Untuk memahami lebih baik jenis sensor dan antarmuka AI yang ada pada kendaraan, kita dapat mengategorikannya berdasarkan integrasi arsitekturalnya ke dalam kendaraan. Kami menggunakan empat kategori untuk membedakannya: sensor yang terhubung ke ECU (*ECU-attached sensors*), sensor yang terhubung ke jaringan (*network-attached sensors*), sensor yang dipasang di bagian luar (*externally mounted sensors*), serta sistem AI dan ML.

Dua kategori pertama sensor yang terhubung ke ECU dan sensor yang terhubung ke jaringan memiliki celah keamanan yang tumpang tindih dengan ECU dan protokol komunikasi di dalam kendaraan. Serangan terhadap jenis sensor ini terutama menyasar *firmware* ECU, antarmuka diagnostik, atau melibatkan penyuntikan pesan (*message injection*) dan serangan pengulangan (*replay attacks*) pada tingkat jaringan kendaraan. Celah keamanan ini serta teknik pengujian penetrasi yang relevan telah dibahas dalam bab-bab mengenai keamanan ECU dan keamanan jaringan kendaraan. Alih-alih mengulang pembahasan tersebut, bab ini hanya mengulas secara singkat sensor yang terhubung ke ECU dan jaringan demi kelengkapan materi dengan merujuk pada pembahasan mendalam di bab-bab sebelumnya, sebelum berfokus secara mendalam pada dua kategori lainnya: sensor yang dipasang di bagian luar serta sistem AI dan ML. Kategori-kategori ini menghadirkan celah keamanan yang sangat berbeda dibandingkan dengan ECU tradisional maupun komponen yang terhubung ke jaringan. Sensor yang dipasang di bagian luar menciptakan celah keamanan pada domain sinyal fisik, yang untuk menilainya diperlukan peralatan serta pengetahuan khusus di bidang frekuensi radio (RF), optik, atau akustik. Sementara itu, sistem AI dan ML memperluas celah keamanan hingga ke alur inferensi (*inference pipeline*) dan rantai pasok model (*model supply chain*). Dalam kasus asisten berbasis model bahasa besar (LLM) di dalam kendaraan, celah keamanan juga mencakup input bahasa alami serta sumber data eksternal yang diizinkan untuk diproses oleh sistem tersebut. Gambaran umum mengenai jenis-jenis sensor kendaraan yang lazim ditemukan dalam *Software-Defined Vehicle* (SDV) disajikan pada Gambar 9.1.



Gambar 9.1 Kelas sensor kendaraan pada SDV (kendaraan yang didefinisikan oleh perangkat lunak) pada umumnya.

Mari kita beralih ke jenis sensor berikutnya.

Sensor yang Terhubung ke ECU dan Sensor yang Terhubung ke Jaringan

Sensor yang terhubung ke ECU (*ECU-attached sensors*) diintegrasikan secara fisik ke dalam unit kontrol elektronik (ECU) atau dihubungkan langsung ke modul pemrosesan sensor khusus. Contohnya meliputi sensor kecepatan roda, sensor posisi katup gas (*throttle*), sensor sudut kemudi, dan sensor benturan kantung udara (*airbag*). ECU mengendalikan antarmuka perangkat keras ke sensor dan memproses sinyal mentah secara internal sebelum hasilnya disediakan untuk jaringan kendaraan. Karena data sensor dihasilkan dan digunakan di dalam batasan pemrosesan ECU itu sendiri, serangan terhadap sensor jenis ini umumnya dilakukan melalui ECU tersebut misalnya dengan menargetkan *firmware* melalui antarmuka diagnostik, mengeksploitasi kelemahan keamanan memori dalam kode pemrosesan sinyal, atau memanipulasi sinyal fisik sensor pada tingkat konektor. Kategori serangan ini dibahas dalam konteks keamanan ECU pada bagian lain.

Satu kategori sensor yang terhubung ke ECU perlu mendapat perhatian khusus dalam bab ini: sensor inersia berbasis sistem mikro-elektromekanikal (MEMS), termasuk akselerometer dan giroskop. Sensor-sensor ini digunakan dalam fungsi seperti kontrol stabilitas elektronik, pengembangan kantung udara, dan navigasi inersia. Meskipun keluaran digitalnya diproses di dalam ECU, sensor-sensor ini rentan terhadap manipulasi melalui stimulus fisik eksternal yang diarahkan langsung ke sensor tersebut. Hal ini menempatkan sensor-sensor ini ke dalam kelas ancaman yang sama dengan sensor yang menghadap ke luar (*eksternal*), yang akan dibahas pada bagian-bagian selanjutnya.

Sensor yang terhubung ke jaringan (*network-attached sensors*) merupakan modul mandiri yang menghasilkan pengukuran terolah dan mengirimkannya melalui bus jaringan di dalam kendaraan. Contohnya mencakup sensor jarak parkir dan sensor bodi seperti modul pendeteksi hujan dan cahaya yang mengirimkan data pengukuran melalui *Local Interconnect Network* (LIN) ke satu ECU pengendali (misalnya, modul kontrol bodi). Contoh lainnya adalah sensor kecepatan roda dan sensor sudut kemudi yang menyiarkan sinyalnya melalui *Controller Area Network* (CAN) ke beberapa ECU secara bersamaan. Dari perspektif keamanan ofensif, sensor yang terhubung ke jaringan dianggap sebagai simpul (*node*) jaringan yang dapat dianalisis untuk mencari kelemahan autentikasi dalam komunikasi pesannya, kerentanan pada antarmuka diagnostik, serta kerentanan terhadap serangan *denial-of-service* atau serangan replay pada bus tersebut. Pertimbangan-pertimbangan ini serupa dengan yang berlaku untuk simpul lain pada bus yang sama dan dibahas dalam bagian mengenai keamanan komunikasi jaringan di dalam kendaraan.

Sensor yang Dipasang di Bagian Luar

Sensor yang dipasang di bagian luar (*externally mounted sensors*) dibedakan berdasarkan fakta bahwa input utamanya berasal dari lingkungan fisik di luar batas kendaraan. Banyak dari sensor jenis ini yang dibahas dalam bab ini berfungsi sebagai input utama bagi fitur ADAS (*Advanced Driver Assistance Systems*), seperti pengereman darurat otomatis, *adaptive cruise control*, dan bantuan penjagaan lajur (*lane keeping assistance*). Subbagian berikut menjelaskan arsitektur dan antarmuka dari jenis-jenis sensor eksternal yang tercakup dalam bab ini.

Sistem Pemantau Tekanan Ban

Sistem pemantau tekanan ban atau *Tire Pressure Monitoring Systems* (TPMS) terdiri dari unit pemancar yang dipasang pada setiap roda dan modul penerima yang terintegrasi ke dalam bodi kendaraan. Setiap pemancar merupakan perangkat berdaya rendah yang ditenagai oleh baterai; perangkat ini mengukur tekanan serta suhu ban dan mengirimkan paket status secara berkala menggunakan sinyal frekuensi radio (*Radio Frequency*/RF) jarak pendek. Sinyal ini biasanya beroperasi pada pita frekuensi 315 MHz di Amerika Utara, atau pita frekuensi *Industrial, Scientific and Medical* (ISM) 433,92 MHz di Eropa. Modul penerima kemudian mendekode transmisi yang masuk dan meneruskan data tekanan serta suhu tersebut ke *gateway* atau modul kontrol bodi melalui bus CAN internal.

Pemancar TPMS diidentifikasi melalui ID sensor unik yang dikenali oleh penerima selama prosedur pendaftaran atau pemasangan (*pairing*), yang biasanya dilakukan saat penggantian ban atau setelah rotasi posisi roda. ID sensor dikirimkan dalam bentuk teks terbuka (*cleartext*) di dalam struktur paket dan, pada sebagian besar sistem yang diterapkan, tidak diautentikasi secara kriptografis. Penerima mengandalkan kecocokan ID sensor dan dalam beberapa implementasi, ambang batas kekuatan sinyal yang diterima sebagai kriteria utama untuk menerima data yang masuk. Arsitektur komunikasi satu arah ini berarti pengirim tidak mengetahui apakah paket-paketnya telah diterima, dan penerima tidak memiliki mekanisme untuk menantang atau memverifikasi identitas pengirim.

Komunikasi dari penerima TPMS ke modul kontrol bodi (*body control module*) atau *gateway* biasanya dilakukan melalui bus di dalam kendaraan, seperti CAN atau LIN. Pada tahap ini, data TPMS telah didekodekan dari transmisi RF dan diperlakukan seperti pesan lainnya di dalam bus. Lebih lanjut, keamanan komunikasi bus internal serupa dengan sensor lain yang terhubung melalui CAN, termasuk dalam hal penggunaan atau ketiadaan autentikasi pesan di tingkat bus.

Sistem Kamera

Sistem persepsi berbasis kamera biasanya melibatkan beberapa sensor gambar yang ditempatkan di sekeliling kendaraan, mencakup bidang pandang ke arah depan, belakang, dan samping. Data gambar mentah dikirimkan dari setiap modul kamera ke unit pemrosesan pusat melalui tautan *Serializer-Deserializer* (SerDes) berbandwidth tinggi. *Gigabit Multimedia Serial Link* (GMSL) dan *Flat Panel Display Link* (FPD-Link) merupakan dua standar eksklusif yang paling banyak digunakan dalam sistem kamera otomotif.

Pemrosesan data kamera bervariasi tergantung pada arsitektur sistemnya. Beberapa desain mengirimkan frame mentah atau frame yang telah diproses oleh *Image Signal Processor* (ISP) ke *High-Performance Computer* (HPC) terpusat untuk keperluan inferensi. Sementara itu, desain lain melakukan deteksi objek dan pengenalan lajur secara lokal di dalam modul kamera atau melalui *Electronic Control Unit* (ECU) ISP khusus, kemudian menyebarkan keluaran terstruktur seperti daftar objek dan penanda lajur melalui jaringan kendaraan. Perbedaan ini penting dari perspektif keamanan, karena celah serangan (*attack surface*) untuk penyuntikan gambar berbahaya (*adversarial image injection*) pada tingkat SerDes berbeda dengan celah serangan untuk penyuntikan keluaran terstruktur yang telah dimanipulasi pada tingkat jaringan internal kendaraan. Saluran kendali untuk sistem kamera, yang membawa pesan sinkronisasi, eksposur, dan konfigurasi, biasanya beroperasi menggunakan protokol *Inter-Integrated Circuit* (I2C) yang berjalan bersamaan dengan atau tertanam di dalam tautan SerDes. Jalur kendali ini beroperasi dengan *bandwidth* yang jauh lebih rendah dibandingkan jalur data video.

Sistem Radar

Sistem radar otomotif beroperasi pada rentang frekuensi gelombang milimeter 24 GHz atau 77 hingga 81 GHz. Saat ini, pita frekuensi yang lebih tinggi lebih banyak digunakan untuk deteksi jarak jauh dan menengah karena memiliki lebar pita (*bandwidth*) yang lebih besar serta ukuran antena yang lebih ringkas. Modul radar memancarkan sinyal *Frequency-Modulated Continuous Wave* (FMCW), kemudian menghitung jarak, kecepatan, dan sudut lateral objek yang terdeteksi berdasarkan pantulan sinyal yang diterima. Hasilnya berupa daftar objek atau peta range-Doppler yang diteruskan ke lapisan fusi sensor (*sensor fusion layer*) pada kendaraan.

Modul radar mengirimkan data keluarannya ke lapisan fusi sensor melalui protokol CAN, FlexRay, atau Automotive Ethernet, tergantung pada kebutuhan laju data aplikasi tersebut. Sistem radar depan jarak jauh yang harus menyediakan daftar objek dengan laju pembaruan tinggi untuk fungsi ADAS seperti *adaptive cruise control* dan pengereman darurat otomatis semakin banyak menggunakan Automotive Ethernet. Sementara itu, radar jarak

pendek yang ditempatkan di sudut kendaraan untuk pemantauan titik buta (*blind-spot monitoring*) dan deteksi lalu lintas melintang (*cross-traffic detection*) dapat menggunakan protokol CAN, mengingat kebutuhan lebar pitanya yang lebih rendah.

Sistem LiDAR

Sistem LiDAR menggunakan cahaya laser berdenyut atau termodulasi biasanya pada panjang gelombang 905 nm untuk sistem *time-of-flight*, atau 1550 nm untuk sistem yang dirancang dengan daya keluaran lebih tinggi namun memiliki risiko lebih rendah terhadap keamanan mata untuk mengukur jarak objek di lingkungan sekitar dengan menghitung selang waktu antara pulsa yang dipancarkan dan pantulan yang kembali. Hasilnya berupa point cloud (kumpulan titik) tiga dimensi dengan laju pembaruan yang umumnya berkisar antara 10 hingga 20 Hz untuk sensor multisaluran yang berputar, atau lebih tinggi untuk sistem solid-state. Pemrosesan *point cloud* LiDAR membutuhkan daya komputasi yang besar, sehingga data tersebut biasanya dikirimkan ke sistem komputasi berkinerja tinggi (*High-Performance Computing* atau HPC) melalui Automotive Ethernet. Penerima fisik pada sistem LiDAR terdiri dari susunan elemen *Single-Photon Avalanche Diode* (SPAD) atau *Avalanche Photodiode* (APD) yang merespons foton yang datang pada panjang gelombang operasional sensor tersebut, tanpa kemampuan bawaan untuk membedakan sumber foton yang berasal dari lingkungan eksternal.

Terdapat dua arsitektur perangkat keras utama yang digunakan. Sensor mekanis berputar menggunakan cermin berputar atau rakitan cermin untuk menyapu sinar laser melintasi bidang pandang horizontal yang luas, menghasilkan pola pemindaian yang deterministik dan dapat diprediksi baik dalam sudut maupun waktu. Sensor *solid-state* menggunakan sinar yang dikendalikan secara elektronik tanpa bagian yang bergerak, yang semakin umum digunakan pada kendaraan produksi massal karena keandalannya yang lebih baik dan profilnya yang lebih rendah. Kedua desain tersebut mengirimkan data *point cloud* mereka ke HPC melalui *Automotive Ethernet*, dan keduanya biasanya menggunakan antarmuka *bandwidth* rendah terpisah, seperti CAN, untuk konfigurasi, pemantauan kesehatan, dan manajemen pembaruan *firmware*.

Sistem Ultrasonik

Sistem sensor ultrasonik beroperasi dalam jarak dekat biasanya hingga lima meter untuk aplikasi parkir dengan menggunakan pulsa suara dalam rentang frekuensi 40 hingga 58 kHz. Sensor memancarkan serangkaian pulsa dan mengukur waktu tempuh gema pantulan untuk menghitung jarak ke rintangan terdekat. Beberapa implementasi menerapkan *echo gating* (penyaringan gema), yang membatasi penerima agar hanya memproses gema yang tiba dalam rentang waktu tertentu sesuai dengan jarak pantulan yang masuk akal secara fisik serta mengabaikan sinyal apa pun yang tiba di luar rentang waktu tersebut. Sebuah kendaraan pada umumnya dilengkapi dengan delapan hingga dua belas sensor ultrasonik yang ditempatkan di sekeliling bumper depan dan belakang. Setiap sensor dihubungkan ke unit kontrol elektronik (ECU) bantuan parkir melalui protokol LIN, CAN, atau sinyal analog *Pulse-Width Modulated* (PWM) sederhana. Karena sensor ultrasonik beroperasi pada jarak yang jauh lebih dekat daripada jangkauan deteksi minimum radar maupun LiDAR, sensor ini menjadi jenis sensor

utama yang digunakan untuk mendeteksi objek di sekitar kendaraan dalam skenario parkir kecepatan rendah dan bantuan manuver otomatis.

Arsitektur Sensor yang Dipasang Secara Eksternal

Dari sudut pandang arsitektur, sensor yang dipasang secara eksternal biasanya mengikuti salah satu dari dua model penyebaran. Model pertama terdiri dari *sensor front-end* yang sebagian besar mengirimkan data mentah ke ECU terpusat, pengontrol domain, atau HPC untuk diproses. Contoh umum adalah modul kamera yang mengirimkan aliran piksel melalui SerDes otomotif ke prosesor sinyal gambar atau *Sistem-on-Chip* (SoC) persepsi. Dalam desain seperti itu, front-end sensor berisi logika pra-pemrosesan minimal, dan sebagian besar komputasi persepsi terjadi di komputasi terpusat.

Model kedua terdiri dari sensor pintar yang melakukan pemrosesan onboard yang substansial dan mempublikasikan output tingkat yang lebih tinggi. Modul radar dan LiDAR sering termasuk dalam kategori ini, menjalankan pipeline pemrosesan sinyal secara internal untuk menghasilkan daftar objek, trek, atau awan titik. Beberapa sistem kamera canggih juga menggabungkan pemrosesan gambar onboard dan mempublikasikan metadata terstruktur selain atau sebagai pengganti frame mentah. Sistem ultrasonik sering menghitung pengukuran waktu tempuh dan jarak dalam modul kontrol lokal sebelum mempublikasikan hasilnya ke hulu.

Dalam praktiknya, platform modern menggabungkan kedua model tersebut. Aliran gambar mentah dapat berdampingan dengan jejak objek radar, awan titik LiDAR, dan pengukuran jarak ultrasonik, semuanya bertemu di pengendali domain atau zona untuk fusi. Konektivitas mencerminkan perbedaan bandwidth dan pemrosesan ini dan biasanya dibagi menjadi bidang data dan bidang kontrol. Bidang data mengangkut data pengukuran dan dapat mencakup:

- Tautan SerDes yang membawa muatan kamera
- Ethernet otomotif yang menghubungkan radar, LiDAR, dan sensor bandwidth tinggi lainnya ke pengendali domain atau zona

Di sisi lain, bidang kontrol mengelola konfigurasi, sinkronisasi, kalibrasi, dan pelaporan kesehatan. Bidang ini umumnya bergantung pada CAN, LIN, I2C, atau saluran samping khusus vendor, bahkan ketika jalur data utama menggunakan Ethernet atau SerDes. Dari perspektif perangkat keras dan perangkat lunak, banyak sensor yang dipasang secara eksternal menyerupai ECU kompak. Sensor ini sering mengintegrasikan:

- Unit mikrokontroler (MCU) atau SoC
- Memori dan penyimpanan khusus
- Serializer berkecepatan tinggi atau pengontrol jaringan
- *Firmware* yang bertanggung jawab atas kontrol sensor, kalibrasi, penyelarasan waktu, pra-pemrosesan sinyal, dan publikasi data

Sistem AI dan ML

Kategori arsitektur keempat adalah sistem AI dan ML, yang mencakup sistem perangkat lunak yang mengonsumsi data sensor atau input lainnya, lalu melakukan tugas inferensi, perencanaan, atau interaksi menggunakan model pembelajaran mesin (*machine learning*).

Terdapat dua subsistem berbeda dalam kategori ini: tumpukan (*stack*) persepsi dan perencanaan ADAS, serta asisten AI di dalam kendaraan.

Persepsi dan Perencanaan ADAS

Aplikasi AI yang paling krusial bagi keselamatan ditemukan dalam tumpukan persepsi dan perencanaan pada platform ADAS serta sistem kemudi otomatis. Tumpukan persepsi ADAS menerima input sensor dari kamera, radar, LiDAR, dan teknologi penginderaan lainnya, kemudian menghasilkan model lingkungan kendaraan yang menjadi masukan bagi lapisan perencanaan dan kontrol. Tugas persepsi yang dilakukan oleh tumpukan ini meliputi deteksi dan klasifikasi objek, pengenalan pejalan kaki dan pesepeda, deteksi lajur, pengenalan rambu dan sinyal lalu lintas, serta estimasi ruang bebas (*free-space*). Tugas-tugas ini dilaksanakan oleh model *Deep Neural Network* (DNN) biasanya berupa *Convolutional Neural Network* (CNN), yaitu jenis arsitektur yang dirancang untuk mengekstrak fitur spasial dari data gambar atau arsitektur berbasis transformer yang dilatih menggunakan kumpulan data besar yang telah diberi anotasi. Output dari tumpukan persepsi memengaruhi keputusan kontrol kendaraan secara langsung, sehingga integritas model-model ini menjadi aspek yang sangat penting bagi keselamatan. Perlu dicatat bahwa model jaringan saraf (*neural network*) bukanlah sekadar perangkat lunak dalam pengertian konvensional. Model ini merupakan aproksimator fungsi yang perilakunya ditentukan oleh karakteristik statistik dari data pelatihannya, dan output-nya dapat dimanipulasi melalui input yang dirancang secara khusus di luar distribusi data yang digunakan saat pelatihan.

Selain itu, tumpukan ADAS berinteraksi dengan perangkat keras sensor kendaraan melalui alur data (*data pipeline*) yang telah ditetapkan: data sensor mentah atau yang telah dipra-pemrosesan diterima oleh HPC (*High-Performance Computer*) melalui Ethernet, kemudian diproses oleh model persepsi. Output terstruktur yang dihasilkan selanjutnya diteruskan ke sistem kontrol dan perencanaan melalui *Application Programming Interface* (API) atau lapisan middleware yang terdefinisi dengan jelas. Dalam banyak implementasi, alur inferensi diisolasi di dalam lingkungan eksekusi khusus pada HPC, dengan akses ke fungsi kontrol kendaraan yang diatur oleh manajer keselamatan (*safety manager*) yang bertugas memastikan batasan domain desain operasional (*operational design domain*) tetap terjaga.

Asisten AI Dalam Kendaraan

Kategori kedua dari penerapan AI adalah asisten AI dalam kendaraan, yaitu antarmuka bahasa alami yang memungkinkan pengemudi dan penumpang untuk mengendalikan fungsi-fungsi kendaraan melalui perintah suara. Fitur ini mencakup pengendalian navigasi, pengelolaan media dan suhu kabin, penanganan panggilan telepon, serta permintaan informasi umum. Sistem ini biasanya dibangun di atas infrastruktur model bahasa besar (LLM), yang dapat di-*hosting* secara lokal di dalam kendaraan atau diakses dari jarak jauh melalui API cloud. Asisten ini menerima input suara melalui sistem mikrofon kendaraan, memprosesnya melalui lapisan pengenalan ucapan dan kemudian melalui model bahasa, lalu menghasilkan respons atau menjalankan fungsi kendaraan melalui API yang telah ditentukan. Selain input suara, asisten ini juga dapat menyerap konten dari sumber data eksternal seperti deskripsi titik tujuan (*point of interest*) dalam navigasi, pesan masuk, dan respons API dari layanan yang

terhubung yang semuanya menjadi bagian dari konteks yang diproses oleh model bahasa saat menghasilkan respons atau menjalankan suatu fungsi.

Batas antara asisten AI dan bagian lain dari kendaraan ditentukan oleh model izin asisten tersebut, yang menetapkan fungsi apa saja yang boleh dijalankan oleh model AI dan dalam kondisi seperti apa. Batas ini merupakan titik krusial dalam penegakan kepercayaan (*trust enforcement*). Jika izin yang diberikan kepada asisten terlalu luas, atau jika model izin tersebut dapat dilewati melalui input yang dirancang secara khusus, maka penyerang yang mampu memengaruhi input asisten dapat menjalankan fungsi kendaraan secara tidak langsung tanpa persetujuan pengemudi maupun penumpang.

Meskipun ketergantungan yang kian besar pada sensor kendaraan dan sistem berbasis AI telah meningkatkan kemampuan persepsi, otomatisasi, dan bantuan pengemudi secara signifikan, teknologi-teknologi ini juga menghadirkan tantangan baru terkait keamanan dan keandalan. Bagian berikut akan mengulas kelemahan umum serta permukaan serangan (*attack surface*) yang berkaitan dengan sistem penginderaan dan AI pada kendaraan modern.

9.3 ANCAMAN DAN KELEMAHAN KEAMANAN PADA SENSOR KENDARAAN DAN SISTEM AI

Kelemahan yang dijelaskan di bagian ini berakar pada kombinasi antara sifat fisik mendasar dari teknologi penginderaan, keputusan desain di tingkat protokol yang dibuat selama proses standardisasi atau komersialisasi, serta pilihan arsitektur yang diambil saat integrasi di tingkat kendaraan. Karena sensor-sensor ini memperoleh input utama dari lingkungan fisik di luar batasan kendaraan, permukaan serangan yang ditimbulkannya secara kualitatif berbeda dibandingkan dengan komponen yang terhubung dalam jaringan internal: penyerang tidak perlu memiliki akses ke sistem elektronik kendaraan untuk memengaruhi apa yang dipersepsikan oleh sensor tersebut. Dengan memanipulasi sinyal elektromagnetik, optik, atau akustik di lingkungan operasional sensor, penyerang dapat memengaruhi data yang dikirimkan sensor ke bagian lain dari kendaraan, tanpa perlu membangun koneksi apa pun ke jaringan internal kendaraan tersebut. Dalam banyak kasus, kelemahan ini bukan muncul akibat kesalahan implementasi, melainkan karena adanya kompromi desain yang sempat dianggap dapat diterima pada masa ketika lanskap ancaman belum dipahami sepenuhnya. Memahami akar penyebab dari setiap kelemahan sangatlah penting bagi seorang penguji penetrasi (*penetration tester*), karena hal ini menentukan apakah kelemahan tersebut dapat dieksploitasi dalam kondisi nyata dan apa dampak yang mungkin ditimbulkannya terutama mengingat bahwa keberhasilan dalam memanipulasi persepsi sensor dapat memengaruhi perilaku kendaraan yang krusial bagi keselamatan.

Kelemahan Injeksi Sinyal Akustik Pada Sensor Inersia MEMS

Sensor inersia MEMS rentan terhadap serangan pada lapisan fisik yang dikenal sebagai injeksi akustik resonan. Sensor ini mengubah gerakan fisik menjadi sinyal listrik melalui struktur mekanis berskala mikro. Bagian depan analog (*analog front end*) dari sensor MEMS kapasitif dapat dipengaruhi oleh nada akustik yang disetel secara presisi dan diarahkan ke sensor dari sumber eksternal. Dalam kondisi tertentu, interaksi antara frekuensi resonansi

mekanis dan jalur konversi analog-ke-digital pada sensor menyebabkan energi akustik yang diinjeksikan menghasilkan pembacaan digital yang terstruktur bukan sekadar derau (*noise*). Hal ini memungkinkan penyerang untuk memicu penyimpangan terkendali pada keluaran sensor tanpa perlu mengakses jaringan kendaraan atau perangkat lunak (*firmware*) pada unit kontrol elektronik (ECU). Keberhasilan serangan ini bergantung pada penempatan sensor, desain selubung (*enclosure*), pengondisian sinyal, serta apakah ECU yang menerima data tersebut menerapkan pemeriksaan plausibilitas (kewajaran) terhadap keluaran sensor. Tanpa pemeriksaan semacam itu, penyimpangan yang disebabkan oleh sinyal akustik dapat langsung memengaruhi fungsi-fungsi yang berkaitan dengan keselamatan.

Transmisi RF Tanpa Autentikasi dan Pelacakan Kendaraan Pada TPMS

Kelemahan paling signifikan dari sistem TPMS yang digunakan saat ini adalah tidak adanya autentikasi pesan. Pemancar TPMS berkomunikasi menggunakan transmisi radio satu arah tanpa adanya rahasia bersama (*shared secret*) antara pemancar dan penerima, serta tanpa ikatan kriptografis antara ID sensor dan muatan data (*data payload*). Struktur paket yang biasanya mencakup ID sensor, nilai tekanan, pembacaan suhu, status baterai, dan *Cyclic Redundancy Check* (CRC) dapat dengan mudah didekode melalui penangkapan sinyal radio secara pasif. Hal ini dimungkinkan karena skema modulasi yang digunakan telah didokumentasikan secara luas dalam literatur terbuka maupun implementasi dekoder yang tersedia untuk publik. CRC sendiri berfungsi untuk mendeteksi kesalahan transmisi, bukan untuk mengautentikasi sumber data. Akibatnya, penerima tidak memiliki mekanisme untuk membedakan transmisi yang sah dari transmisi yang dibuat oleh perangkat milik penyerang yang mampu memancarkan sinyal pada frekuensi yang tepat dengan format muatan data yang benar.

Persyaratan regulasi yang mengharuskan TPMS untuk memberi peringatan kepada pengemudi dalam jangka waktu tertentu setelah terjadi penurunan tekanan ban yang nyata, membuat sistem ini rentan menerima data masuk dengan penyaringan yang minimal. Artinya, jika penerima menerapkan pemeriksaan plausibilitas yang rumit, terdapat risiko melampaui ambang batas waktu peringatan yang ditetapkan oleh regulasi. Kondisi ini memperkuat kecenderungan sistem untuk menerima paket dengan ID sensor yang cocok tanpa melakukan validasi lebih lanjut. Terlebih lagi, karya ilmiah yang telah dipublikasikan menunjukkan bahwa sejak tahun 2010, pemantauan pasif terhadap sinyal TPMS dari posisi diam di tepi jalan sudah cukup untuk mendapatkan ID sensor dan melacak kendaraan tertentu dalam jarak beberapa meter dengan menggunakan perangkat keras radio yang murah. Selain itu, pengiriman ulang (*replay*) paket data yang telah ditangkap ke kendaraan target dapat memicu peringatan tekanan ban palsu pada sistem yang digunakan saat ini. Kelemahan spesifik yang muncul akibat tidak adanya autentikasi sinyal RF pada TPMS antara lain sebagai berikut:

- **Transmisi pengidentifikasi sensor dalam bentuk teks terbuka (*cleartext*) yang memungkinkan serangan replay pasif**: Setiap paket yang dikirimkan memuat ID sensor dalam kolom yang tidak dienkripsi dan tidak diautentikasi. Penyerang yang mampu melakukan demodulasi sinyal dapat memperoleh ID sensor tersebut dan menyisipkannya ke dalam paket yang telah dimanipulasi. Hal ini membuat serangan

replay dan penyisipan paket (*packet injection*) dapat dilakukan dengan sangat mudah menggunakan perangkat keras *software-defined radio* (SDR) yang tersedia secara komersial.

- **Penerimaan pembacaan palsu akibat penyaringan (*filtering*) penerima yang terlalu longgar**: Penerima yang dikonfigurasi untuk menerima paket apa pun dengan ID sensor yang cocok tanpa melakukan pemeriksaan plausibilitas terhadap nilai tekanan akan menerima paket palsu yang melaporkan tekanan nol, nilai di luar rentang yang seharusnya, atau hilangnya tekanan secara bersamaan pada keempat roda. Kondisi ini memicu peringatan palsu yang dapat mengalihkan perhatian pengemudi atau memicu respons otomatis dari sistem kendaraan.
- **Penyalahgunaan mekanisme pendaftaran (*enrollment*) dan penambahan sensor ilegal**: Prosedur pendaftaran yang biasanya dipicu saat penggantian ban atau rotasi posisi roda mengharuskan penerima untuk menerima ID sensor baru. Penyerang yang mampu mengirimkan paket pendaftaran dengan format yang tepat dapat mendaftarkan pemancar ilegal, sehingga paket-paket selanjutnya dari pemancar tersebut akan dianggap sebagai data sensor yang sah. Selain itu, pada implementasi di mana pendaftaran dilakukan melalui antarmuka *Bluetooth Low Energy* (BLE), analisis terhadap sistem produksi tertentu menunjukkan adanya kerentanan keamanan memori dalam kode yang dijalankan selama prosedur pendaftaran, yang berpotensi untuk dieksploitasi.
- **ID sensor yang bersifat tetap memungkinkan pelacakan kendaraan secara terus-menerus**: Karena setiap sensor mengirimkan pengidentifikasi unik yang tetap secara berkala sepanjang masa pakai kendaraan, pengamat pasif yang berada di tepi jalan atau di area parkir dapat menghubungkan transmisi dari ID sensor yang sama dari waktu ke waktu. Dengan demikian, pengamat tersebut dapat menyusun riwayat pergerakan kendaraan terkait tanpa perlu melakukan interaksi aktif apa pun dengan kendaraan tersebut.

Kelemahan Sistem Kamera Pada Lapisan Fisik, Antarmuka, dan Inferensi

Sistem persepsi berbasis kamera rentan terhadap kelemahan yang beroperasi pada tiga tingkatan berbeda: lapisan optik fisik, lapisan konektivitas antarmuka, dan lapisan inferensi pembelajaran mesin. Pada lapisan fisik, sensor gambar pada dasarnya bersifat pasif dan menerima foton dari sumber mana pun yang berada dalam bidang pandangnya. Pancaran cahaya inframerah-dekat (*near-infrared*) berintensitas tinggi yang diarahkan ke bukaan kamera dapat menjenuhkan fotodioda pada sensor gambar, sehingga menghasilkan area piksel jenuh dalam bingkai keluaran yang menghilangkan informasi visual penting di area tersebut. Ambang batas kejenuhan bergantung pada rentang dinamis (*dynamic range*) sensor dan efektivitas sistem eksposur otomatis; namun, sumber cahaya terarah yang cukup terang dapat melampaui mekanisme kompensasi tersebut. Karena cahaya inframerah-dekat berada di luar spektrum yang dapat dilihat, serangan semacam ini tidak akan disadari oleh penumpang kendaraan yang mengamati tampilan kamera, dan tidak pula menimbulkan indikasi apa pun yang dapat dilihat oleh pengguna jalan lainnya.

Pada lapisan antarmuka, tautan SerDes membawa aliran video mentah dari kamera ke unit pemrosesan dalam bentuk sinyal elektrik berkelanjutan. Dalam sebagian besar penerapan saat ini, aliran data ini tidak diautentikasi secara kriptografis dari ujung ke ujung (*end-to-end*). Dengan demikian, penyerang yang memiliki akses fisik ke rangkaian kabel dapat menyisipkan aliran video buatan pada tingkat tautan yakni melalui kabel SerDes di antara kamera dan ECU sehingga menyajikan citra palsu kepada ECU pemroses, sementara modul kamera fisik tetap berfungsi seperti biasa.

Bahkan jika autentikasi diterapkan pada jalur video, kelemahan tetap dapat muncul dalam hal penanganan sertifikat, perlindungan terhadap pengulangan bingkai (*frame replay*), dan cakupan autentikasi yang tidak menyeluruh. Sebagai contoh, sistem yang hanya mendukung autentikasi pada wilayah tertentu (*region of interest*) alih-alih seluruh bingkai menghadirkan risiko tambahan; jika wilayah yang diautentikasi dalam suatu bingkai dipilih tanpa menggunakan sumber keacakan yang aman secara kriptografis, penyerang yang dapat memprediksi wilayah mana yang akan diautentikasi dapat menyisipkan bingkai palsu yang tetap mempertahankan bagian yang diautentikasi, namun mengganti bagian lainnya dengan konten buatan. Selain itu, saluran kontrol I2C yang mengatur pengaturan pencahayaan, sinkronisasi, dan konfigurasi kamera umumnya lebih mudah diakses dengan perangkat keras sederhana dibandingkan jalur data video, serta sering kali memiliki kontrol autentikasi yang lebih minim.

Kelemahan lain pada lapisan antarmuka berkaitan dengan autentikasi identitas sensor. Pada kebanyakan sistem yang diterapkan saat ini, ECU pemroses tidak memiliki mekanisme untuk memverifikasi identitas fisik modul kamera yang terhubung ke port SerDes tertentu. Akibatnya, modul kamera dapat diganti dengan perangkat tiruan yang menyajikan antarmuka elektrik yang valid, namun menghasilkan citra yang telah dimodifikasi atau dibuat secara selektif, tanpa ECU pemroses mampu membedakan perangkat pengganti tersebut dari sensor aslinya.

Pada lapisan inferensi, detektor objek berbasis DNN (*Deep Neural Network*) rentan terhadap input adversarial. Kerentanan ini muncul dari struktur batasan keputusan (*decision boundaries*) yang dipelajari oleh jaringan tersebut, yang dapat dieksploitasi dengan menambahkan gangguan kecil yang dihitung secara cermat ke dalam input yang sah. Dalam konteks otomotif, penelitian sebelumnya menunjukkan bahwa *adversarial patch* seperti stiker yang ditempelkan pada rambu lalu lintas dalam bidang pandang kamera dapat menyebabkan detektor objek kelas industri gagal mendeteksi pejalan kaki, salah mengklasifikasikan rambu berhenti, atau mendeteksi objek yang sebenarnya tidak ada (*ghost detections*). Kelemahan spesifik yang berkaitan dengan ketiga lapisan ini meliputi hal-hal berikut:

- **Saturasi inframerah dekat (*near-infrared*) pada fotodioda sensor gambar**: Pencahayaan inframerah dekat yang diarahkan dari sumber di luar spektrum tampak dapat menyebabkan saturasi pada piksel di area sensor yang terdampak. Area yang mengalami saturasi ini tidak dapat dibedakan dari segmen bingkai (*frame*) yang mengalami paparan berlebih (*overexposed*), sehingga dapat menyebabkan model

persepsi gagal mendeteksi objek atau menghasilkan deteksi yang tidak akurat di bidang pandang yang terdampak.

- **Kegagalan deteksi objek akibat adversarial *patch* (tambalan adversarial)**: *Adversarial patch* berupa pola cetak khusus yang ditempatkan pada atau di dekat objek dalam bidang pandang kamera dapat menyebabkan model deteksi objek gagal mengenali objek tersebut, salah mengklasifikasikannya sebagai kategori yang tidak berbahaya, atau memicu deteksi palsu di area sekitarnya.
- **Tidak adanya autentikasi end-to-end pada aliran video SerDes**: Penyerang yang memiliki akses ke tautan SerDes antara modul kamera dan unit kontrol elektronik (ECU) pemroses dapat menyuntikkan aliran video buatan. ECU pemroses akan menerima aliran tersebut sebagai keluaran kamera yang sah karena tidak adanya autentikasi kriptografis *end-to-end*. Jalur serangan ini memerlukan akses fisik ke kendaraan, namun tidak mengharuskan eksploitasi kerentanan perangkat lunak apa pun dalam tumpukan pemrosesan (*processing stack*).
- **Manipulasi konfigurasi saluran kontrol I2C**: Bus I2C yang dapat diakses di titik konektor kamera (atau di dekatnya) dapat diperiksa dan dimodifikasi menggunakan perangkat keras yang mudah diperoleh, seperti adaptor USB-ke-I2C atau papan mikrokontroler murah yang mampu berfungsi sebagai bus master. Penulisan data tanpa autentikasi ke dalam register konfigurasi kamera dapat mengubah parameter seperti eksposur, gain, pengaturan waktu bingkai (*frame timing*), dan sinkronisasi. Hal ini dapat menurunkan kualitas gambar atau mengganggu operasional modul kamera tanpa menunjukkan tanda-tanda apa pun secara eksternal.

Kelemahan Terkait Penerimaan Sinyal RF Tanpa Autentikasi dan Prediktabilitas Bentuk Gelombang Pada Sistem Radar

Kelemahan mendasar dari sistem penginderaan berbasis radar adalah ketidakmampuan penerima untuk membedakan antara pantulan sinyal yang sah dengan transmisi buatan yang meniru karakteristik sinyal dari pantulan yang sah. Radar otomotif mengukur jarak dengan mengorelasikan perbedaan frekuensi antara sinyal chirp yang dipancarkan dan sinyal yang diterima yang dikenal sebagai frekuensi beat untuk menghitung jarak ke objek yang terdeteksi. Penyerang yang memancarkan sinyal yang menghasilkan frekuensi beat yang tepat saat dicampur dengan bentuk gelombang pancaran radar itu sendiri, akan menyebabkan sistem pemrosesan radar mendeteksi adanya objek pada jarak yang sesuai dengan sinyal tersebut. Karena radar beroperasi berdasarkan sifat-sifat matematis dari sinyal yang diterima bukan berdasarkan bentuk autentikasi atau verifikasi identitas apa pun maka pantulan palsu yang disisipkan tidak dapat dibedakan dari pantulan asli pada tahap pemrosesan sinyal.

Kelayakan serangan jenis ini bergantung pada kemampuan penyerang untuk memulihkan parameter bentuk gelombang (*waveform*) radar. Radar otomotif generasi awal yang menggunakan bentuk gelombang chirp tetap yang sederhana rentan terhadap penyerang yang mampu memperkirakan kemiringan (*slope*) dan waktu chirp melalui pengamatan pasif dalam durasi singkat. Radar modern yang menggunakan urutan chirp acak atau fleksibilitas

bentuk gelombang (*waveform agility*) membuat penyuntikan sinyal secara tersinkronisasi menjadi lebih sulit; dengan kata lain, hal ini meningkatkan hambatan bagi penyerang, namun tidak sepenuhnya menghilangkan ancaman tersebut. Kelemahan spesifik yang terkait dengan arsitektur penerimaan RF terbuka pada radar otomotif meliputi:

- **Penyuntikan chirp tersinkronisasi yang menciptakan objek semu (*phantom objects*)**: Penyerang yang berhasil memulihkan parameter chirp radar melalui pengamatan pasif singkat dapat memancarkan sinyal yang kemudian diinterpretasikan oleh sistem pemrosesan range-Doppler radar sebagai pantulan dari suatu objek pada jarak dan kecepatan yang ditentukan oleh penyerang. Objek semu yang muncul dalam daftar objek terdeteksi ini kemudian diteruskan ke lapisan fusi sensor (*sensor fusion layer*) sebagai input sensor yang tampak valid, kecuali jika terdapat pemeriksaan konsistensi lintas-modal (*cross-modal consistency checks*) untuk mendeteksi ketidaksesuaian dengan input dari teknologi penginderaan lain seperti kamera atau LiDAR.
- **Gangguan derau pita lebar (*broadband noise jamming*) yang menghambat deteksi objek nyata**: Interferensi pita lebar yang dipancarkan secara terus-menerus pada frekuensi operasional radar akan meningkatkan ambang batas derau (*noise floor*) yang terbaca oleh algoritma CFAR. Akibatnya, ambang batas deteksi akan naik ke tingkat yang justru menekan pantulan sah dari objek nyata. Dalam skenario ini, penyerang memiliki keunggulan daya yang signifikan karena sinyal penyerang hanya mengalami rugi-rugi propagasi satu arah (*single-path propagation loss*), sementara pantulan asli mengalami rugi-rugi propagasi dua arah (*round-trip propagation loss*) yang sebanding dengan pangkat empat dari jarak target. Artinya, penyerang hanya memerlukan daya pancar yang jauh lebih kecil dibandingkan radar itu sendiri untuk mengacaukan proses deteksi.

Kelemahan Lapisan Fisik dan Antarmuka LiDAR

Sistem LiDAR yang beroperasi berdasarkan prinsip *time-of-flight* (waktu tempuh) memiliki kelemahan pada lapisan fisik yang serupa dengan radar: penerima tidak memiliki cara untuk membedakan antara foton pantulan yang berasal dari pemancarnya sendiri dengan foton yang berasal dari sumber eksternal. LiDAR berdenyut (*pulsed LiDAR*) memancarkan pulsa laser dan mendeteksi foton yang kembali dalam rentang waktu tertentu yang sesuai dengan pantulan dari jarak tertentu. Seorang penyerang yang menyuntikkan foton pada panjang gelombang operasional sensor dengan penundaan yang diatur sedemikian rupa terhadap pulsa yang dipancarkan dapat menyebabkan sensor mencatat pengukuran jarak yang keliru, yang sesuai dengan durasi penundaan tersebut.

Karena jarak ditentukan semata-mata berdasarkan waktu kedatangan foton tanpa adanya atribut yang dapat mengidentifikasi asal foton tersebut serangan ini beroperasi sepenuhnya pada lapisan fisik dan tidak memerlukan upaya untuk melewati kontrol kriptografi maupun protokol apa pun. Teknik anti-pemalsuan (*anti-spoofing*) yang berbasis pada identifikasi karakteristik unik pulsa (*pulse fingerprinting*) dapat diakali sebagian dengan cara memancarkan pulsa dalam jumlah yang sangat besar dan tersebar di sepanjang siklus pemindaian sensor; hal ini melumpuhkan mekanisme identifikasi tersebut melalui volume

pulsa, bukan melalui presisi. Selain itu, paparan cahaya inframerah-dekat berdaya tinggi yang diarahkan ke bukaan sensor dapat menjenuhkan susunan detektor SPAD atau APD, sehingga menciptakan area buta (*blind sector*) permanen yang menghalangi semua sinyal pantulan dari arah tersebut.

Antarmuka kontrol berbasis bus CAN yang digunakan untuk konfigurasi dan pemantauan kondisi LiDAR menghadirkan kelemahan lain: pesan kontrol tanpa autentikasi dapat memaksa perubahan mode, memicu urutan pembaruan *firmware*, atau mengubah parameter operasional yang memengaruhi perilaku deteksi sensor. Demikian pula, antarmuka Automotive Ethernet yang digunakan untuk mengirimkan data point cloud ke HPC rentan terhadap ketiadaan autentikasi pesan, serangan replay, serta kurangnya verifikasi integritas pada lapisan tautan (*link-layer*) jika kontrol-kontrol tersebut belum diimplementasikan. Kedua kelemahan pada tingkat antarmuka ini serupa dengan kelemahan yang dibahas dalam bagian 6. Kelemahan spesifik yang terkait dengan LiDAR adalah sebagai berikut:

- **Injeksi foton berwaktu (*timed photon injection*) yang menciptakan pengukuran jarak palsu**: Foton yang dikirimkan pada panjang gelombang operasional sensor dengan penundaan terkontrol setelah pulsa keluar, akan didaftarkan oleh susunan SPAD atau APD sebagai pantulan jarak yang valid. Penyerang yang mampu mengkarakterisasi waktu pulsa sensor dapat menghasilkan objek point cloud palsu pada posisi arbitrer di dalam bidang pandang sensor.
- **Pintasan statistik terhadap kontrol anti-pemalsuan (*anti-spoofing*) berbasis pulse fingerprinting**: Dalam kasus di mana sensor mencoba memverifikasi apakah pulsa yang kembali cocok dengan tanda emisi miliknya sendiri, pemancaran pulsa dengan laju pengulangan tinggi yang didistribusikan secara seragam di seluruh siklus pemindaian akan meningkatkan probabilitas bahwa setidaknya satu pulsa yang disuntikkan memenuhi jendela waktu yang diharapkan secara kebetulan (statistik). Hal ini sebagian menggagalkan pemeriksaan fingerprinting tanpa memerlukan pengetahuan presisi mengenai parameter sinkronisasi sensor.
- **Iluminasi inframerah-dekat (*near-infrared*) berintensitas tinggi yang menciptakan sektor buta (*blind sector*) persisten**: Iluminasi inframerah-dekat berdaya tinggi yang diarahkan secara terus-menerus ke apertur sensor akan menjenuhkan susunan penerima pada segmen sudut yang terdampak, sehingga menciptakan sektor buta dalam keluaran point cloud selama iluminasi tersebut berlangsung. Akibatnya, objek yang berada di dalam sektor buta tersebut tidak terdeteksi oleh sistem persepsi kendaraan selama periode iluminasi berlangsung.
- **Saluran kontrol CAN tanpa autentikasi yang memungkinkan manipulasi mode**: Sensor LiDAR yang menerima perintah konfigurasi dan mode operasional melalui antarmuka CAN tanpa autentikasi rentan terhadap pesan yang dirancang khusus. Pesan ini dapat mengubah ambang batas sensitivitas deteksi, memaksa penggunaan mode daya rendah yang mengurangi jangkauan efektif, atau memicu urutan pembaruan *firmware* yang tergantung pada implementasi pembaruannya dapat digunakan untuk memuat *firmware* yang telah dimodifikasi.

Kelemahan Sensor Jarak Ultrasonik Akibat Kurangnya Validasi Sumber Gema

Sensor ultrasonik rentan terhadap gangguan akustik dari sumber mana pun yang mampu menghasilkan suara dalam rentang operasional 40 hingga 58 kHz. Kelemahan ini disebabkan oleh tidak adanya validasi sumber dalam alur pemrosesan gema. Dengan kata lain, sensor mengukur waktu tempuh (*time of flight*) dari gema pertama yang kembali dan melampaui ambang batas deteksi, tanpa mekanisme apa pun untuk menentukan apakah gema tersebut berasal dari pulsa yang dipancarkan oleh sensor itu sendiri atau dari sinyal akustik yang disuntikkan dari luar. Penyerang yang berada dalam jarak fisik yang dekat dapat menyuntikkan gema palsu, yang menyebabkan sistem bantuan parkir melaporkan adanya hambatan yang sebenarnya tidak ada, atau justru menghalangi deteksi gema yang sah dengan memancarkan gangguan terus-menerus yang meningkatkan tingkat kebisingan (*noise floor*) hingga melampaui ambang batas deteksi.

Karena transduser ultrasonik yang murah yang beroperasi pada rentang 40 hingga 58 kHz tersedia secara komersial sebagai komponen dalam perangkat elektronik konsumen dan aplikasi robotika, hambatan perangkat keras untuk melakukan serangan semacam ini tergolong rendah. Jangkauan efektif serangan yang terbatas yang bergantung pada jarak operasional sensor itu sendiri mengharuskan penyerang untuk berada dalam jarak fisik yang dekat dengan kendaraan. Dalam skenario parkir otomatis, di mana kendaraan bergerak dengan kecepatan rendah di ruang terbatas seperti gedung parkir atau jalan masuk, penyerang dapat dengan mudah menempatkan diri dalam jangkauan injeksi yang efektif dan merusak data sensor jarak (*proximity sensor*) untuk memengaruhi keputusan kendali kendaraan. Kelemahan spesifik akibat tidak adanya validasi sumber gema pada sensor ultrasonik meliputi hal-hal berikut:

- **Injeksi gema palsu yang menghasilkan laporan adanya rintangan semu**: Pulsa akustik yang disuntikkan dari luar pada frekuensi operasional sensor yang diatur waktunya agar tiba di sensor dalam jendela deteksi diproses sebagai pantulan sah dari suatu rintangan pada jarak tertentu. ECU (*Electronic Control Unit*) bantuan parkir kemudian menghasilkan peringatan jarak atau memicu respons berhenti otomatis berdasarkan pembacaan sensor yang telah dipalsukan tersebut.
- **Gangguan akustik berkelanjutan yang menghambat deteksi pantulan yang sah**: Transmisi akustik terus-menerus pada frekuensi operasional sensor akan meningkatkan ambang batas kebisingan (*noise floor*) yang digunakan untuk deteksi gema, sehingga sensor gagal mendeteksi rintangan yang sebenarnya. Sistem parkir atau manuver otomatis yang tidak menerima sinyal jarak dapat menginterpretasikan ketiadaan pantulan sebagai ruang kosong, sehingga terus bergerak menuju rintangan nyata yang berpotensi menyebabkan tabrakan.

Autentikasi Asimetris Antara Bidang Data Sensor (*Sensor Data Plane*) dan Bidang Kendali (*Control Plane*)

Sensor yang dipasang di luar kendaraan dengan konektivitas jaringan biasanya memisahkan keluaran pengukuran ber-bandwidth tinggi dari saluran kendali dan konfigurasi yang memiliki bandwidth lebih rendah. Modul kamera, misalnya, dapat mengirimkan aliran

video melalui tautan SerDes, sementara perintah untuk pengaturan pencahayaan (*exposure*) dan sinkronisasi diterima melalui protokol I2C. Unit LiDAR dapat mengirimkan data point cloud melalui Automotive Ethernet, sementara perintah mode operasional dan pemicu pembaruan *firmware* diterima melalui protokol CAN. Pemisahan ini merupakan fitur desain fungsional, namun menimbulkan konsekuensi keamanan jika bidang kendali (*control plane*) dilindungi dengan tingkat pengamanan yang lebih rendah dibandingkan bidang data (*data plane*). Hal ini terjadi karena bandwidth-nya lebih kecil dan tingkat kepentingan fungsionalnya dianggap lebih rendah. Persepsi tersebut dapat menyebabkan autentikasi menjadi opsional atau bahkan dihilangkan sama sekali pada antarmuka bidang kendali.

Dalam praktiknya, *control plane* (bidang kendali) merupakan target serangan yang efektif justru karena ia dapat memengaruhi perilaku sensor secara berkelanjutan. Pesan kendali yang dirancang khusus yang dapat menyesuaikan parameter kalibrasi, memicu peristiwa sinkronisasi, atau mengaktifkan mode pembaruan *firmware* dapat memengaruhi keluaran sensor, bahkan jika *data plane* (bidang data) itu sendiri telah dilindungi oleh autentikasi ujung-ke-ujung (*end-to-end*) dan verifikasi integritas. Oleh karena itu, kedua bidang tersebut harus disertakan secara eksplisit dalam analisis permukaan serangan (*attack surface analysis*) dan harus memenuhi standar autentikasi yang setara, sejauh hal tersebut dimungkinkan secara teknis.

Kesalahan Konfigurasi Batasan Kepercayaan (*Trust Boundary*) Dalam Arsitektur Berbasis Domain dan Zona

Dalam arsitektur kendaraan berbasis domain dan zona, terdapat kelemahan mendasar yang muncul dari cara data sensor yang dipasang di luar dikumpulkan di pengontrol zona (*zone controller*) atau pengontrol domain (*domain controller*) sebelum diteruskan ke pusat komputasi berkinerja tinggi (HPC). Hal ini berlaku terlepas dari jenis sensor apa pun yang digunakan. Proses pengumpulan data ini mengubah batasan kepercayaan yang diterapkan pada data sensor dan memengaruhi bagaimana dampak kompromi pada satu sensor dapat menyebar. Satu sensor yang telah disusupi dapat menyuntikkan data berbahaya ke dalam aliran data yang terkumpul, bahkan sebelum data tersebut diautentikasi atau divalidasi oleh pengontrol domain penerima. Selain itu, pengontrol zona yang telah disusupi dapat memengaruhi data yang dikirimkan oleh berbagai sensor secara bersamaan.

Kesalahan desain yang umum dalam arsitektur ini adalah menganggap data sensor sebagai data tepercaya segera setelah diterima oleh pengontrol zona, dengan hanya mengandalkan batasan zona sebagai satu-satunya titik penegakan kepercayaan. Jika pengontrol zona tidak memvalidasi pesan dari setiap sensor sebelum mengumpulkannya, atau jika jalur komunikasi antara sensor dan pengontrol zona tidak menerapkan autentikasi, maka batasan kepercayaan yang sebenarnya justru berada pada antarmuka RF atau optik sensor tersebut, bukan pada pengontrol zona. Akibatnya, penyerang yang menargetkan jalur sinyal fisik dapat menyuntikkan data yang kemudian diteruskan melalui pengontrol zona ke HPC, dengan tingkat kepercayaan yang sama seperti data sensor yang sah.

Kelemahan Sistem Persepsi Berbasis DNN Pada Lapisan Pelatihan dan Integritas Model

Kelemahan sistem persepsi berbasis DNN (*Deep Neural Network*) muncul dari karakteristik mendasar proses pembelajaran itu sendiri, bukan sekadar dari kesalahan implementasi perangkat lunak pada umumnya. Jaringan saraf tiruan (*neural network*) berfungsi sebagai aproksimator fungsi yang mempelajari pemetaan dari input ke output berdasarkan struktur statistik data pelatihannya. Proses ini tidak menghasilkan model yang dapat digeneralisasi secara andal untuk semua kemungkinan input, melainkan hanya untuk input yang secara statistik serupa dengan distribusi data pelatihan. Input yang berada di luar distribusi pelatihan, atau yang dirancang khusus untuk mengeksploitasi batasan keputusan (*decision boundaries*) yang telah dipelajari oleh model, dapat menyebabkan model menghasilkan output yang tampak meyakinkan namun sebenarnya salah.

Alur persepsi juga rentan terhadap pergeseran distribusi (*distributional shift*), di mana karakteristik statistik lingkungan operasional berbeda dari data pelatihan, sehingga menurunkan kinerja model meskipun tidak ada input yang sengaja dirancang untuk menyerang sistem. Kondisi pencahayaan yang tidak biasa, pengaruh cuaca, atau objek dengan karakteristik visual yang kurang terwakili dalam kumpulan data pelatihan dapat menyebabkan penurunan kinerja deteksi, yang secara operasional memiliki implikasi keselamatan serupa dengan serangan yang disengaja.

Kekhawatiran lainnya berkaitan dengan integritas model itu sendiri. Kelemahan dalam proses pembaruan *Over-the-Air* (OTA) dan keamanan rantai pasok model dapat memungkinkan penyerang untuk mengganti atau memanipulasi model, sehingga memunculkan perilaku berbahaya yang sulit dideteksi melalui pengujian fungsional standar. Model yang telah dimanipulasi mungkin tetap berfungsi dengan benar dalam kondisi operasional normal, namun menghasilkan keluaran yang keliru pada skenario pemicu tertentu, hal ini dapat menyebabkan perilaku kendaraan yang tidak aman atau membahayakan keselamatan. Kelemahan spesifik yang terkait dengan persepsi *Deep Neural Network* (DNN) meliputi

- **Eksploitasi perturbasi adversarial pada batas keputusan (*decision boundaries*) DNN**: Input yang dirancang khusus untuk memaksimalkan kesalahan klasifikasi model tanpa dapat dideteksi oleh pengamat manusia menyebabkan model menghasilkan klasifikasi yang salah dengan tingkat keyakinan tinggi. Dalam sistem deteksi objek berbasis kamera, hal ini dapat diwujudkan secara fisik melalui penempatan adversarial *patch* (tambalan adversarial) dalam bidang pandang kamera. Efektivitas serangan ini dapat berpindah sebagian ke model lain yang memiliki karakteristik arsitektur serupa, sehingga membatasi nilai perlindungan dari kerahasiaan arsitektur model.
- **Penurunan kinerja akibat pergeseran distribusi (*distributional shift*) tanpa adanya serangan yang disengaja**: Kondisi operasional yang tidak terwakili dengan baik dalam data pelatihan seperti sudut sinar matahari yang tidak lazim, kontaminasi pada sensor, atau kemunculan objek yang belum pernah ditemui sebelumnya dapat menurunkan akurasi deteksi. Dampak terhadap keselamatannya setara dengan dampak dari serangan yang disengaja. Hal ini relevan dalam penilaian keamanan karena seorang

penyerang dapat dengan sengaja memicu kondisi tersebut tanpa perlu menghitung perturbasi adversarial yang rumit.

- **Penggantian model melalui pembaruan OTA (*Over-the-Air*) tanpa tanda tangan kriptografis**: Proses pembaruan OTA yang tidak mewajibkan verifikasi tanda tangan kriptografis sebelum mengaktifkan model baru memungkinkan penyerang yang berhasil menyusup ke dalam saluran pengiriman pembaruan atau infrastruktur distribusi untuk mengganti model yang sedang aktif dengan versi yang telah dimanipulasi. Model pengganti tersebut langsung diaktifkan pada kendaraan tanpa melalui validasi fungsional apa pun; akibatnya, perilaku yang salah atau berbahaya akan langsung muncul pada siklus inferensi berikutnya.
- **Manipulasi model dalam rantai pasok yang menciptakan backdoor laten**: Penyerang yang mampu memanipulasi atau mengganti model selama fase pengembangan, pelatihan, atau integrasi dapat menyusun model yang berperilaku normal dalam kondisi biasa, namun menghasilkan keluaran yang salah saat menerima masukan pemicu (*trigger inputs*) tertentu masukan yang biasanya tidak tercakup dalam rangkaian pengujian standar. Model yang telah disisipi backdoor ini kemudian lolos validasi sebelum penerapan (*pre-deployment*) dan dirilis sebagai bagian dari proses pengiriman perangkat lunak yang sah. Perilaku berbahaya tersebut tetap tidak aktif (*dormant*) sampai kondisi pemicu ditemui saat model digunakan di lapangan.

Kelemahan Asisten AI di Dalam Kendaraan

Asisten AI berbasis LLM di dalam kendaraan menerima masukan tidak hanya melalui perintah suara, tetapi juga melalui sumber data eksternal seperti deskripsi titik tujuan (*point of interest*) pada navigasi, pesan masuk, dan respons dari layanan yang terhubung yang semuanya diproses oleh model sebagai bagian dari konteksnya. Hal ini menciptakan kerentanan terhadap *prompt injection* (menurut *Open Worldwide Application Security Project* (OWASP) LLM Top 10: LLM01 Prompt Injection). Dalam skenario ini, konten dari sumber data tersebut yang memuat teks menyerupai instruksi dapat dianggap oleh model sebagai perintah otoritatif, bukan sekadar data pasif. Akibatnya, asisten AI mungkin menjalankan fungsi tertentu tanpa adanya interaksi dari pengguna yang sah. Berbeda dengan kelemahan validasi masukan konvensional, prompt injection tidak muncul akibat kesalahan penguraian (*parsing*), melainkan karena karakteristik mendasar LLM yang tidak dapat membedakan secara andal antara instruksi dan data ketika keduanya berada dalam jendela konteks yang sama.

Kelemahan lain yang serupa namun berbeda adalah *acoustic command injection* (injeksi perintah melalui akustik). Serangan ini mengeksploitasi perbedaan antara persepsi pendengaran manusia dan pemrosesan sinyal statistik yang dilakukan oleh lapisan pengenalan suara (*speech recognition*). Dengan menyisipkan gangguan (*perturbation*) yang telah dihitung secara cermat ke dalam sinyal audio, penyerang dapat menciptakan suara yang oleh pendengar manusia dianggap sebagai kebisingan latar belakang atau musik, namun oleh model pengenalan suara diterjemahkan sebagai perintah spesifik yang kemudian dieksekusi oleh asisten AI. Serangan ini terjadi pada lapisan pengenalan suara sebelum masukan

mencapai LLM, dan dapat disampaikan melalui audio yang diputar oleh perangkat di dekatnya, media siaran, atau konten yang dialirkan melalui sistem audio kendaraan itu sendiri.

Dampak praktis dari kedua kelemahan di atas sangat bergantung pada cakupan izin yang diberikan kepada asisten AI (menurut OWASP LLM Top 10: LLM06 *Excessive Agency* atau kewenangan berlebih). Asisten AI yang memiliki izin untuk mengubah pengaturan navigasi, melakukan panggilan, mengirim pesan, atau mengendalikan perangkat rumah pintar yang terhubung, merupakan target yang jauh lebih bernilai dibandingkan asisten AI yang hanya terbatas pada pemberian informasi. Kekhawatiran sistemik lainnya berkaitan dengan privasi data suara (menurut OWASP LLM Top 10: LLM02 *Sensitive Information Disclosure* atau pengungkapan informasi sensitif). Asisten di dalam kendaraan memproses bahasa lisan yang mungkin mencakup alamat rumah dan kantor, nama kontak, serta entri kalender. Jika data ini dikirimkan ke infrastruktur cloud tanpa enkripsi dan kontrol akses yang memadai, atau disimpan lebih lama dari yang diperlukan untuk kebutuhan operasional, hal tersebut dapat menimbulkan masalah privasi yang berkelanjutan.

Setelah meninjau kelemahan keamanan sensor yang umum, kini saatnya melihat bagaimana kelemahan-kelemahan tersebut dapat dimanfaatkan dalam kegiatan pengujian penetrasi (*penetration testing*).

9.4 TEKNIK PENGUJIAN PENETRASI TERHADAP SENSOR KENDARAAN DAN AI

Pengujian penetrasi pada sensor kendaraan dan sistem AI disusun berdasarkan permukaan serangan (*attack surface*) yang berbeda dari setiap komponen. Untuk penyederhanaan, bagian ini berfokus pada tiga fase utama yang berlaku dalam pengujian sensor dan sistem AI: penemuan (*discovery*), di mana penguji keamanan mengidentifikasi antarmuka sensor dan karakteristik sinyal secara pasif sebelum melakukan intervensi aktif; analisis kerentanan (*vulnerability analysis*), di mana temuan dari tahap penemuan dievaluasi untuk menentukan jalur serangan mana yang layak; dan eksploitasi (*exploitation*), di mana teknik-teknik yang relevan diterapkan untuk mengonfirmasi serta menunjukkan dampak dari kelemahan yang telah ditemukan.

Pengujian Sensor Inersia MEMS

Penguji keamanan memulai fase penemuan dengan mengidentifikasi model sensor target beserta frekuensi resonansinya, yang dapat ditemukan dalam lembar data (*datasheet*) atau literatur teknis. Penguji kemudian menentukan lokasi fisik sensor di dalam kendaraan serta mengevaluasi desain selubung (*enclosure*) dan jalur kopling pada papan sirkuit cetak (PCB) yang mungkin memengaruhi kopling akustik. Secara bersamaan, penguji mengidentifikasi bagaimana keluaran dari unit kontrol elektronik (ECU) yang menggunakan data sensor tersebut dapat diakses baik melalui antarmuka diagnostik kendaraan, lalu lintas jaringan CAN, maupun sinyal keluaran sensor khusus guna menetapkan titik observasi yang akan digunakan sebagai pembanding terhadap hasil dari stimulus yang diberikan nantinya.

Selanjutnya, analisis kerentanan berfokus pada penilaian apakah ECU yang menerima data tersebut menerapkan pemeriksaan plausibilitas (kewajaran) terhadap keluaran sensor inersia. Penguji penetrasi meninjau respons ECU terhadap pembacaan sensor dasar dan

menentukan apakah terdapat validasi rentang atau pemeriksaan konsistensi antar-sensor yang diterapkan pada keluaran akselerometer atau giroskop. Frekuensi resonansi dan penempatan sensor juga dievaluasi untuk menentukan tingkat tekanan suara (*sound pressure level*) serta sudut penyampaian yang diperlukan guna mencapai kopling akustik yang memadai, karena faktor-faktor inilah yang menentukan kelayakan praktis dari eksploitasi tersebut.

Terakhir, fase eksploitasi melibatkan pembuatan stimulus akustik pada frekuensi resonansi yang telah diidentifikasi (dan di sekitarnya) menggunakan sumber audio presisi. Penguji memvariasikan tingkat tekanan suara dan sudut penyampaian untuk memetakan cakupan serangan (*attack envelope*).

Selama pemberian stimulus, penguji dapat memantau keluaran ECU melalui antarmuka diagnostik atau jaringan kendaraan untuk mengamati apakah muncul pembacaan inersia yang tidak wajar, serta apakah pembacaan tersebut memengaruhi fungsi-fungsi yang berkaitan dengan keselamatan, seperti sistem kontrol stabilitas elektronik (*electronic stability control*). Perlu dicatat bahwa serangan ini tidak memerlukan akses ke jaringan kendaraan dan dapat dilakukan menggunakan peralatan penghasil audio yang tersedia secara komersial, sehingga menurunkan hambatan praktis untuk jenis serangan ini dibandingkan dengan teknik eksploitasi sensor lainnya yang dibahas dalam bab ini.

Pengujian TPMS

Tahap penemuan dimulai dengan pengintaian RF pasif, yaitu menangkap transmisi TPMS dari kendaraan target menggunakan penerima *Software-Defined Radio* (SDR) yang beroperasi pada pita frekuensi 315 MHz atau 433,92 MHz, bergantung pada standar regional yang berlaku. Pemilihan pita frekuensi yang tepat merupakan prasyarat agar penangkapan data berhasil, karena perangkat yang dikalibrasi untuk satu wilayah tidak akan dapat menerima transmisi dari kendaraan yang diproduksi untuk wilayah lain. Contoh perangkat yang sesuai mencakup RTL-SDR V3 dan HackRF One, yang memiliki sensitivitas serta cakupan frekuensi yang memadai untuk kedua pita tersebut. Selanjutnya, dekoding sinyal dapat dilakukan menggunakan rtl_433 yang telah dilengkapi dukungan protokol TPMS bawaan untuk berbagai jenis sensor atau menggunakan *Universal Radio Hacker* (URH) untuk melakukan demodulasi manual dan pemeriksaan protokol jika sensor target tidak dikenali oleh dekoder otomatis. Selain itu, GNU Radio dengan modul gr-tpms menyediakan lingkungan *flow-graph* yang fleksibel untuk menangkap, mendemodulasi, dan mengurai paket data secara *real-time*. Penguji penetrasi (*pentester*) dapat mencatat ID sensor, nilai tekanan, serta struktur paket dari berbagai sensor roda, lalu memastikan apakah ID sensor bersifat statis di setiap siklus transmisi; hal ini memungkinkan pelacakan pasif terhadap kendaraan target.

Berikutnya, pemeriksaan kerentanan difokuskan pada penilaian aspek autentikasi dalam struktur paket TPMS serta mekanisme penyaringan (*filtering*) pada penerima. Penguji penetrasi dapat memeriksa paket yang telah ditangkap untuk memastikan apakah terdapat kolom autentikasi kriptografis selain *checksum* CRC (yang hanya berfungsi mendeteksi kesalahan transmisi, bukan mengautentikasi sumber). Penyaringan berdasarkan plausibilitas (*plausibility filtering*) pada penerima dievaluasi dengan mengamati bagaimana sistem menerima paket dengan ID sensor yang sesuai; hal ini bertujuan untuk menentukan apakah

ketentuan regulasi mengenai peringatan penurunan tekanan secara cepat justru mendorong sistem untuk menerima paket dengan validasi yang minim. Mekanisme pendaftaran (*enrollment*) sensor juga diperiksa pada tahap ini apakah proses tersebut memerlukan akses tingkat diagnostik atau mendukung pendaftaran melalui BLE karena kedua faktor ini menentukan jalur eksploitasi mana yang dapat diterapkan.

Terakhir, fase eksploitasi mencakup injeksi paket, penyalahgunaan mekanisme pendaftaran (enrollment), dan penilaian jalur pendaftaran berbasis BLE (*Bluetooth Low Energy*), yang disesuaikan dengan kelemahan-kelemahan yang telah diidentifikasi sebelumnya. Dalam melakukan injeksi paket, seorang pentester menggunakan pemancar SDR (*Software-Defined Radio*) untuk memutar ulang paket yang telah ditangkap serta mengirimkan paket yang telah dimodifikasi misalnya dengan mengubah nilai tekanan guna menguji apakah sistem memicu peringatan palsu dan apakah penerima menerapkan penyaringan plausibilitas (*plausibility filtering*) terhadap data yang tidak masuk akal. Skenario pengujian meliputi injeksi nilai tekanan di bawah ambang batas, pelaporan tekanan nol pada keempat roda, serta pengiriman paket yang memuat ID sensor yang tidak terdaftar pada kendaraan; semua pengujian ini menargetkan kelemahan pada sistem penyaringan penerima yang terlalu longgar. Mekanisme pendaftaran dapat diuji dengan mengirimkan paket pendaftaran berformat benar yang menyertakan ID sensor pilihan penyerang dilakukan saat atau sesaat setelah peristiwa pemicu pendaftaran terjadi untuk mengevaluasi apakah perangkat pemancar asing dapat didaftarkan sebagai sensor yang sah. Untuk implementasi TPMS yang mendukung pendaftaran via BLE, penguji keamanan akan memeriksa jalur pemrosesan *advertisement* BLE menggunakan perangkat sniffer paket BLE, seperti Ubertooth One yang dikombinasikan dengan Wireshark. Fokus utama diberikan pada bagian kode yang memproses pesan pendaftaran, mengingat studi sebelumnya telah mendokumentasikan adanya kerentanan keamanan memori (*memory safety weaknesses*) pada bagian tersebut yang berpotensi untuk dieksploitasi.

Pengujian Sistem Kamera

Pada tahap penemuan (*discovery*), seorang penguji penetrasi (*pentester*) dapat memulai dengan inspeksi fisik untuk mengidentifikasi jenis kamera, tata letak pin konektor (*pinout*), serta varian tautan SerDes yang digunakan pada kendaraan target. Penguji dapat mengamati lalu lintas bus I2C secara pasif menggunakan `` `i2cdump` `` untuk memetakan konfigurasi register dan menetapkan tolok ukur pengaturan normal untuk eksposur, penguatan (*gain*), dan sinkronisasi. Secara bersamaan, penguji dapat mengidentifikasi fungsi ADAS apa saja yang memanfaatkan keluaran kamera seperti pengereman darurat otomatis, bantuan penjagaan jalur (*lane keeping assistance*), atau deteksi objek karena hal ini menentukan dampak hilir (*downstream*) jika serangan pada lapisan optik atau antarmuka berhasil dilakukan.

Pada tahap berikutnya, yaitu pemeriksaan kerentanan, penguji menilai aspek autentikasi pada aliran video SerDes maupun saluran kontrol I2C, serta mengevaluasi lapisan inferensi terhadap kerentanan terhadap input yang bersifat adversarial (sengaja dimanipulasi untuk mengecoh sistem). Penguji penetrasi memeriksa aliran SerDes untuk menentukan

apakah terdapat penanda autentikasi kriptografis dari ujung ke ujung (*end-to-end*), serta menggunakan `i2cdetect` untuk mendata perangkat yang dapat diakses pada bus I2C dan memeriksa apakah saluran kontrol menerapkan kendali akses sebelum menerima perintah penulisan register. Selain itu, sensitivitas spektral sensor gambar diperiksa untuk memastikan apakah rentang inframerah dekat (*near-infrared*) antara 850 nm hingga 940 nm berada dalam cakupan sensitivitas sensor, yang menentukan kelayakan untuk melakukan pengujian saturasi optik. Terakhir, model deteksi objek yang diterapkan dievaluasi terkait kerentanannya terhadap *adversarial patches* (tambalan yang dirancang untuk mengecoh model), berdasarkan arsitektur model tersebut serta riset yang telah dipublikasikan mengenai sistem serupa yang digunakan dalam produksi.

Pada fase eksploitasi, penguji penetrasi (*pentester*) melakukan berbagai pengujian, seperti saturasi optik, kegagalan deteksi akibat adversarial patch (tambalan adversarial), injeksi aliran data SerDes, dan manipulasi saluran kontrol I2C. Sebagai contoh, untuk melakukan pengujian saturasi inframerah-dekat (*near-infrared*), penguji mengarahkan pencahayaan ke apertur kamera dengan intensitas dan jarak yang bervariasi. Penguji kemudian mencatat ambang batas di mana artefak saturasi muncul pada bingkai (*frame*) keluaran dan saat fungsi deteksi ADAS mulai menurun kinerjanya, sembari memastikan bahwa efek tersebut tidak terlihat oleh penumpang kendaraan. Pengujian adversarial patch dapat dilakukan menggunakan kerangka kerja seperti *Adversarial Robustness Toolbox* (ART) atau Foolbox untuk membuat patch yang diterapkan pada objek dalam bidang pandang kamera. Penguji keamanan kemudian mengamati dampak lanjutannya terhadap perilaku ADAS, misalnya penekanan fungsi pengereman darurat otomatis (*automatic emergency braking*) saat sistem menghadapi rintangan yang telah ditempeli patch tersebut.

Selain itu, pengujian pada tingkat antarmuka memerlukan akses fisik ke sistem target. Sebagai contoh, injeksi pada tautan SerDes memerlukan akses fisik ke rangkaian kabel (*wiring harness*) serta penggunaan interposer SerDes yang dikonfigurasi untuk menangkap dan menyuntikkan kembali sinyal tautan tersebut. Penguji penetrasi kemudian menilai apakah aliran video yang telah dimanipulasi dan disuntikkan ke dalam tautan dapat diterima oleh ECU pemroses tanpa menimbulkan kesalahan. Di samping itu, pengujian saluran kontrol I2C menggunakan perintah `i2cset` untuk melakukan penulisan register secara terarah. Pengujian ini bertujuan untuk mengetahui apakah penulisan yang tidak terautentikasi ke dalam register konfigurasi dan mode dapat diterima oleh modul kamera, serta apakah penulisan tersebut mampu mengubah parameter seperti eksposur, gain, pengaturan waktu bingkai (*frame timing*), atau sinkronisasi, yang pada akhirnya dapat menurunkan kualitas gambar atau mengganggu operasional kamera.

Pengujian Radar

Pengujian penetrasi dimulai dengan fase penemuan yang mencakup karakterisasi pasif sinyal RF radar, sebelum transmisi aktif apa pun dilakukan. Dengan menggunakan penerima radio yang didefinisikan oleh perangkat lunak (*software-defined radio*) dan *flow graph* GNU Radio, penguji penetrasi dapat menangkap emisi radar serta menentukan parameter chirp termasuk kemiringan (*slope*), durasi, serta fleksibilitas frekuensi (*frequency agility*) atau

penjadwalan semu-acak (*pseudo-random scheduling*) yang diterapkan oleh radar tersebut. Secara paralel, penguji keamanan dapat mengidentifikasi antarmuka jaringan kendaraan yang membawa keluaran daftar objek dari radar, lalu mencatat format pesan, laju pembaruan (*update rate*), serta kolom autentikasi atau integritas apa pun yang dapat diamati. Pemeriksaan kerentanan berfokus pada penilaian aspek autentikasi, baik pada keluaran jaringan radar maupun antarmuka konfigurasinya. Penguji penetrasi mengevaluasi apakah daftar objek memuat tag autentikasi kriptografis, serta apakah modul radar menerima perintah konfigurasi dan kalibrasi tanpa autentikasi melalui bus CAN.

Parameter chirp yang diperoleh diperiksa untuk menentukan kelayakan injeksi tersinkronisasi yakni apakah bentuk gelombang tersebut cukup dapat diprediksi sehingga sinyal pantulan buatan dapat diatur waktunya secara tepat. Urutan chirp yang bersifat tetap atau memiliki kompleksitas rendah menghadirkan risiko yang lebih tinggi dibandingkan bentuk gelombang yang diacak atau fleksibel, karena hal tersebut secara signifikan mempermudah upaya eksploitasi aktif. Fase eksploitasi mencakup serangan pada lapisan RF serta upaya untuk mengelabui sistem fusi lintas-modal (*cross-modal fusion*). Penguji penetrasi dapat melakukan pengujian RF aktif dengan memancarkan sinyal pantulan chirp buatan yang disinkronkan dengan siklus emisi radar, kemudian memantau lapisan fusi untuk melihat apakah muncul objek semu (*phantom object*) dalam daftar objek. Jamming derau pita lebar (*broadband noise jamming*) dilakukan dengan memancarkan interferensi pada frekuensi operasional, lalu memantau apakah objek nyata menghilang karena berada di bawah ambang batas CFAR (*Constant False Alarm Rate*). Perlu dicatat bahwa semua pengujian yang melibatkan transmisi aktif pada pita frekuensi 76 hingga 81 GHz memerlukan izin resmi dari otoritas terkait dan harus dilaksanakan di lingkungan yang terkendali. Selain itu, pengujian untuk mengelabui fusi sensor melibatkan penyajian objek radar semu pada jarak tertentu, sembari menerapkan *patch adversarial* pada kamera yang mencegah deteksi objek yang sama; tujuannya adalah untuk mengamati apakah lapisan fusi menangani masukan yang kontradiktif tersebut dengan pendekatan yang konservatif, atau justru menerima pembacaan yang hanya berasal dari radar.

Pengujian LiDAR

Pertama, penguji penetrasi (*pentester*) memulai dengan fase penemuan melalui inspeksi fisik terhadap sensor untuk mengidentifikasi panjang gelombang operasionalnya apakah 905 nm atau 1550 nm serta memastikan apakah desain sensor tersebut bersifat berputar (*rotating*) atau *solid-state*. Karakteristik ini menentukan teknik serangan optik mana yang dapat diterapkan. Penguji keamanan juga dapat mencatat lalu lintas kontrol CAN menggunakan candump selama proses penyalaan sensor maupun saat beroperasi secara normal, guna mengidentifikasi pengenal pesan kontrol serta menetapkan tolok ukur untuk konfigurasi dan perintah mode yang diharapkan.

Secara paralel, aliran *point cloud* (awan titik) melalui *Automotive Ethernet* dapat ditangkap menggunakan Wireshark atau perangkat lunak diagnostik dari produsen sensor, kemudian didekode untuk mengonfirmasi format, laju pembaruan (*update rate*), dan kerangka koordinat. Dalam fase ini, penguji juga mengarakterisasi parameter waktu pulsa sensor, yang diperlukan untuk menilai kelayakan injeksi foton berbasis waktu (*timed photon injection*) serta

menentukan apakah mekanisme pulse *fingerprinting* (identifikasi unik pulsa) telah diimplementasikan. Pemeriksaan kerentanan mencakup penilaian terhadap aspek autentikasi pada antarmuka kontrol CAN maupun jalur data Ethernet, serta mengevaluasi permukaan serangan optik berdasarkan pola kelemahan yang telah diidentifikasi sebelumnya. Penguji penetrasi dapat mengirimkan pesan tertentu melalui saluran kontrol CAN untuk menguji apakah perintah mode, modifikasi sensitivitas deteksi, penulisan parameter kalibrasi, atau pemicu pembaruan *firmware* dapat diterima tanpa adanya otorisasi. Aliran *point cloud* diperiksa untuk melihat apakah terdapat kolom perlindungan integritas. Jika *pulse fingerprinting* ditemukan, penguji akan mengevaluasi implementasinya untuk menentukan apakah mekanisme tersebut dapat diakali secara statistik dengan memproyeksikan pulsa berfrekuensi tinggi yang tersebar di sepanjang siklus pemindaian.

Fase eksploitasi menguji setiap kelemahan pada lapisan optik yang ditemukan selama pemeriksaan kerentanan. Dalam melakukan injeksi foton berbasis waktu, penguji menggunakan sumber laser berpulsa dengan panjang gelombang yang sesuai dengan sensor disinkronkan dengan siklus pemindaian untuk menyisipkan objek palsu ke dalam *point cloud* pada posisi yang ditentukan oleh penyerang. Jika *pulse fingerprinting* digunakan untuk mencegah pemalsuan (*spoofing*), penguji akan memproyeksikan pulsa dengan laju pengulangan tinggi yang tersebar secara seragam di seluruh siklus pemindaian. Tujuannya adalah untuk melihat apakah setidaknya satu pulsa yang disuntikkan dapat memenuhi jendela waktu yang diharapkan secara kebetulan secara statistik, sehingga mampu melumpuhkan sebagian pemeriksaan *fingerprinting* tersebut. Pengujian saturasi apertur dilakukan dengan memberikan iluminasi inframerah-dekat secara terus-menerus ke arah apertur sensor, lalu mengukur seberapa luas sektor buta (*blind sector*) yang dihasilkan, di mana objek nyata menjadi tidak terdeteksi.

Penting agar pengujian optik aktif yang menggunakan sumber laser dilakukan dengan pengendalian keselamatan yang memadai, mengingat adanya risiko terhadap mata yang terkait dengan pencahayaan inframerah-dekat pada tingkat daya yang diperlukan untuk pengujian tersebut. Selain itu, pengujian penghindaran fusi sensor (*sensor fusion evasion*) melibatkan penggunaan saturasi apertur untuk menghilangkan objek nyata dari point cloud LiDAR, sementara pantulan radar untuk objek yang sama tetap ada. Penguji penetrasi kemudian mengamati apakah lapisan fusi mendeteksi ketidaksesuaian lintas modal tersebut atau justru menganggap objek tersebut tidak ada, hanya berdasarkan keluaran dari LiDAR.

Pengujian Ultrasonik

Tahap penemuan dimulai dengan penguji penetrasi (*penetration tester*) memeriksa target secara fisik untuk memastikan penempatan sensor dan mengidentifikasi jenis antarmuka baik LIN maupun PWM melalui susunan pin (*pinout*) konektor. Penguji kemudian memantau lalu lintas bus CAN atau LIN dari ECU bantuan parkir (parking assistance ECU) selama manuver kecepatan rendah untuk mengarakterisasi format data jarak, laju pembaruan (*update rate*), dan ambang batas deteksi. Tahap ini juga menetapkan interval pengulangan pulsa akustik serta menentukan apakah sensor menerapkan *echo gating* (penyaringan gema) mekanisme yang menolak sinyal yang tiba di luar rentang waktu yang diharapkan karena hal

ini secara langsung memengaruhi kelayakan injeksi sinyal berwaktu selama tahap eksploitasi. Tahap berikutnya, yaitu pemeriksaan kerentanan, menilai apakah sensor menerapkan validasi sumber gema; artinya, apakah sensor mampu membedakan antara pantulan asli dari pulsa yang dipancarkannya sendiri dengan sinyal akustik yang disuntikkan dari luar. Selain itu, penguji penetrasi dapat meninjau interval pengulangan pulsa dan jendela deteksi yang diperoleh selama tahap penemuan untuk menentukan presisi waktu yang diperlukan agar injeksi gema palsu berhasil.

Perlu dicatat bahwa hambatan perangkat keras untuk melakukan injeksi akustik terhadap sensor ultrasonik tergolong rendah, mengingat transduser yang beroperasi pada rentang 40 hingga 58 kHz tersedia secara komersial dalam berbagai perangkat elektronik konsumen dan komponen robotika, yang meningkatkan risiko praktis terkait kelemahan ini. Selain itu, penguji harus memeriksa apakah sensor atau ECU bantuan parkir menyediakan antarmuka konfigurasi yang memungkinkan perubahan ambang batas sensitivitas deteksi atau parameter waktu gema tanpa otorisasi; perubahan yang tidak sah pada pengaturan ini dapat memengaruhi secara langsung informasi mengenai objek penghalang yang dilaporkan oleh sensor.

Terakhir, tahap eksploitasi menguji kerentanan yang telah diidentifikasi. Sebagai contoh, penguji penetrasi dapat melakukan pengujian injeksi gema palsu dengan menempatkan transduser ultrasonik di dekat sensor target, lalu menyuntikkan pulsa akustik yang diatur waktunya sedemikian rupa sehingga menghasilkan laporan jarak sesuai dengan keinginan penyerang. Dengan memantau keluaran pada bus CAN atau LIN, penguji dapat memastikan apakah sinyal pantulan yang disuntikkan tersebut diproses oleh ECU bantuan parkir sebagai deteksi objek penghalang yang nyata. Selain itu, pengujian interferensi berkelanjutan melibatkan pemancaran energi akustik secara terus-menerus pada frekuensi operasional sensor. Pengujian ini bertujuan untuk melihat apakah tingkat kebisingan (*noise floor*) meningkat cukup tinggi hingga menghalangi sensor mendeteksi objek penghalang yang sebenarnya berada dalam jangkauannya. Dalam skenario parkir otomatis, hal ini dapat menyebabkan sistem menganggap tidak adanya sinyal pantulan sebagai ruang kosong, sehingga kendaraan terus bergerak hingga menabrak objek penghalang yang sesungguhnya.

Pengujian Asisten AI dan LLM

Untuk pengujian asisten AI dan LLM, fase penemuan berfokus pada enumerasi antarmuka dan penilaian rantai pasok. Penguji keamanan harus mengidentifikasi semua sumber data yang diizinkan untuk diakses oleh asisten seperti titik tujuan navigasi, pesan masuk, entri kalender, dan respons API dari layanan yang terhubung serta memetakan seluruh fungsi kendaraan yang dapat dipicu oleh asisten tersebut. Dalam proses ini, perlu dicatat fungsi mana yang memerlukan konfirmasi eksplisit dari pengguna dan mana yang dapat diakses hanya melalui perintah percakapan. Saluran pembaruan OTA (*Over-the-Air*) juga diperiksa untuk memastikan apakah paket pembaruan model telah ditandatangani secara kriptografis dan apakah verifikasi tanda tangan tersebut benar-benar diterapkan sebelum aktivasi.

Selanjutnya, pemeriksaan kerentanan dilakukan untuk menilai cakupan izin asisten serta tingkat kepercayaan yang diterapkan pada setiap sumber data yang diakses. Penguji

penetrasi meninjau fungsi-fungsi yang telah dipetakan untuk mengidentifikasi fungsi yang berkaitan dengan keselamatan atau privasi yang dapat diakses tanpa langkah konfirmasi eksplisit; cakupan izin yang terlalu luas akan memperbesar dampak dari serangan *prompt injection* maupun *acoustic command injection*. Setiap sumber data yang teridentifikasi dievaluasi untuk menentukan apakah kontennya dianggap sebagai input tepercaya atau data yang tidak tepercaya (yang memerlukan validasi sebelum dimasukkan ke dalam konteks prompt model). Selain itu, penguji keamanan dapat memeriksa apakah rekaman suara dan log percakapan dikirimkan ke infrastruktur awan (*cloud*). Jika ya, perlu dipastikan apakah transmisi tersebut dienkripsi dan apakah terdapat kebijakan penyimpanan terdokumentasi yang mengatur durasi penyimpanan data tersebut (OWASP LLM Top 10: LLM02 *Sensitive Information Disclosure*).

Fase eksploitasi kemudian mencakup pengujian seperti *acoustic command injection*, *prompt injection*, dan pengujian cakupan pemanggilan alat (*tool call scope*), yang diselaraskan dengan kelemahan-kelemahan yang telah diidentifikasi sebelumnya. Sebagai contoh, penguji penetrasi dapat melakukan *acoustic command injection* dengan membuat *audio adversarial examples* yaitu menyisipkan gangguan yang tidak terdengar oleh telinga manusia ke dalam audio yang diputar melalui sistem audio kendaraan atau sumber suara di dekatnya kemudian memantau keluaran pengenalan suara untuk memastikan apakah transkripsi target berhasil dihasilkan.

Pengujian *prompt injection* (OWASP LLM Top 10: LLM01 *Prompt Injection*) berfokus pada penyusunan konten yang menyerupai instruksi di dalam instans terkendali dari setiap sumber data yang telah diidentifikasi seperti titik navigasi yang dimuat dari kumpulan data uji atau pesan yang dikirimkan melalui perangkat uji yang terhubung kemudian mengamati apakah model memperlakukan konten yang disisipkan tersebut sebagai perintah dan berupaya mengeksekusinya tanpa adanya interaksi dari pengguna. Selain itu, pengujian cakupan pemanggilan alat (*tool call scope testing*) menyasar kelemahan agensi berlebih (*excessive agency*—OWASP LLM Top 10: LLM06 *Excessive Agency*) dengan cara mencoba memanggil fungsi-fungsi sensitif melalui jalur percakapan tidak langsung tanpa memicu permintaan konfirmasi. Pengujian ini mendokumentasikan kategori fungsi mana saja yang dapat diakses tanpa otorisasi eksplisit dari pengguna, sekaligus memastikan batasan izin efektif yang dimiliki oleh asisten tersebut. Setelah memahami berbagai teknik pengujian penetrasi yang umum digunakan, kini saatnya menelaah beberapa studi kasus nyata di mana sensor serta sistem AI/ML berhasil disusupi atau dikompromikan.

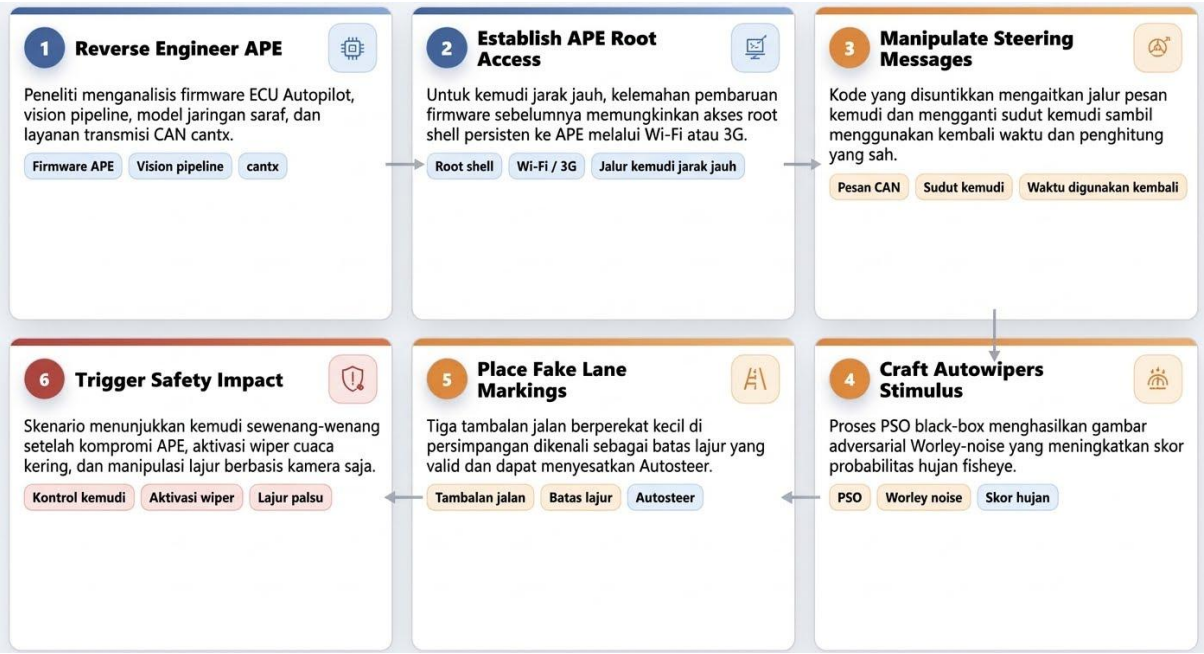
9.5 STUDI KASUS SERANGAN NYATA TERHADAP SENSOR KENDARAAN DAN SISTEM

Pada bagian ini, kami mengkaji tiga serangan dunia nyata terhadap sensor kendaraan dan sistem AI/ML, serta memeriksa bagaimana penyerang mengidentifikasi dan mengeksploitasi kelemahan dalam alur inferensi pembelajaran mesin (*machine learning*), lapisan penginderaan fisik, dan tumpukan komunikasi sensor nirkabel. Studi kasus ini mengilustrasikan bagaimana permukaan serangan yang dijelaskan pada bagian sebelumnya dapat ditemukan dan dieksploitasi dalam praktik. Secara keseluruhan, studi ini mencakup

berbagai skenario ancaman mulai dari manipulasi adversarial terhadap sistem persepsi jaringan saraf tiruan (*neural network*) yang digunakan dalam produksi, pemalsuan fisik sensor LiDAR menggunakan peralatan berbiaya rendah, hingga rantai eksekusi kode jarak jauh (*remote code execution*) tanpa interaksi pengguna (*zero-click*) yang masuk ke dalam kendaraan melalui sensor TPMS.

Studi kasus 1: Serangan pembelajaran mesin adversarial terhadap Tesla Autopilot

Studi kasus ini mengkaji tiga skenario serangan yang dikembangkan oleh peneliti keamanan terhadap sistem Tesla Autopilot pada kendaraan Model S. Para peneliti melakukan rekayasa balik (*reverse engineering*) terhadap *Autopilot* ECU (APE) dan tumpukan komunikasi bus CAN, kemudian merancang serta mendemonstrasikan serangan terhadap tiga fungsi Autopilot: kemampuan pengenalan rambu lalu lintas, sistem otonom berbasis jaringan saraf tiruan, dan sistem deteksi lajur jalan. Temuan yang paling signifikan secara operasional adalah serangan lajur palsu (*fake lane attack*), di mana tiga tempelan kecil yang diletakkan di permukaan jalan pada sebuah persimpangan menyebabkan sistem Autosteer mengarahkan kendaraan ke jalur yang berlawanan, tanpa memerlukan akses jaringan apa pun ke kendaraan tersebut. Studi ini diungkapkan kepada publik pada tahun 2019, dan Tesla kemudian mengatasi kerentanan akses root pada APE melalui pembaruan *firmware*. Gambaran umum mengenai langkah-langkah serangan untuk studi kasus ini disajikan dalam Gambar 9.2:



Gambar 9.2 Bagaimana kompromi perangkat lunak dan manipulasi sensor di dunia fisik terjadi

Penemuan dan Pengintaian

Para peneliti memulai dengan melakukan rekayasa balik statis dan penelusuran kesalahan (*debugging*) dinamis pada *firmware* APE, yang menjalankan lingkungan Linux yang telah disesuaikan dengan perangkat lunak Autopilot yang tersimpan di direktori /opt/*autopilot*. Modul APE berbasis pada platform PX2 AutoChauffeur dan terhubung ke

semua kamera utama yaitu kamera utama (*main*), kamera sudut sempit (*narrow*), dan kamera mata ikan (*fisheye*) secara eksklusif melalui koneksi *Camera Serial Interface* (CSI), sementara data radar diteruskan melalui koprosesor Infineon Aurix.

Alur pemrosesan visual (*vision pipeline*) diidentifikasi sebagai komponen sentral yang memproses bingkai gambar dari kamera melalui jaringan saraf tiruan (*neural network*) besar yang digunakan bersama. Jaringan ini menangani berbagai tugas secara simultan, termasuk deteksi lajur, pelacakan objek, dan pemetaan lingkungan. Fungsi penghapus kaca depan otomatis (*autowipers*) menggunakan jaringan saraf tiruan khusus untuk kamera mata ikan yang didefinisikan dalam berkas fisheye.prototxt; jaringan ini berfungsi sebagai pengklasifikasi hujan yang menghasilkan nilai probabilitas antara 0 hingga 1, yang merepresentasikan tingkat kemungkinan adanya kelembapan pada kaca depan. Deteksi lajur beroperasi menggunakan citra beresolusi 1280x960 dengan rentang dinamis tinggi (*High Dynamic Range*/HDR) dari kamera utama dan kamera sudut sempit, yang kemudian menghasilkan peta probabilitas lajur berukuran 416x640 melalui serangkaian kernel *Compute Unified Device Architecture* (CUDA).

Terkait komponen bus CAN, rekayasa balik terhadap layanan cantx mengungkap protokol APE2LB_CAN sebuah bus CAN logis yang ditransmisikan melalui *User Datagram Protocol* (UDP). Protokol ini mengarahkan pesan dari APE ke koprosesor Infineon Aurix, yang selanjutnya meneruskan pesan tersebut ke bus CAN untuk sasis, bodi, dan sistem penggerak (*powertrain*) kendaraan. Pesan bernama *DasSteeringControlMessage* yang memuat informasi sudut kemudi, jenis kontrol, penghitung pesan (*message counter*), dan checksum satu byte diidentifikasi sebagai pesan kunci yang digunakan oleh unit El*ectric Power Assisted Steering* (EPAS) untuk mengeksekusi perintah kemudi dalam mode *Adaptive Cruise Control* (ACC) dan *Automatic Parking Control* (APC).

Akses Awal

Penting untuk dicatat bahwa ketiga skenario serangan tersebut memerlukan tingkat akses yang berbeda. Pertama, serangan ini bergantung pada akses root pada APE, yang diperoleh melalui kerentanan dalam mekanisme pembaruan *firmware*. Akses awal didasarkan pada pendekatan yang dipresentasikan di Black Hat USA 2018, yang menggunakan rantai serangan jarak jauh melalui jaringan 3G atau Wi-Fi untuk menyusupi *gateway* kendaraan terlebih dahulu, sebelum bergerak secara lateral untuk mencapai APE. Layanan pembaruan *firmware* pada APE menerima citra *firmware* SquashFS yang di-hosting melalui *Hypertext Transfer Protocol* (HTTP) tanpa autentikasi yang memadai. Dengan menyajikan citra *firmware* yang sah pada permintaan pertama dan menggantinya dengan citra berbahaya pada permintaan kedua, para peneliti membuat APE menjalankan skrip pembaruan yang telah disisipi *backdoor* dengan hak akses *root*. Skrip ini memodifikasi konfigurasi *Secure Shell* (SSH) dan aturan *firewall*, sehingga memberikan akses *shell root* yang persisten kepada para peneliti di APE tersebut. Sebaliknya, serangan lainnya tidak memerlukan akses fisik maupun akses jarak jauh ke kendaraan sama sekali. Serangan-serangan ini beroperasi sepenuhnya melalui lingkungan fisik yang diamati oleh kamera kendaraan; artinya, penyerang yang berada dalam jangkauan pandang kamera kendaraan dapat melancarkan serangan tersebut tanpa perlu terhubung ke jaringan kendaraan atau mengendalikan perangkat lunak ECU apa pun.

Eksploitasi

Eksploitasi untuk masing-masing dari ketiga skenario serangan tersebut dijelaskan sebagai berikut. Pertama, serangan ini menggunakan teknik injeksi kode dinamis ke dalam layanan cantx yang berjalan pada APE yang telah disusupi. Teknik ini dilakukan dengan mengaitkan (*hooking*) fungsi `*DasSteeringControlMessageEmitter*::*finalize_message*()` untuk mencegat dan mengganti nilai sudut kemudi dengan nilai yang dikendalikan oleh penyerang, sembari tetap menggunakan stempel waktu (*timestamp*) dan penghitung (*counter*) pesan yang sah. Langkah ini berhasil melewati perlindungan integritas pesan CAN dan memungkinkan pengiriman perintah kemudi arbitrer ke unit EPAS.

Serangan ini didemonstrasikan saat kendaraan beroperasi dalam mode ACC (*Adaptive Cruise Control*), di mana sistem Autosteer secara aktif mengelola pergerakan lateral kendaraan; dengan demikian, perintah yang disuntikkan dapat secara langsung mengubah lintasan kendaraan yang seharusnya. Para peneliti mendemonstrasikan skenario ini menggunakan sebuah *gamepad* yang berfungsi untuk menghasilkan input sudut kemudi yang dihubungkan melalui *Bluetooth* ke perangkat seluler; perangkat tersebut kemudian menerjemahkan sudut-sudut kemudi itu menjadi nilai-nilai *DasSteeringControlMessage* dan mengirimkannya ke layanan APE cantx yang telah disusupi melalui jaringan 3G atau Wi-Fi.

Selanjutnya, serangan terhadap sistem penyeka kaca otomatis (*autowipers*) menggunakan pendekatan contoh adversarial *black-box* (kotak hitam). Jaringan saraf tiruan untuk kamera fisheye tersebut dikompilasi dengan kuantisasi bilangan bulat 8-bit dan implementasi kernel CUDA privat parsial, yang membuat komputasi internalnya sulit diakses untuk pemeriksaan eksternal, sehingga para peneliti memperlakukannya sebagai sebuah *black box*. Mereka menerapkan *Particle Swarm Optimization* (PSO) untuk menghasilkan gangguan (*perturbation*) pada citra yang dapat memaksimalkan skor probabilitas hujan pada jaringan saraf tiruan tersebut, tanpa perlu mengakses bobot atau gradien internalnya. Citra yang dihasilkan mampu meningkatkan skor hingga melampaui ambang batas aktivasi, namun tetap terlihat sama persis bagi pengamat manusia. Awalnya, para peneliti mencoba menggunakan *salt-and-pepper noise* (derau bintik hitam-putih), tetapi resolusi kamera ternyata tidak cukup untuk menangkap detail tersebut. Mereka kemudian menguji pola Worley noise, yang terbukti lebih cocok untuk tugas ini kemungkinan karena jaringan deteksi hujan tersebut merespons terutama terhadap tekstur citra.

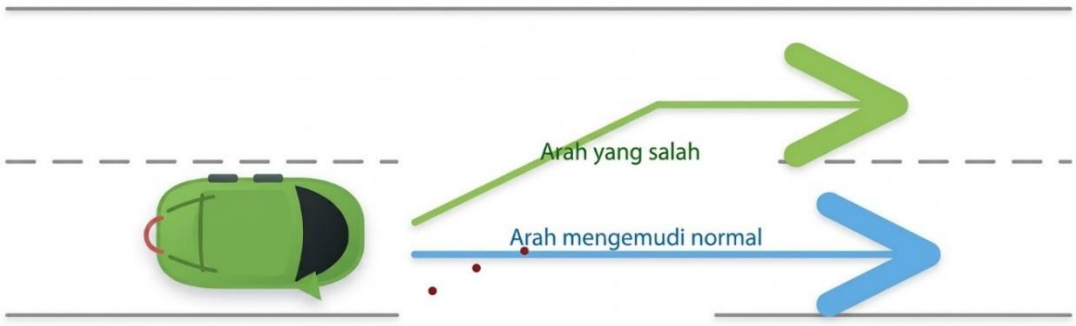
Untuk implementasi di dunia nyata, citra adversarial berbasis pola-pola tersebut dibuat dan dioptimalkan secara menyeluruh (*end-to-end*) menggunakan kendaraan uji yang memiliki akses root ke sistem autowipers. Setiap kandidat citra ditampilkan pada layar di dalam bidang pandang kamera *fisheye*, ditangkap oleh kamera tersebut, dievaluasi oleh jaringan saraf tiruan *autowipers*, lalu skor probabilitas hujan yang dihasilkan digunakan untuk menyempurnakan citra secara berulang hingga ambang batas aktivasi tercapai. Pendekatan ini memastikan bahwa distorsi dunia nyata seperti distorsi lensa, karakteristik layar, dan kondisi pencahayaan telah diperhitungkan selama proses pembuatan citra. Perlu dicatat bahwa akses root hanya diperlukan selama fase pengembangan citra ini. Setelah citra adversarial berhasil dibuat, penerapannya pada kendaraan target tidak lagi memerlukan akses jaringan maupun akses fisik

ke kendaraan tersebut, serta tidak membutuhkan eksploitasi perangkat lunak apa pun. Cukup dengan menempatkan layar yang menampilkan citra yang telah diproses sebelumnya dalam jangkauan kamera *fisheye* kendaraan target, sistem penyeka kaca otomatis dapat terpicu untuk aktif meskipun dalam kondisi cuaca kering.

Istilah:

- Kuantisasi bilangan bulat 8-bit (8-bit *integer quantization*) adalah teknik yang mengurangi parameter numerik jaringan saraf tiruan dari nilai titik mengambang (*floating-point*) 32-bit menjadi bilangan bulat 8-bit, sehingga mengurangi kebutuhan memori dan komputasi untuk menjalankan model pada perangkat keras tertanam (*embedded hardware*).
- *Particle Swarm Optimization* (PSO) adalah algoritma optimasi berbasis populasi yang mencari solusi dengan menyesuaikan sekumpulan kandidat input secara berulang berdasarkan kinerja individu maupun kelompok, tanpa memerlukan akses ke struktur internal fungsi target. Citra adversarial (*adversarial images*) adalah input untuk jaringan saraf tiruan yang dirancang secara sengaja untuk menghasilkan keluaran yang salah biasanya dengan menambahkan perubahan halus pada tingkat piksel yang tidak dapat dideteksi oleh pengamat manusia, namun secara signifikan mengubah prediksi model.
- Skor probabilitas hujan (*rain probability score*) adalah keluaran numerik dari jaringan saraf tiruan untuk fitur penyeka kaca depan (*autowipers*) pada kendaraan Tesla, yang merepresentasikan estimasi model mengenai kemungkinan kaca depan dalam kondisi basah. Ketika nilai ini melampaui ambang batas yang telah ditentukan, sistem akan mengaktifkan penyeka kaca depan.
- Pola derau Worley (*Worley noise patterns*) adalah jenis tekstur prosedural yang digunakan dalam grafika komputer untuk menyimulasikan permukaan alami, seperti batu atau struktur seluler.

Serangan lanc Kcíccíion menghasilkan hasil yang paling signifikan secara operasional dengan potensi kerusakan keselamatan. Para peneliti menemukan bahwa pengenalan jalur Tesla Autopilot dalam mode Autosteer sepenuhnya bergantung pada jaringan saraf visi komputer, tanpa validasi silang terhadap input lain seperti data peta Sistem Pemosisian Global (GPS) atau geometri jalan yang diketahui. Dalam percobaan ini, tiga stiker perekat kecil yang ditempatkan di persimpangan mensimulasikan kelanjutan jalur kanan yang melengkung melintasi garis tengah. Ketika kendaraan dalam mode Autosteer mencapai persimpangan, jaringan saraf mengenali stiker tersebut sebagai batas jalur yang valid, mengabaikan marka jalan dan jalur sebenarnya, dan dengan demikian mengarahkan kendaraan ke jalur yang berlawanan, seperti yang diilustrasikan pada Gambar 9.3:



Gambar 9.3 Serangan jalur palsu menggunakan tiga stiker di dunia nyata

Persistensi

Dalam serangan terhadap sistem kemudi, *root shell* yang persisten pertama-tama dibuat pada APE melalui konfigurasi SSH yang telah disisipi *backdoor*. Hal ini memungkinkan para peneliti untuk terhubung kembali ke kendaraan kapan saja melalui Wi-Fi atau 3G tanpa perlu mengulangi rangkaian eksploitasi awal. Sebaliknya, serangan terhadap sistem otonom dan sistem deteksi lajur tidak mencapai persistensi dalam pengertian tradisional yaitu dengan menanamkan pijakan perangkat lunak. Sebagai gantinya, kedua serangan ini bergantung pada keberadaan objek fisik di lingkungan sekitar kendaraan target dan tidak meninggalkan jejak apa pun yang dapat dideteksi dalam sistem kendaraan tersebut. Sebagai contoh, stiker yang ditempelkan di permukaan jalan untuk menyerang sistem deteksi lajur seperti yang ditunjukkan pada Gambar 9.3 dapat dipasang dan dilepas tanpa memerlukan akses langsung ke kendaraan itu sendiri.

Dampak

Serangan terhadap sistem kemudi menunjukkan bahwa setelah APE berhasil dikompromikan, pesan CAN palsu dapat disuntikkan langsung ke aktuator kemudi untuk mengambil alih kendali kemudi secara penuh, bahkan pada kecepatan tinggi. Saat fitur *Autosteer* (kemudi otomatis) sedang aktif mengendalikan pergerakan lateral kendaraan, perintah-perintah palsu ini berdampak langsung pada lintasan kendaraan, yang berpotensi menyebabkan kendaraan keluar dari lajurnya atau bahkan terperosok keluar dari badan jalan.

Serangan terhadap sistem otonom dan sistem deteksi lajur bekerja dengan prinsip yang sama sekali berbeda; serangan ini tidak memerlukan akses jaringan maupun eksploitasi kerentanan perangkat lunak. Dalam kedua kasus tersebut, stimulus yang diberikan kepada jaringan saraf tiruan berbasis kamera sudah cukup untuk memicu respons otomatis yang tidak diinginkan: sebuah gambar adversarial yang ditampilkan pada layar elektronik di depan kamera *fisheye* membuat jaringan saraf tiruan untuk fitur penyeka kaca otomatis (*autowipers*) mengklasifikasikan kondisi lingkungan sebagai hujan, sehingga penyeka kaca pun aktif. Selain itu, tiga stiker kecil yang ditempelkan di persimpangan jalan terbukti cukup untuk membuat sistem *Autosteer* mengarahkan kendaraan masuk ke jalur yang berlawanan arah.

Ketiga skenario ini menggambarkan bahwa *attack surface* (permukaan serangan) pada sistem kecerdasan buatan (AI) otomotif tidak hanya mencakup tumpukan perangkat lunak, tetapi juga meluas hingga ke lingkungan fisik yang diamati oleh *sensor-sensor* kendaraan.

Manipulasi adversarial pada lapisan persepsi dapat melampaui mekanisme keselamatan di tingkat perangkat lunak ketika kendaraan tidak memiliki cara untuk membedakan input asli dari input yang direkayasa baik berupa pesan CAN palsu dari ECU yang telah disusupi, stimulus optik keliru yang diarahkan ke sensor, maupun marka jalan yang dibuat menggunakan stiker perekat.

Studi kasus 2: Pemalsuan (*spoofing*) LiDAR melalui perekaman dan pemutaran ulang pulsa

Dalam serangan ini, para peneliti keamanan menunjukkan bahwa sistem LiDAR yang diandalkan oleh sebagian besar kendaraan otonom dapat diserang dengan perangkat yang biayanya sekitar Rp 600.000. Studi yang dipresentasikan pada ajang Black Hat Europe 2015 ini memperlihatkan bahwa penyerang dapat menyisipkan objek semu (*phantom objects*) pada jarak tertentu dalam bidang pandang sensor yang mencakup 110 derajat. Hal ini dapat memaksa kendaraan otonom untuk melakukan pengereman yang tidak perlu, atau membanjiri sistem pelacakan objek dengan sinyal palsu dalam jumlah cukup banyak sehingga objek nyata di lingkungan sekitar tidak terdeteksi. Studi ini menunjukkan bahwa lapisan penginderaan fisik pada kendaraan otonom merupakan permukaan serangan yang sebagian besar belum tertangani, terlepas dari ada atau tidaknya kerentanan pada perangkat lunak maupun komunikasi jaringan. Gambar 9.4 menyajikan gambaran umum mengenai tahapan dalam rangkaian serangan tersebut:



Gambar 9.4 Pemutaran ulang pulsa *time-of-flight* yang tidak diautentikasi untuk memalsukan objek nyata dan menghabiskan kapasitas pelacakan

Penemuan dan Pengintaian

Investigasi para peneliti bermula dari pengamatan bahwa sensor lingkungan utama pada platform kendaraan otonom di masa itu khususnya sistem LiDAR beroperasi dengan

prinsip yang tidak menyertakan perlindungan integritas bawaan. Logika dasarnya sederhana: jika kendaraan swakemudi menerima input sensor yang rusak, maka kendaraan tersebut akan mengambil keputusan mengemudi yang buruk. Target mereka adalah unit LiDAR ibeo LUX 3, sebuah sensor yang tersedia secara komersial dan digunakan dalam skenario lalu lintas perkotaan maupun perjalanan di jalan raya. Para peneliti menemukan bahwa sistem ibeo LUX 3 menggunakan pulsa yang tidak disertai penyandian (*encoding*), enkripsi, ataupun penanda temporal (*temporal fingerprint*). Hal ini berarti perangkat yang mampu menerima dan memancarkan kembali pulsa-pulsa tersebut dengan penundaan (*delay*) yang terkontrol dapat mensimulasikan keberadaan objek pada jarak tertentu secara meyakinkan.

Akses Awal

Serangan ini tidak memerlukan akses ke sistem internal, antarmuka jaringan, atau perangkat lunak kendaraan. Penyerang hanya perlu berada dalam jangkauan sensor LiDAR target, dengan perangkat keras laser pemancar yang mampu beroperasi hingga jarak 100 meter baik dari posisi di depan, samping, maupun belakang sensor tanpa perlu membidik unit tersebut secara presisi. Perangkat serangan ini terdiri dari dioda laser, penerima fotodetektor, dan dua generator pulsa, yang dirakit dari komponen-komponen umum dengan total biaya kurang dari Rp 600.000. Karena pulsa yang dipancarkan oleh ibeo LUX 3 tidak memiliki penyandian atau enkripsi, para peneliti dapat merekam dan memancarkannya kembali dengan penundaan yang diatur. Satu-satunya aspek teknis yang menantang dalam pengaturan ini adalah sinkronisasi, khususnya memastikan bahwa sinyal yang dipancarkan kembali diarahkan ke sensor dalam rentang waktu yang tepat.

Eksploitasi

Setelah tahap perekaman selesai, para peneliti mendemonstrasikan dua jenis serangan: serangan relai (*relay attack*) yang memanipulasi persepsi lokasi objek nyata, dan serangan pemalsuan (*spoofing attack*) yang menyisipkan objek semu (*phantom objects*) yang sebenarnya tidak ada. Keduanya mengeksploitasi kelemahan mendasar yang sama, yaitu bahwa pulsa ibeo LUX 3 tidak disertai penyandian atau autentikasi, sehingga perangkat apa pun yang mampu menerima dan memancarkan kembali pulsa tersebut dapat memengaruhi persepsi sensor.

Dalam serangan relai, dua perangkat pemancar-penerima (*transceiver*) ditempatkan di sekitar unit ibeo LUX 3; satu perangkat berfungsi menerima pulsa LiDAR yang keluar, sementara perangkat lainnya memancarkannya kembali dari posisi berbeda dengan penundaan yang terkontrol. Karena sensor menghitung jarak berdasarkan waktu tempuh pulang-pergi pulsa, sinyal yang dipancarkan kembali tersebut menyebabkan dinding nyata yang berjarak 1 meter tampak sebagai gema palsu pada jarak sekitar 20 dan 50 meter dari sensor. Perlu dicatat bahwa serangan relay tidak memerlukan garis pandang langsung ke sensor, karena pulsa LiDAR yang dipantulkan dan melewati perangkat tersebut dapat dicegat lalu dipancarkan kembali dari posisi di dekatnya.

Serangan *spoofing* memperluas kemampuan ini dengan menggunakan pulsa asli sebagai pemicu untuk menghasilkan pulsa palsu dengan penundaan waktu yang diatur secara presisi, sehingga menciptakan point cloud (kumpulan titik) semu pada jarak tertentu di dalam

bidang pandang sensor yang mencakup 110 derajat. Para peneliti mendemonstrasikan dua varian serangan *spoofing*. Pada varian pertama, sebuah dinding semu disisipkan pada jarak antara 40 hingga 70 meter; sistem pelacakan berbasis Kalman filter yang tertanam dalam perangkat ibeo LUX 3 berhasil mengklasifikasikan dan melacak objek semu tersebut. Pada lapisan aplikasi, sistem kemudi otomatis yang menerima input semacam itu akan menganggap objek semu tersebut sebagai rintangan dan melakukan pengereman meskipun konsekuensi ini tidak didemonstrasikan secara langsung dalam pengaturan laboratorium. Pada varian serangan *spoofing* kedua, para peneliti menunjukkan bahwa penyisipan objek palsu menyebabkan ibeo LUX 3 mengalokasikan sumber daya pelacakannya secara keliru. Mereka juga mencatat bahwa jika jumlah objek palsu yang dimasukkan cukup banyak, kapasitas sensor untuk melacak hingga 65 objek secara bersamaan dapat habis, yang berpotensi menyebabkan objek nyata di lingkungan sekitar tidak terdeteksi.

Persistensi

Karena serangan relay maupun *spoofing* beroperasi sepenuhnya pada lapisan fisik melalui injeksi sinyal optik, serangan-serangan ini tidak berinteraksi secara langsung dengan tumpukan perangkat lunak (*software stack*) atau antarmuka jaringan kendaraan. Dengan demikian, entri apa pun yang tercatat dalam log perangkat lunak hanya akan mencerminkan keluaran sensor seperti deteksi objek yang tidak terduga atau anomali pelacakan dan bukan manipulasi optik yang menyebabkannya.

Selain itu, penyerang yang beroperasi dari tepi jalan atau dari kendaraan yang melaju di dekatnya dapat mempertahankan serangan tersebut selama sensor target masih berada dalam jangkauan operasional. Perlu dicatat pula bahwa perangkat penyerang memancarkan pulsa dalam rentang inframerah dekat pada panjang gelombang 905 nm, yang tidak terlihat oleh mata manusia; selain itu, kedua jenis serangan ini tidak memerlukan kontak fisik dengan kendaraan target. Oleh karena itu, penyerang yang berada di tepi jalan atau di kendaraan lain di dekatnya tidak akan mudah terdeteksi secara visual, sehingga serangan dapat terus berlangsung.

Dampak

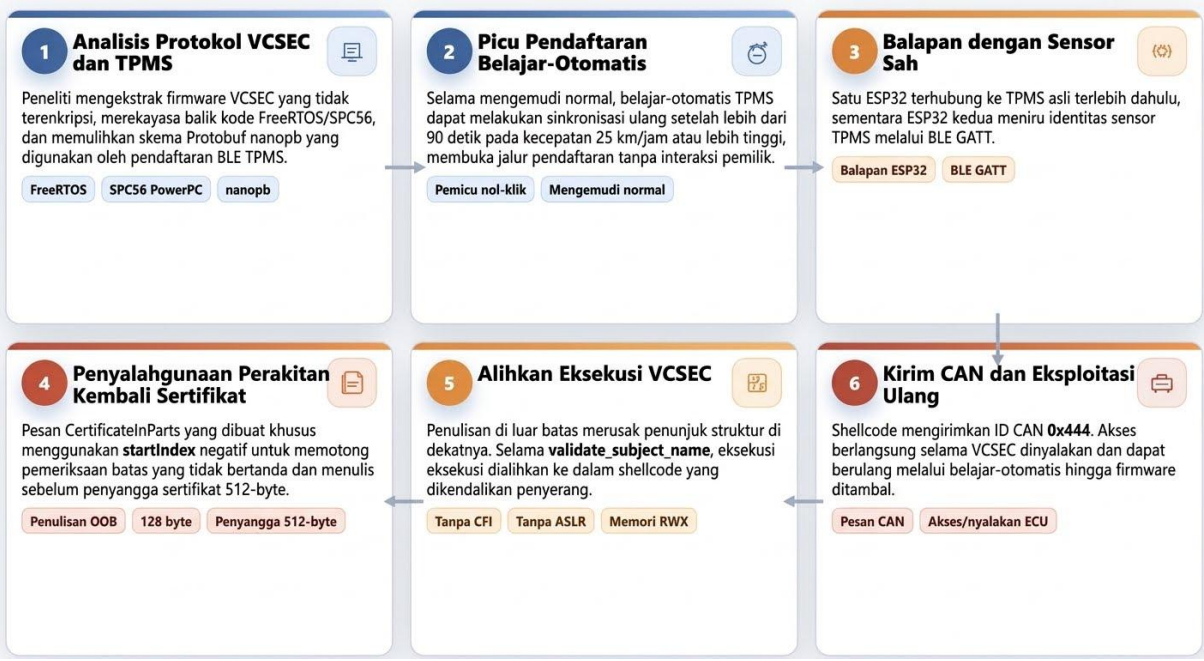
Implikasi dari studi ini melampaui satu produk atau platform tertentu. Serangan *spoofing sensor* memanfaatkan saluran fisik yang sama dengan sensor target untuk memanipulasi pembacaan pada titik perolehan data, sebelum pemrosesan digital apa pun terjadi. Karena serangan ini tidak menimbulkan anomali apa pun dalam tumpukan perangkat lunak maupun antarmuka komunikasi, keluaran sensor yang telah dimanipulasi tersebut tidak dapat dibedakan dari pengukuran yang sah saat mencapai tahap pemrosesan data.

Pada saat publikasi, belum ada produsen LiDAR yang menerapkan pertahanan terhadap jenis serangan ini. Studi tersebut menyimpulkan bahwa seluruh kategori sensor optik *time-of-flight* (waktu tempuh cahaya) secara struktural rentan terhadap serangan *replay* (pengulangan) dan *spoofing* (pemalsuan) jika pulsa yang dipancarkan tidak disertai identitas kriptografis atau pengacakan waktu. Generasi sistem LiDAR berikutnya mulai mengatasi masalah ini melalui teknik seperti pengodean pulsa (*pulse coding*) dan interval pemeriksaan acak, meskipun efektivitas langkah-langkah penanggulangan ini dalam kondisi serangan aktif

masih terus diteliti. Dari perspektif keamanan siber otomotif, para peneliti menyusun model ancaman awal untuk serangan terhadap integritas LiDAR. Model ini tetap menjadi acuan dalam memahami bagaimana lapisan penginderaan fisik pada kendaraan otonom dan sistem bantuan pengemudi (ADAS) dapat dimanipulasi tanpa perlu mengakses perangkat lunak atau infrastruktur komunikasi.

Studi kasus 3: Eksekusi kode jarak jauh tanpa interaksi pengguna (*zero-click*) melalui sensor TPMS

Pada ajang Pwn2Own Vancouver 2024, para peneliti keamanan mendemonstrasikan kerentanan pada Tesla Model 3 yang memungkinkan penyerang menjalankan kode arbitrer (sewenang-wenang) pada unit kontrol elektronik (ECU). Rangkaian serangan ini bermula dari akses melalui sistem pemantauan tekanan ban (TPMS), di mana kelemahan dalam proses pengaturan sensor menyebabkan terjadinya penulisan data di luar batas memori (*out-of-bounds write*). Celah ini memungkinkan peneliti untuk mengambil kendali atas ECU yang krusial bagi keamanan, yakni ECU yang bertanggung jawab atas akses kendaraan dan fungsi penyalaan mesin. Hasil studi ini kemudian dipresentasikan di Hexacon 2024 pada bulan Oktober tahun yang sama. Eksploitasi ini tidak memerlukan interaksi apa pun dari pengguna untuk dapat menjalankan kode dari jarak jauh pada ECU pengatur akses tersebut. Gambaran umum mengenai tahapan serangan dalam studi kasus ini disajikan pada Gambar 9.5.



Gambar 9.5 Bagaimana fitur autolearn pada TPMS serta bug dalam penyusunan ulang sertifikat dapat berujung pada eksekusi kode berbahaya

Penemuan dan Pengintaian

Unit Kontrol Kendaraan Sekunder (*Vehicle Controller Secondary*/VCSEC) mengelola data tekanan ban sekaligus bertanggung jawab atas berbagai fungsi kendaraan lainnya. Unit ini berfungsi sebagai titik akhir (*endpoint*) bagi aplikasi seluler Tesla, yang menangani kontrol

akses kendaraan serta proses penyalaan mesin. Untuk mendukung fungsi-fungsi tersebut, VCSEC berkomunikasi melalui beberapa antarmuka nirkabel, seperti BLE untuk data sensor TPMS dan konektivitas aplikasi Tesla, serta *Near-Field Communication* (NFC) untuk akses kendaraan tanpa kunci (*keyless*). VCSEC juga terhubung dengan bus CAN kendaraan, mengirim dan menerima pesan CAN standar, serta mendukung penyediaan (*provisioning*) dan pemeliharaan berbasis *Unified Diagnostics Services* (UDS).

Para peneliti memulai investigasi mereka dengan mengekstrak citra *firmware* VCSEC dari sistem berkas akar (*root filesystem*) sistem infotainmen Tesla, tempat *firmware* untuk berbagai revisi perangkat keras disimpan tanpa enkripsi. VCSEC sendiri menjalankan FreeRTOS pada *system-on-chip* ST Microelectronics SPC56 PowerPC. Rekayasa balik (*reverse engineering*) dilakukan menggunakan IDA Pro [26], yang menyediakan dukungan dekompilasi untuk set instruksi PowerPC *Variable Length Encoding* (VLE) yang digunakan oleh SPC56. Dua pustaka standar ditemukan di dalam *firmware* tersebut: Mbed-TLS, yang menangani *Transport Layer Security* (TLS) dan operasi sertifikat X.509, serta nanopb, sebuah implementasi Protocol Buffer yang dirancang untuk sistem tertanam (*embedded systems*) dengan memori terbatas. Keduanya relevan dengan rantai serangan yang dibahas dalam subbagian berikutnya.

Karena komunikasi antara TPMS dan VCSEC berbasis *Protocol Buffer* (Protobuf) format serialisasi biner yang tidak dapat diinterpretasikan tanpa skema pesan yang mendasarinya maka pemulihan definisi pesan Protobuf dari *firmware* menjadi prasyarat untuk memahami dan merekonstruksi pesan yang dipertukarkan selama pendaftaran sensor. Untuk tujuan ini, para peneliti menggunakan perangkat *anvilsecure*/nanopb-decompiler. Mengingat SPC56 adalah prosesor dengan arsitektur *big-endian*, sementara perangkat tersebut awalnya dirancang untuk target *little-endian*, para peneliti melakukan modifikasi (*patch*) agar perangkat tersebut mendukung penyandian big-endian.

Sensor TPMS berkomunikasi dengan VCSEC melalui BLE menggunakan karakteristik *Generic Attribute Profile* (GATT), dengan pesan-pesan yang diserialisasi menggunakan Protobuf. Perlu dicatat bahwa skema. Proto yang mengatur komunikasi antara TPMS dan aplikasi Tesla telah diekstraksi sebagian dari aplikasi seluler Tesla oleh komunitas peneliti, yang memberikan referensi berguna untuk memahami struktur protokol tersebut. Sebagai contoh, proyek TeslaProtobufs di GitHub, yang mencakup berkas vcsec.proto, merupakan salah satu contohnya.

Akses Awal

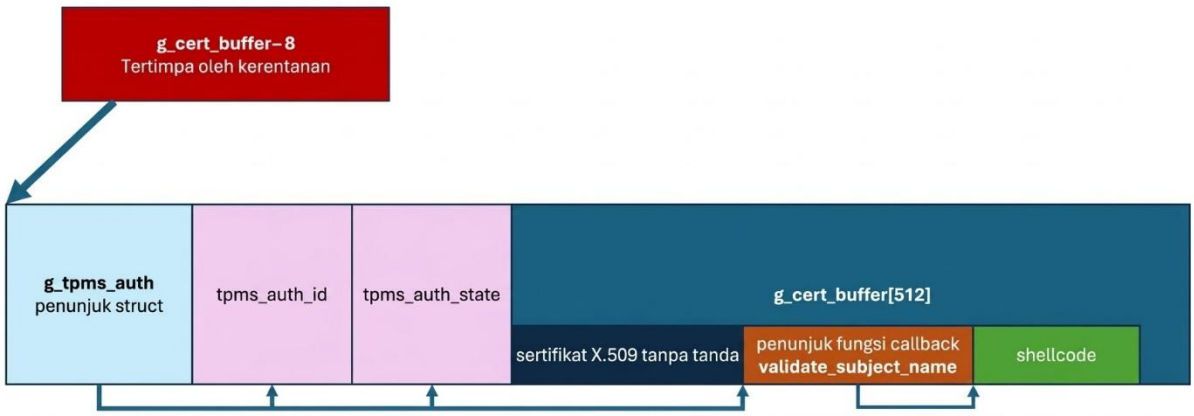
Sebelum VCSEC dapat menerima data dari sensor TPMS, sensor tersebut harus menyelesaikan proses pendaftaran (*enrollment*). Untuk sensor TPMS tipe 5, proses pendaftaran ini juga memerlukan autentikasi berbasis sertifikat guna memverifikasi keabsahan sensor; dalam tahap ini, VCSEC akan meminta sertifikat X.509 dari TPMS. Karena kemampuan penanganan sertifikat ini belum diterapkan pada sensor yang diproduksi secara massal pada saat studi dilakukan, para peneliti harus melakukan emulasi terhadap sensor TPMS tipe 5 untuk memicu jalur kode yang rentan dalam proses penyusunan ulang sertifikat (*certificate reassembly*).

Penangan penyusunan ulang sertifikat di dalam *firmware* VCSEC menerima pesan Protobuf `CertificateInParts` yang terdiri dari tiga kolom: `startIndex`, yang menentukan offset byte di dalam buffer sertifikat global tempat setiap fragmen yang masuk akan ditulis; `certificateSize`, yang menentukan total panjang byte dari sertifikat yang sedang disusun; serta buffer data dengan ukuran maksimum 128 byte yang dibatasi oleh `nanopb`. Pemeriksaan batas (*bounds check*) menghitung jumlah 32-bit tak bertanda (*unsigned*) dari `startIndex` dan `data_size`, lalu membandingkan hasilnya dengan batas buffer sertifikat sebesar 512 byte.

Para peneliti menemukan bahwa pemilihan nilai negatif yang tepat untuk `startIndex` dapat menyebabkan *overflow* pada operasi penjumlahan tak bertanda dalam pemeriksaan batas tersebut, sehingga menghasilkan nilai yang cukup kecil untuk lolos dari pemeriksaan. Operasi `memcpy` berikutnya kemudian menggunakan `startIndex` sebagai offset bertanda (*signed offset*) ke dalam buffer sertifikat, yang memungkinkan penulisan hingga 128 byte data yang dikendalikan oleh penyerang ke lokasi memori sebelum *buffer* tersebut. Hal ini menciptakan kerentanan penulisan di luar batas (*out-of-bounds write*) dengan kapasitas hingga 128 byte data yang dapat dikendalikan oleh penyerang.

Eksploitasi

Sebuah penunjuk (*pointer*) ke struktur yang memuat penunjuk fungsi (*function pointer*) terletak tepat sebelum buffer sertifikat global di dalam memori. Dengan menimpa penunjuk struktur ini untuk mengarahkannya ke *buffer* yang dikendalikan oleh penyerang, serta menempatkan *payload shellcode* di dalam buffer tersebut, para peneliti berhasil mengalihkan eksekusi saat pemanggilan validasi sertifikat tepatnya pada fungsi `validate_subject_name`. Pada titik ini, VCSEC akan langsung melompat ke *shellcode* milik penyerang, sebagaimana diilustrasikan dalam Gambar 9.6:



Gambar 9.6 Alur eksploitasi - penimpaan penunjuk struktur (*structure pointer*) sebelum `g_cert_buffer` mengalihkan eksekusi ke shellcode yang dikendalikan oleh penyerang [diadaptasi dari 1G]

Firmware VCSEC tidak dilengkapi dengan perlindungan *Control Flow Integrity* (CFI) maupun *Address Space Layout Randomization* (ASLR). Selain itu, *Memory Management Unit* (MMU) / *Memory Protection Unit* (MPU) pada *firmware* tersebut tidak dikonfigurasi, sehingga seluruh

wilayah memori dapat dibaca, ditulis, dan dieksekusi secara bersamaan. *Shellcode* dikompilasi dari kode C menggunakan toolchain `powerpc-eabivle` dan memanggil fungsi-fungsi *firmware* pada alamat yang telah diketahui.

Pendaftaran (*enrollment*) sensor TPMS pada VCSEC dapat dimulai melalui sesi UDS. Namun, karena para peneliti bertujuan untuk mendemonstrasikan eksploitasi *zero-click* yaitu serangan yang tidak memerlukan interaksi dari pemilik kendaraan maupun akses ke antarmuka diagnostik mereka meneliti mekanisme *auto-learn* (pembelajaran otomatis) TPMS milik Tesla sebagai alternatif untuk memicu jalur kode pendaftaran yang rentan. Ketika kendaraan Tesla memenuhi kriteria untuk *auto-learn* yakni berkendara selama lebih dari 90 detik dengan kecepatan minimal 25 km/jam VCSEC dan TPMS akan mencoba melakukan sinkronisasi ulang. Hal ini menciptakan peluang untuk menyisipkan sensor TPMS palsu ke dalam proses pendaftaran tersebut.

Skenario serangan ini menggunakan dua perangkat BLE. Perangkat pertama, sebuah papan pengembangan ESP32, diprogram untuk terhubung ke sensor TPMS yang sah sebelum VCSEC sempat melakukannya; tindakan ini secara efektif menghalangi akses VCSEC ke sensor tersebut selama fase *auto-learn*. Perangkat ESP32 kedua dikonfigurasi sebagai simulator sensor TPMS yang merespons pesan pendaftaran dari VCSEC, sekaligus mengirimkan *payload* berbahaya `CertificateInParts` dengan nilai `startIndex` yang telah dimanipulasi untuk memicu penulisan di luar batas memori (*out-of-bounds write*).

Implementasi awal yang menggunakan BlueZ terbukti terlalu lambat untuk memenangkan persaingan koneksi secara andal. Oleh karena itu, para peneliti menerapkan logika koneksi tersebut langsung pada perangkat keras ESP32, yang menghasilkan tingkat keberhasilan yang jauh lebih konsisten. Setelah eksploitasi berhasil, *shellcode* mengirimkan pesan CAN arbitrer melalui ID CAN 0x444 di bus CAN kendaraan. Hal ini menunjukkan bahwa penyerang yang menyamar sebagai sensor TPMS melalui BLE dapat menyuntikkan pesan ke dalam bus CAN untuk mengendalikan fungsi-fungsi kendaraan, tanpa perlu akses fisik ke kendaraan maupun port diagnostiknya.

Persistensi

Persistensi dalam serangan ini bekerja pada dua tingkatan. Saat sistem berjalan (*runtime), shellcode* yang dieksekusi pada VCSEC memberikan akses ke bus CAN kendaraan dan komunikasi BLE selama ECU tetap mendapatkan daya. Jika VCSEC dimulai ulang (*restart*), *shellcode* tersebut akan hilang; namun, selama kerentanan yang mendasarinya masih ada dalam *firmware*, penyerang dapat mengeksploitasinya kembali. Kondisi *auto-learn* (pembelajaran otomatis) akan muncul secara alami selama penggunaan kendaraan sehari-hari yaitu saat kendaraan melaju selama lebih dari 90 detik dengan kecepatan minimal 25 km/h sehingga memungkinkan penyerang untuk mengeksploitasi kerentanan yang sama berulang kali dalam sesi berkendara yang berbeda, tanpa memerlukan akses fisik ke kendaraan. Dalam skenario yang lebih lanjut, penyerang dengan kemampuan *shellcode* yang memadai dapat mencoba menulis kode persisten ke dalam penyimpanan *flash firmware*. Kode ini akan tetap ada meskipun sistem dimulai ulang, sehingga memberikan akses berkelanjutan ke bus CAN dan fungsi kontrol akses BLE kendaraan. Ketiadaan perlindungan seperti CFI, ASLR, dan

MMU/MPU pada *firmware* membuat sistem tidak memiliki pertahanan terhadap upaya semacam itu.

Dampak

VCSEC adalah ECU yang krusial bagi keamanan; komponen ini mengelola kontrol akses kendaraan dan memberikan otorisasi untuk menyalakan mesin. Dengan akses ke bus CAN kendaraan, penyerang yang berhasil mengeksekusi kode pada VCSEC dapat mengirimkan pesan CAN arbitrer dan memengaruhi fungsi-fungsi kendaraan secara langsung. Perlu dicatat bahwa sifat eksploitasi yang tidak memerlukan interaksi pengguna (*zero-click*) secara signifikan menurunkan hambatan untuk melakukan serangan. Tidak diperlukan tindakan apa pun dari pemilik kendaraan, sementara kondisi pemicunya yakni kendaraan melaju selama lebih dari 90 detik dengan kecepatan minimal 25 km/h terjadi secara alami selama penggunaan kendaraan yang normal.

Perangkat yang digunakan untuk serangan ini terdiri dari dua papan pengembangan ESP32 yang tersedia secara komersial serta komponen BLE standar. Hal ini memungkinkan perangkat serangan tersebut dirakit dengan biaya rendah dan disembunyikan di dalam perangkat yang diletakkan di pinggir jalan, atau di dalam kendaraan lain yang melaju di dekat kendaraan target. Tesla menanggapi kerentanan tersebut menyusul pengungkapan terkoordinasi dengan merilis pembaruan *firmware* yang memperbaiki masalah *integer overflow* (peluapan bilangan bulat) serta menambahkan pemeriksaan validasi pada variabel-variabel terkait. Menangani kelemahan keamanan umum yang berdampak pada sensor kendaraan dan sistem AI/ML memerlukan ketelitian proses yang sama seperti yang kita terapkan pada komponen kendaraan lainnya. Meskipun penting untuk mengatasi kelemahan-kelemahan ini secara sistematis, pemahaman mengenai praktik terbaik keamanan yang dapat membantu menghilangkan kelemahan tersebut tetaplah bermanfaat; inilah yang akan kita bahas selanjutnya.

9.6 PRINSIP DAN STRATEGI PENGUATAN KEAMANAN SENSOR KENDARAAN SERTA SISTEM AI

Kelemahan serta teknik pengujian penetrasi yang dijelaskan dalam bab ini mengarah pada serangkaian prinsip tingkat tinggi yang harus menjadi pedoman dalam perancangan, integrasi, dan pengoperasian sensor kendaraan dan sistem AI yang aman. Prinsip-prinsip ini dirancang agar dapat diterapkan di sepanjang siklus hidup pengembangan dan mencakup berbagai teknologi sensor serta AI yang dibahas dalam bab ini, alih-alih menetapkan implementasi teknis spesifik untuk setiap jenis sensor.

Menerapkan Pertahanan Berlapis (*defense-in-depth*) Pada Seluruh Tumpukan (*stack*) Sensor dan AI

Penting untuk dipahami bahwa tidak ada satu pun sensor atau komponen AI yang boleh dianggap sebagai satu-satunya sumber kebenaran (*source of truth*) untuk pengambilan keputusan yang krusial bagi keselamatan. Arsitektur persepsi dan perencanaan kendaraan harus dirancang sedemikian rupa sehingga serangan terhadap satu antarmuka sensor seperti serangan *spoofing* radar yang menyisipkan objek semu tidak menyebabkan kendaraan

melakukan tindakan yang tidak aman tanpa adanya bukti pendukung dari input sensor lainnya. Logika fusi sensor harus dirancang untuk mendeteksi dan menandai anomali yang tidak konsisten di antara berbagai modalitas sensor; selain itu, keputusan yang krusial bagi keselamatan sebaiknya memerlukan kesesuaian dari berbagai sumber independen jika memungkinkan.

Prinsip ini juga berlaku untuk output inferensi AI. Klasifikasi model persepsi terhadap rambu lalu lintas, objek, atau pengguna jalan lainnya harus diperlakukan sebagai *input* probabilistik dalam alur pengambilan keputusan, bukan sebagai kebenaran mutlak (*ground truth*). Arsitektur kendaraan secara keseluruhan harus mencakup logika pemantauan yang mampu mengidentifikasi situasi di mana *output* sistem AI dianggap tidak masuk akal berdasarkan konteks fisik yang ada. Dengan demikian, penerapan pemeriksaan konsistensi antar-sensor sebagai bentuk pertahanan seperti membandingkan daftar objek dari radar dengan *point cloud* LiDAR serta deteksi objek dari kamera merupakan salah satu pendekatan paling praktis untuk menangani serangan *spoofing* yang canggih. Pemeriksaan fusi sensor ini tepat karena banyak teknik *spoofing* justru paling efektif jika hanya diterapkan pada satu modalitas sensor saja.

Lebih lanjut, dalam platform HPC (*High-Performance Computing*), jalur penerimaan data sensor harus diperlakukan sebagai batasan perangkat lunak berisiko tinggi, terlepas dari jenis sensor yang digunakan. Komponen perangkat lunak yang bertanggung jawab untuk menerima dan menormalisasi data sensor mentah harus beroperasi dengan hak akses minimal serta terisolasi secara ketat dari tumpukan aplikasi lainnya. Komponen ini harus memvalidasi keaslian dan kebaruan data sebelum menyalurkannya ke aplikasi pengguna, sehingga aliran data sensor yang telah disusupi atau dimanipulasi tidak dapat langsung masuk ke dalam alur persepsi tanpa melalui titik validasi yang terkendali.

Autentikasi Data Sensor Pada Tingkat Antarmuka

Jika protokol dan jenis antarmuka mendukungnya, data sensor harus menyertakan tag autentikasi kriptografis yang memungkinkan ECU penerima untuk memverifikasi asal, keaslian, serta kebaruan (*freshness*) data tersebut. Hal ini berlaku langsung pada sistem TPMS, di mana penambahan kode autentikasi pesan (*message authentication code*) pada paket yang dikirimkan dapat menghilangkan serangan pemalsuan (*spoofing*) dan serangan pengulangan (*replay attack*) sebagaimana dijelaskan dalam bagian mengenai kelemahan sistem. Untuk antarmuka di dalam kendaraan, seperti tautan kamera SerDes, autentikasi menyeluruh (*end-to-end*) terhadap aliran video harus dievaluasi selama perancangan sistem, dengan mempertimbangkan beban kinerja (*performance overhead*) dari proses autentikasi tersebut terhadap persyaratan latensi aplikasi persepsi.

Selain autentikasi data, ECU penerima juga harus memverifikasi identitas modul sensor itu sendiri. Jika didukung, setiap modul sensor harus dilengkapi dengan kredensial unik seperti sertifikat X.509 atau kunci simetris yang dapat diuji oleh ECU penerima selama inisialisasi untuk memastikan bahwa perangkat yang terhubung adalah modul sensor yang asli dan resmi. Autentikasi identitas sensor ini membantu memitigasi risiko penggantian dengan sensor palsu atau tiruan, di mana perangkat yang memiliki antarmuka elektrik yang valid justru

menghasilkan data rekayasa atau manipulasi yang mungkin diterima begitu saja oleh ECU penerima.

Untuk sensor yang terhubung ke jaringan, praktik komunikasi terautentikasi yang dijelaskan dalam bagian C dapat diterapkan. *Secure Onboard Communication* (SecOC) dari AUTomotive *Open System Architecture* (AUTOSAR) menyediakan mekanisme yang terdefinisi dengan baik untuk menambahkan autentikasi pesan dan informasi kebaruan data ke dalam lalu lintas CAN dan CAN-FD. Untuk sensor dengan bandwidth lebih tinggi yang menggunakan *Automotive Ethernet*, *Media Access Control Security* (MACsec) menyediakan kerahasiaan, integritas, dan autentikasi asal data pada lapisan tautan (*link-layer*). Perlu dicatat bahwa MACsec hanya memberikan perlindungan pada lapisan tautan; setelah frame melewati *switch* dan diproses secara *hop-by-hop*, integritas menyeluruh dari keluaran sensor hingga ke aplikasi persepsi memerlukan kontrol tambahan pada lapisan aplikasi (*application-layer*).

Untuk antarmuka sensor di mana autentikasi kriptografis tidak dapat diterapkan karena sifat komunikasi yang searah (*unidirectional*) atau terbatas sistem penerima harus menerapkan pemeriksaan plausibilitas (*plausibility checking*) sebagai kontrol kompensasi. Pembacaan yang berada di luar rentang yang masuk akal secara fisik, atau yang menyimpang secara drastis dari nilai historis terkini tanpa adanya perubahan yang selaras pada dinamika kendaraan, harus ditandai untuk evaluasi lebih lanjut, alih-alih langsung digunakan tanpa validasi.

Merancang Sistem AI Dengan Mempertimbangkan Ketahanan Terhadap Serangan (*adversarial robustness*)

Model jaringan saraf (*neural network*) yang diterapkan dalam sistem persepsi kendaraan harus dievaluasi ketahanannya terhadap serangan sebagai bagian dari proses pengembangan dan validasi. Pengujian tidak hanya mencakup akurasi pada input yang bersih (normal), tetapi juga harus mencakup berbagai skenario penyalahgunaan yang representatif. Teknik pelatihan adversarial di mana model dilatih menggunakan campuran antara input yang bersih dan input yang telah dimanipulasi secara sengaja dapat meningkatkan ketahanan terhadap kategori serangan yang sudah diketahui. Penggunaan beberapa model yang dilatih secara independen untuk saling memeriksa keluaran satu sama lain dapat menjadi langkah pengamanan arsitektural terhadap input adversarial yang mencoba mengeksploitasi batasan keputusan (*decision boundaries*) dari satu model tunggal. Perlu dicatat bahwa jika jalur data sensor menjadi input bagi model persepsi berbasis pembelajaran mesin (ML), maka alur persepsi tersebut harus dianggap sebagai bagian dari permukaan serangan (*attack surface*) sensor itu sendiri. Dengan demikian, pengujian melalui metode *fuzzing* pada jalur data serta evaluasi ketahanan terhadap pergeseran distribusi data (*distributional shift*) harus dimasukkan ke dalam lingkup pengujian keamanan untuk sensor tersebut.

Melindungi Asisten AI di Dalam Kendaraan dari Injeksi dan Penyalahgunaan

Asisten berbasis LLM di dalam kendaraan harus beroperasi di bawah model perizinan yang didefinisikan dan ditegakkan secara jelas, yang menentukan fungsi kendaraan apa saja yang boleh diakses oleh asisten tersebut serta dalam kondisi seperti apa. Perizinan harus diberikan dengan mengikuti prinsip hak akses minimum (*principle of least privilege*), dan asisten tidak boleh memiliki wewenang untuk memicu fungsi kendaraan yang berkaitan

dengan keselamatan hanya melalui perintah percakapan tanpa adanya langkah konfirmasi eksplisit.

Sumber data yang boleh diakses oleh asisten di dalam kendaraan seperti data navigasi, entri kalender, dan respons dari layanan yang terhubung harus diperlakukan sebagai input yang tidak tepercaya (*untrusted inputs*) dan wajib melalui validasi konten sebelum dimasukkan ke dalam konteks prompt model. Perlu dicatat bahwa jika asisten memproses konten eksternal, harus ada kontrol teknis untuk mencegah konten yang menyerupai instruksi di dalam data tersebut mengambil alih arahan tingkat sistem (*system-level directives*) milik asisten.

Data pribadi yang diproses oleh asisten AI di dalam kendaraan harus tunduk pada kebijakan minimalisasi dan retensi data yang didokumentasikan dengan jelas. Rekaman suara, riwayat navigasi, dan log percakapan hanya boleh disimpan selama jangka waktu minimum yang diperlukan untuk menjalankan fungsi pengumpulannya, serta harus dilindungi dengan enkripsi dan kontrol akses yang memadai, baik saat data sedang dikirim maupun saat disimpan.

Menangani Keamanan Sensor di Sepanjang Siklus Hidup Pengembangan

Pertimbangan keamanan sensor dan AI harus diintegrasikan ke dalam sistem manajemen keamanan siber kendaraan sejak tahap pengembangan paling awal. Proses ini harus mengikuti praktik terbaik untuk siklus hidup pengembangan perangkat lunak yang aman (*secure software development lifecycle*), serta selaras dengan persyaratan standar ISO/SAE 21434. Pemeriksaan Ancaman dan Penilaian Risiko (*Threat Analysis and Risk* Assessment atau TARA) untuk komponen sensor dan AI harus secara eksplisit mempertimbangkan kategori serangan yang dijelaskan dalam bab ini, termasuk manipulasi sinyal fisik, input yang bersifat adversarial (*adversarial inputs*), dan eksploitasi asisten AI. Hasil penilaian risiko tersebut kemudian harus menjadi dasar bagi persyaratan keamanan yang diterapkan selama tahap perancangan, implementasi, verifikasi, hingga pemantauan pascaproduksi. Pengujian penetrasi terhadap sistem sensor dan kecerdasan buatan (AI) harus disertakan dalam kegiatan validasi keamanan siber kendaraan, dengan cakupan yang meliputi komponen sensor individual maupun sistem persepsi dan AI yang terintegrasi.

Selain itu, pemantauan pascaproduksi harus dijalankan pada dua tingkat. Deteksi anomali saat waktu berjalan (*runtime*) yang diterapkan pada kendaraan akan memantau keluaran dari sistem sensor dan persepsi, serta memberikan peringatan jika perilaku yang teramati menyimpang dari batasan operasional yang diharapkan seperti deteksi objek yang tiba-tiba tidak masuk akal, atau munculnya objek tanpa adanya perubahan yang selaras pada dinamika kendaraan. Peringatan ini kemudian dikumpulkan dan diperiksa di dalam *Vehicle Security Operations Center* (VSOC) milik produsen kendaraan. Di sini, korelasi data di seluruh armada dapat mengidentifikasi pola tertentu, misalnya anomali berulang yang terjadi pada beberapa kendaraan, yang mungkin mengindikasikan serangan berskala luas atau kerentanan yang sedang dieksploitasi pada teknologi sensor yang digunakan.

Lebih lanjut, pelibatan pemasok sangatlah penting, mengingat sensor dan komponen AI biasanya diperoleh dari pemasok khusus, bukan dikembangkan secara internal oleh produsen kendaraan. Persyaratan keamanan siber untuk sensor dan komponen AI harus

ditetapkan dalam perjanjian dengan pemasok sebagai bagian dari perjanjian antarmuka keamanan siber khusus dan produsen kendaraan wajib memverifikasi melalui kegiatan penilaian yang tepat bahwa persyaratan tersebut telah dipenuhi sebelum komponen diintegrasikan ke dalam kendaraan.

Terakhir, rantai pasok pengembangan AI harus tunduk pada kontrol integritas yang sama dengan yang diterapkan pada rantai pasok perangkat lunak. Hal ini mencakup upaya untuk memastikan bahwa berkas model AI dan kumpulan data pelatihan (*training datasets*) diperoleh dari sumber yang terverifikasi, disimpan dengan kontrol akses yang memadai, serta divalidasi menggunakan tanda tangan kriptografis sebelum diterapkan. Perlu dicatat bahwa kumpulan data pelatihan menjadi perhatian khusus; penyerang yang mampu menyisipkan sampel berbahaya ke dalam data pelatihan yang dikenal sebagai serangan peracunan data (*data poisoning attack*) dapat memengaruhi perilaku model dengan cara yang sulit dideteksi melalui pengujian fungsional standar.

Selain itu, proses pembaruan melalui udara (*Over-the-Air* atau OTA) untuk model AI harus dikelola dengan ketat, setara dengan proses pembaruan perangkat lunak kendaraan lainnya. Ini mencakup verifikasi keaslian yang tepat, perlindungan terhadap pemulihan versi sebelumnya (*rollback protection*), serta penerapan bertahap yakni meluncurkan pembaruan ke sebagian kecil kendaraan terlebih dahulu sebelum diterapkan ke seluruh armada, selama hal tersebut dimungkinkan secara teknis.

9.7 EVALUASI KERENTANAN DAN STRATEGI PERTAHANAN SENSOR SERTA AI OTOMOTIF

Sensor dan sistem AI pada kendaraan memperluas cakupan kerentanan (*attack surface*) kendaraan, tidak hanya terbatas pada jaringan internal kendaraan, tetapi juga mencakup domain penginderaan fisik dan lapisan inferensi AI. Sebagaimana telah ditunjukkan dalam bab ini, kelemahan di ranah ini tidak hanya muncul akibat kesalahan implementasi, tetapi juga karena karakteristik fisik dan arsitektural yang mendasar: sensor yang tidak mampu mengautentikasi sinyal yang diterimanya, antarmuka yang tidak dilengkapi dengan perlindungan kriptografis, model persepsi yang rentan terhadap input yang bersifat adversarial (sengaja dimanipulasi untuk mengecoh), serta asisten AI yang tidak dapat membedakan instruksi dari data secara andal.

Bab ini mengkaji kelemahan-kelemahan tersebut pada berbagai teknologi sensor dan kecerdasan buatan (AI), serta memetakannya ke dalam teknik pengujian penetrasi (*penetration testing*) yang mencakup fase penemuan (*discovery*), pemeriksaan kerentanan, dan eksploitasi. Untuk memberikan gambaran yang lebih jelas mengenai penerapan teknik-teknik ini dalam praktik, bab ini juga menyajikan tiga studi kasus yang menunjukkan bagaimana kelemahan pada sensor dan AI dapat dieksploitasi untuk memengaruhi perilaku kendaraan atau mengompromikan sistem lain yang terkait. Praktik-praktik terbaik yang dibahas mencerminkan pendekatan pertahanan berlapis (*defense-in-depth*), di mana setiap input sensor dianggap tidak tepercaya sampai diverifikasi; autentikasi kriptografis diterapkan pada tingkat antarmuka jika memungkinkan; serta keamanan sensor dan AI diintegrasikan ke dalam siklus hidup pengembangan sejak tahap perancangan awal. Pemantauan pascaproduksi

melalui VSOC (*Vehicle Security Operations Center*) milik produsen kendaraan memberikan lapisan deteksi tambahan setelah kendaraan digunakan di lapangan. Pada bab berikutnya, kita akan meninjau lanskap ancaman yang terus berkembang bagi kendaraan terhubung (*connected vehicles*) dan kendaraan otonom, serta membahas bagaimana industri otomotif dapat terus memajukan pendekatan keamanan siber mereka dalam merespons tantangan-tantangan baru yang muncul.

BAB 10
TRANSFORMASI KEAMANAN SIBER TERKELOLA

Sepanjang buku ini, kita telah menelusuri berbagai metode serangan yang menyasar beragam komponen dan antarmuka kendaraan modern, serta praktik-praktik terbaik untuk memitigasinya. Namun, seiring dengan evolusi arsitektur kendaraan menuju *software-defined vehicle* (SDV) kendaraan yang berbasis perangkat lunak yang berpusat pada platform komputasi berkinerja tinggi (*high-performance computing* atau HPC), pendekatan keamanan yang bersifat *ad-hoc* yang sering disebut sebagai strategi "tambal lalu berharap" (*patch and pray*) tidak lagi dapat dipertahankan. Pola pikir reaktif semacam ini, di mana kerentanan hanya ditangani saat ancaman sudah di depan mata, tidak memberikan jaminan apa pun bahwa suatu sistem akan tetap tangguh menghadapi ancaman yang kian canggih.

Dalam bab ini, kita beralih dari ranah ofensif ke ranah defensif, dengan menekankan pada pendekatan keamanan terkelola yang sistematis. Inti dari transisi ini adalah sistem manajemen keamanan siber (*Cybersecurity Management System* atau CSMS), dengan turut mempertimbangkan peran Generative AI dalam mempercepat serta memperluas cakupan kepatuhan. Kita akan melampaui sekadar pemenuhan daftar periksa (*checkbox compliance*) untuk mendalami filosofi *secure-by-design* (keamanan sejak tahap perancangan). Filosofi ini mengintegrasikan aspek keamanan di sepanjang siklus hidup produk mulai dari perencanaan awal, pemeriksaan ancaman dan penilaian risiko (*Threat Analysis and Risk Assessment* atau TARA), hingga proses produksi yang aman serta pemantauan berkelanjutan setelah produk digunakan.

Lebih jauh lagi, kita harus menyadari bahwa kita sedang berlomba melawan pihak lawan yang terus berevolusi. Munculnya agen penyerang yang diperkuat oleh kecerdasan buatan (AI-*augmented offensive agents*) telah mengubah model biaya bagi penyerang secara mendasar; hambatan utama kini bergeser dari ketergantungan pada keahlian manusia menjadi batasan komputasi. Untuk memenangkan perlombaan ini, pihak pertahanan juga harus memanfaatkan Generative AI guna mengotomatisasi tugas-tugas keamanan, menutup "kesenjangan otomatisasi" (*automation gap*), serta berupaya untuk mengeliminasi kerentanan sepenuhnya, bukan sekadar memitigasinya. Dengan membangun sinergi antara wawasan ofensif dan ketangguhan defensif, kita dapat menciptakan masa depan yang resilien bagi generasi mobilitas berikutnya. Pada bagian selanjutnya dari bab ini, kita akan membahas topik-topik utama berikut:

- Pendekatan keamanan ad-hoc
- Pendekatan keamanan terkelola (*managed security*)
- Meningkatkan standar keamanan dalam sistem otomotif
- Memenangkan persaingan

10.1 KELEMAHAN DAN RISIKO PENDEKATAN AD-HOC DALAM KEAMANAN SIBER OTOMOTIF

Pendekatan ad-hoc terhadap keamanan siber lazim ditemukan pada organisasi yang menganggap keamanan sebagai beban yang hanya perlu ditangani saat krisis terjadi. Pada dasarnya, ini merupakan pola pikir reaktif sering disebut sebagai "tambal lalu berdoa" (*patch and pray*) di mana tim teknis terus-menerus berupaya memperbaiki kelemahan yang ditemukan, sembari berharap tidak ada kerentanan lain yang dieksploitasi. Metode ini tidak memberikan jaminan keamanan yang dapat dibuktikan dan gagal menjaga ketahanan sistem seiring dengan berkembangnya arsitektur kendaraan. Organisasi yang beroperasi dalam lingkup ini biasanya berada pada berbagai tingkat kematangan.

Pola Pikir yang Hanya Berorientasi Pada Kepatuhan (*compliance-only*)

Di satu sisi spektrum, terdapat organisasi yang tidak akan melakukan apa pun terkait keamanan siber otomotif kecuali jika didorong oleh persyaratan eksplisit dari pelanggan atau regulasi. Bahkan dalam kondisi tersebut, mereka hanya akan melakukan upaya minimal untuk memenuhi persyaratan, tanpa mempertimbangkan aspek lain yang pada akhirnya justru dapat menyebabkan pelanggaran terhadap persyaratan itu sendiri.

Sebagai contoh, sebuah organisasi mungkin memutuskan untuk tidak menerapkan penandatanganan kode (*code signing*) dan enkripsi pada binary perangkat lunak untuk semua pelanggan. Namun, jika pelanggan tertentu mewajibkan adanya penandatanganan kode dan enkripsi, mereka akan mengaktifkannya melalui opsi konfigurasi. Organisasi semacam ini menganggap penandatanganan kode dan enkripsi sekadar sebagai fitur yang dapat diaktifkan atau dinonaktifkan sesuai permintaan.

Akan tetapi, berbeda dengan fitur pada umumnya, mendistribusikan binary tanpa enkripsi dapat berdampak pada pihak lain dalam rantai pasok tanpa sepengetahuan mereka. Bayangkan sebuah skenario di mana pemasok mendistribusikan pustaka (*library*) umum, komponen *firmware*, atau modul perangkat lunak yang digunakan di berbagai ECU atau platform kendaraan. Jika penyerang memperoleh binary yang tidak dienkripsi dari konfigurasi satu OEM, mereka mungkin dapat melakukan rekayasa balik (*reverse engineering*) terhadap kode bersama yang juga terdapat dalam versi terlindungi yang dikirimkan ke OEM lain. Dalam kasus ini, binary yang tidak terlindungi tersebut menjadi peta jalan untuk memahami logika eksklusif, mengidentifikasi kerentanan, atau mengembangkan eksploitasi terhadap sistem yang sebenarnya telah menggunakan enkripsi pada binary-nya. Pendekatan *ad-hoc* tidak memandang keamanan siber sebagai aspek teknis menyeluruh yang memiliki implikasi melampaui satu produk atau pelanggan saja; inilah salah satu kelemahan mendasar dari pendekatan tersebut.

Kesesatan Berpikir "Hanya Mengandalkan PSIRT"

Di sisi lain spektrum, terdapat organisasi yang memiliki program keamanan siber yang hanya menangani risiko saat risiko tersebut menjadi ancaman nyata atau terbukti melalui eksploitasi aktif. Organisasi semacam ini menginvestasikan sumber daya pada program *Product Security and Incident Response* (PSIRT) yang secara aktif mencari laporan kerentanan serta memastikan kerentanan tersebut dipilah dan ditangani. Meskipun memiliki program PSIRT

yang matang untuk mendeteksi dan menambal kerentanan saat ditemukan adalah hal yang baik, memilih untuk bertindak hanya ketika insiden siber terjadi merupakan bentuk kelalaian yang ceroboh.

Berbeda dengan sistem TI di mana kegiatan penambalan (*patching*) mingguan atau bahkan harian dianggap sebagai prosedur bisnis yang wajar keharusan untuk terus-menerus menambal sistem otomotif bukanlah hal yang praktis maupun etis. Pembaruan perangkat lunak memerlukan proses pengujian yang ketat untuk memastikan bahwa perbaikan keamanan tidak menimbulkan masalah kualitas atau keselamatan yang tidak terduga. Proses ini membuat tahapan penambalan menjadi lebih lama dibandingkan pada sistem TI pada umumnya. Beberapa produsen kendaraan (OEM) memiliki keterbatasan yang membuat mereka tidak mungkin merilis pembaruan perangkat lunak lebih cepat dari dua bulan (atau bahkan lebih) sejak masalah dilaporkan.

Selain itu, karena penyaluran perangkat lunak bergantung pada proses *over-the-air* (OTA), ada kemungkinan pelanggan tidak menerima pembaruan pada kendaraan mereka jika mereka terus menunda pembaruan tersebut atau berada di luar jangkauan layanan. Terlebih lagi, jika terjadi serangan yang mengakibatkan hilangnya nyawa atau cedera, proses PSIRT tidak akan pernah bisa memulihkan kondisi manusia yang terdampak. Oleh karena itu, jelas bahwa memusatkan perhatian hanya pada respons terhadap insiden tidak jauh lebih baik daripada pendekatan di ujung spektrum lainnya, di mana upaya keamanan siber yang dilakukan berada pada tingkat paling minimal.

Mengapa Pendekatan Ad-Hoc Gagal di Era Modern

Sebagaimana dibahas dalam buku ini, pendekatan ad-hoc pada dasarnya cacat karena beberapa faktor krusial:

- **Meluasnya Permukaan Serangan (*attack surfaces*)**: Peralihan dari unit kontrol elektronik (ECU) yang terisolasi secara fisik menuju kendaraan berbasis perangkat lunak (*Software-Defined Vehicles*—SDV) dan platform komputasi berkinerja tinggi (*High-Performance Computing*—HPC) telah meruntuhkan batasan perangkat keras tradisional. Pendekatan ad-hoc sering kali mengandalkan keamanan melalui kerahasiaan (*security* through *obscurity*) sebuah mitos yang telah dipatahkan oleh kemudahan ekstraksi *firmware* dan rekayasa balik (*reverse engineering*) yang terlihat dalam berbagai eksploitasi modern.
- **Kelemahan Dalam Kepercayaan Implisit (*implicit trust*)**: Keamanan yang bersifat ad-hoc tidak mampu memperhitungkan klaim kepercayaan implisit di dalam sistem karena tidak menerapkan pendekatan pemodelan ancaman (*threat modeling*) yang ketat. Hal ini mau tidak mau berujung pada ancaman yang tidak tertangani, dengan berpegang pada anggapan bahwa "ketidaktahuan adalah kebahagiaan" (*ignorance is bliss*). Salah satu contohnya, sebagaimana terlihat pada kasus Chauícu T, adalah kurangnya perhatian terhadap serangan periferal yang menyalahgunakan mesin DMA (*Direct Memory Access*) melalui USB. Karena mesin DMA secara fisik merupakan bagian dari sistem, muncul asumsi keliru bahwa komponen tersebut pada dasarnya "aman" dan tidak memerlukan pengawasan ketat. Akibatnya, sistem menjadi rentan terhadap

serangan yang memanfaatkan konfigurasi SMMU (*System Memory Management Unit*) yang longgar untuk menulis data ke wilayah memori di luar jangkauan yang seharusnya diakses oleh pengontrol tersebut.

- **Titik Balik AI**: Mungkin ancaman paling signifikan terhadap model pengembangan yang bersifat ad-hoc (tidak terencana secara sistematis) adalah runtuhnya model biaya bagi penyerang akibat kehadiran AI Generatif. Ketika pengembangan eksploit tidak lagi bergantung pada keahlian khusus, melainkan pada kapasitas komputasi, rentang waktu antara munculnya kerentanan hingga tersedianya eksploit yang siap digunakan untuk menyerang akan menyusut drastis dari hitungan bulan menjadi hitungan hari. Siklus penanganan kerentanan yang bersifat manual dan reaktif tidak akan mampu mengimbangi sistem serangan berbasis agen AI yang mampu memeriksa seluruh arsitektur untuk menemukan rangkaian eksploit seperti yang akan kita bahas lebih lanjut dalam bab ini.
- **Kerentanan Rantai Pasok**: Organisasi yang menggunakan pendekatan ad-hoc sering kali tidak memiliki daftar komponen (*Bill of Materials* atau BOM) yang komprehensif untuk perangkat lunak maupun kekayaan intelektual (IP) perangkat keras yang menjadi bagian dari produk akhir. Ini mencakup pustaka pihak ketiga, komponen sumber terbuka (*open-source*), modul perangkat lunak komersial, IP semikonduktor, *firmware*, serta berbagai komponen perangkat keras yang digunakan dalam kendaraan. Tanpa BOM yang andal, organisasi mungkin tidak menyadari adanya kerentanan bawaan dalam komponen-komponen yang mereka sertakan dalam produk tersebut.

Risiko lain yang terpisah namun saling berkaitan muncul dari lingkungan pengembangan itu sendiri. Penyerang dapat menargetkan kompilator, rantai alat pengembangan (*build toolchain*), SDK, sistem penandatanganan, alur CI/CD, repositori paket, atau stasiun kerja pengembang untuk menyusupkan ancaman ke dalam perangkat lunak sebelum perangkat lunak tersebut sampai ke kendaraan. Dalam skenario ini, elemen yang rentan mungkin bukan bagian dari produk akhir yang dikirimkan, namun tetap dapat merusak produk selama tahap pengembangan atau perilisan. Kedua jenis risiko ini menuntut adanya visibilitas dan kendali. Jika suatu organisasi tidak mampu melacak apa yang mereka kirimkan dan tidak dapat melindungi perangkat yang digunakan untuk membangun produk tersebut, maka proses penambalan (*patching*) yang kuat sekalipun hanya akan bersifat reaktif dan tidak menyeluruh.

Indikasi Budaya Kerja yang Bersifat Ad-Hoc

Sebuah organisasi kemungkinan besar beroperasi dalam lingkup yang bersifat ad-hoc jika mereka meyakini salah satu dari kesalahpahaman berbahaya berikut:

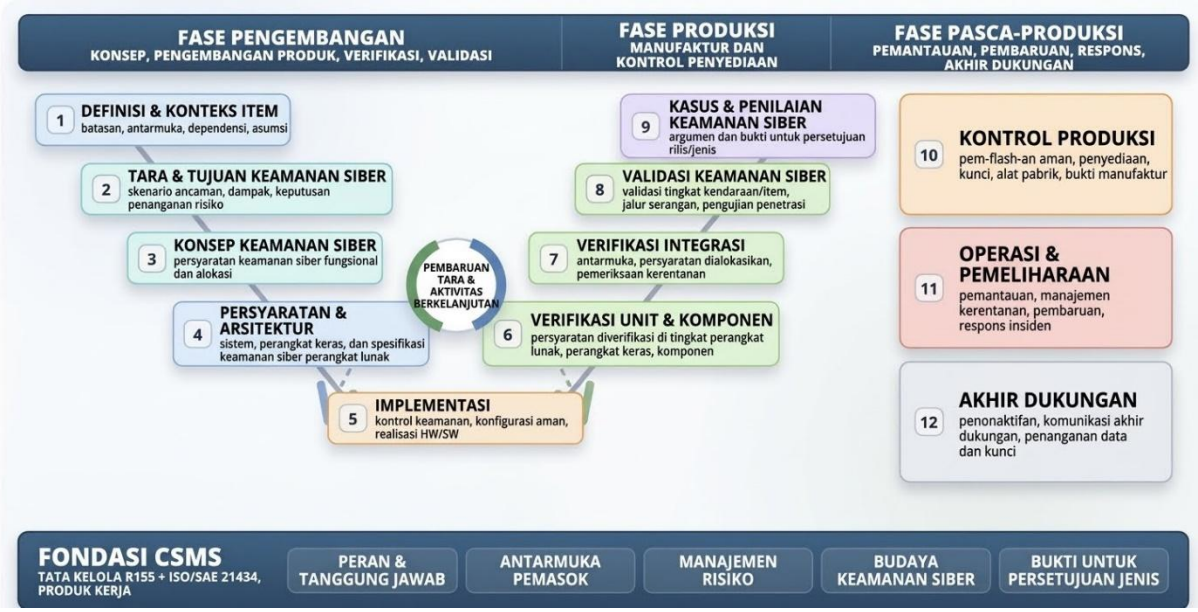
- Risiko keamanan hanya dipertimbangkan tepat sebelum peluncuran produk atau setelah terjadi insiden peretasan.
- Keamanan siber dianggap sebagai tanggung jawab satu "orang yang ditunjuk" saja, bukan tanggung jawab seluruh tim rekayasa.
- Pengujian penetrasi (*penetration test*), baik yang dilakukan oleh pihak ketiga maupun internal, dianggap sebagai bukti keamanan yang sudah memadai.

- Keberadaan satu fitur perangkat keras, seperti HSM (*Hardware Security Module*) tertanam, secara otomatis membuat ECU menjadi aman.
- Penerapan proses keamanan siber dianggap mahal dan memakan waktu, ditambah lagi karena pelanggan tidak memintanya, maka tanggung jawab keamanan organisasi dianggap cukup seminimal mungkin.

Pada dasarnya, pendekatan *ad-hoc* mengasumsikan bahwa kemampuan penyerang untuk meningkatkan skala aksinya lebih lambat dibandingkan kemampuan pihak pertahanan. Di era yang ditandai dengan adanya alur penemuan kerentanan secara otonom dan perancangan eksploitasi yang berlangsung cepat, asumsi ini keliru. Akibatnya, kendaraan beserta penumpangnya menjadi rentan hingga ke tingkat yang tidak dapat diterima. Setelah mempelajari pendekatan *ad-hoc*, kini kita akan beralih membahas pendekatan keamanan terkelola (*managed security approach*) yang dirancang untuk mengatasi berbagai kelemahan tersebut.

10.2 PENERAPAN CSMS SEPANJANG SIKLUS HIDUP KENDARAAN

Inti dari pendekatan ini adalah sistem manajemen keamanan siber (*Cybersecurity Management System* atau CSMS), yang mengatur aspek keamanan siber di sepanjang siklus hidup rekayasa produk, sebagaimana ditunjukkan pada Gambar 10.1.



Gambar 10.1 Penerapan sistem manajemen keamanan siber pada seluruh siklus hidup pengembangan

Pendekatan keamanan terkelola bukan sekadar menambahkan fitur perangkat lunak yang aman seperti enkripsi, *secure boot*, atau kontrol akses. Regulasi seperti UNECE R155 dan standar GB 44495-2024 dari Tiongkok mewajibkan produsen untuk menetapkan serta menunjukkan kerangka kerja siklus hidup dalam mengelola risiko keamanan siber. Hal ini mencakup penerapan tata kelola terhadap penilaian risiko, persyaratan, perancangan,

implementasi, verifikasi, produksi, operasional, penanganan insiden, pembaruan sistem, hingga koordinasi dengan pemasok.

Karena para insinyur otomotif telah familier dengan model V-cycle, CSMS dapat dipandang sebagai lapisan tata kelola keamanan siber yang mencakup setiap fase pengembangan. ISO/SAE 21434 hadir dalam konteks ini sebagai standar rekayasa yang menetapkan bagaimana aktivitas keamanan siber harus dilaksanakan di sepanjang siklus hidup kendaraan. Dalam praktiknya, organisasi menggunakan ISO/SAE 21434 untuk menyusun proses, produk kerja, dan bukti yang diperlukan guna mendukung kepatuhan terhadap CSMS di bawah kerangka regulasi seperti UNECE R155 dan GB 44495-2024.

Sebelum membahas CSMS lebih dalam, kita perlu menelaah peran sistem manajemen rekayasa yang sudah ada sistem yang mendukung siklus hidup produk serta mendokumentasikan bukti pelaksanaan pekerjaan rekayasa yang disiplin. Sistem-sistem tersebut seperti manajemen kebutuhan (*requirements management*), manajemen konfigurasi, manajemen perubahan, pelacakan masalah (*issue tracking*), dan manajemen verifikasi bukanlah sistem yang khusus ditujukan untuk keamanan siber, dan sebaiknya tidak dibangun sebagai entitas terpisah yang berdiri sendiri (*silo*). Sebaliknya, sistem-sistem ini harus diperluas agar kebutuhan, hasil kerja, keputusan, dan bukti terkait keamanan siber dapat dikelola dengan tingkat ketelitian yang sama seperti halnya aspek keselamatan, kualitas, dan kebutuhan fungsional lainnya. Jika sistem siklus hidup ini tidak ada atau lemah, maka keamanan siber akan sulit untuk diperluas skalanya, sulit untuk diaudit, dan rentan untuk dilewati (diabaikan). Secara minimal, pendekatan yang terkelola biasanya mencakup hal-hal berikut:

- **Manajemen Kebutuhan**: Sistem manajemen kebutuhan memungkinkan pendefinisian, penetapan tolok ukur (*baselining*), dan penelusuran (*traceability*), baik terhadap kebutuhan pelanggan maupun kebutuhan produk/komponen. Dalam CSMS, penelusuran bukanlah sekadar formalitas administratif, melainkan mekanisme yang memungkinkan penilai (*assessor*) untuk melacak alur mulai dari keputusan penanganan risiko hingga pengendalian yang diimplementasikan beserta bukti verifikasinya.
- **Manajemen Konfigurasi**: Keamanan siber sangat bergantung pada integritas kode maupun hasil kerja (*work products*). Manajemen konfigurasi memastikan bahwa perangkat lunak, data kalibrasi/konfigurasi, serta dokumentasi pendukung memiliki kontrol versi, telah ditinjau, dan dapat ditelusuri kembali ke tolok ukur rekayasa (*engineering baselines*). Tanpa disiplin ini, organisasi tidak dapat menjawab pertanyaan jaminan dasar secara andal: "Artefak spesifik apa saja yang telah dinilai dan dirilis?"
- **Manajemen Dokumentasi**: Standar keamanan siber otomotif berbasis pada bukti (*evidence-driven*). CSMS bergantung pada hasil kerja yang terkontrol, seperti catatan TARA (*Threat Analysis and Risk Assessment*), kebutuhan keamanan siber, perjanjian antarmuka (*interface agreements*), hasil verifikasi, dan argumen kesiapan rilis. Jika dokumentasi tidak dikelola sebagai artefak rekayasa, program tersebut tidak akan

mampu mempertahankan dasar pemikiran keamanan (*security rationale*) seiring berjalannya waktu.

- **Manajemen Kualitas**: Dalam konteks keamanan siber, sistem manajemen kualitas merupakan prasyarat, bukan sekadar tambahan opsional. Hal ini sangat penting dalam rantai pasok otomotif modern, di mana pelaku baru yang masuk ke industri tidak selalu memiliki tingkat kematangan proses yang setara dengan pemasok otomotif yang sudah mapan. Manajemen kualitas yang lemah tidak hanya menghasilkan produk dengan kualitas lebih rendah, tetapi juga menyebabkan persyaratan tidak dipahami dengan baik, maksud desain tidak tercermin sepenuhnya dalam implementasi, serta cacat sistem yang lolos ke lingkungan produksi terutama dalam kondisi ekstrem atau skenario serangan. Akibatnya, celah bagi penyerang untuk menemukan kerentanan menjadi lebih luas. Cacat dalam penanganan *buffer*, ketidakkonsistenan dalam *state machine*, atau fitur yang berperilaku tidak sesuai rencana saat menerima input yang tidak valid semuanya bermula sebagai "masalah kualitas" sebelum akhirnya berkembang menjadi kerentanan keamanan. Oleh karena itu, penerapan standar kualitas dasar sangatlah penting jika sebuah organisasi ingin membangun sistem yang aman.
- **Manajemen Perangkat (*tool management*)**: Perangkat merupakan bagian dari siklus hidup pengembangan produk, sehingga juga menjadi bagian dari ruang lingkup serangan. Jika rantai perangkat (*toolchain*) berhasil disusupi, penyerang tidak perlu meretas ECU secara langsung; mereka cukup memengaruhi apa yang dibangun, ditandatangani, dan didistribusikan. Dengan demikian, pendekatan keamanan yang terkelola harus memandang pemilihan perangkat, pembaruan perangkat, kontrol akses terhadap perangkat, dan asal-usul perangkat sebagai keputusan teknis yang berkaitan dengan keamanan.

Di antara berbagai sistem manajemen terkait lainnya, kita akan memprioritaskan pembahasan mengenai manajemen perangkat karena relevansinya yang kuat, baik dalam aspek keamanan defensif maupun ofensif.

Realitas Pengelolaan Perangkat (*tool*) dan Rantai Pasok

Integritas perangkat menjadi krusial karena pengembangan otomotif modern merupakan ekosistem yang terdiri dari sistem pembangunan (*build systems*), alur CI/CD, kompilator, pengelola dependensi, repositori artefak, infrastruktur penandatanganan kode, dan lingkungan pengujian. Perangkat atau vendor yang disusupi dapat memungkinkan penyisipan kode berbahaya atau manipulasi selama proses pembangunan tindakan yang sulit dideteksi pada tahap selanjutnya, terutama ketika artefak yang dihasilkan telah ditandatangani dan didistribusikan secara sah. Inilah sebabnya mengapa serangan rantai pasok bukanlah kasus yang jarang terjadi (*edge case*).

Seiring dengan semakin matangnya proses pengembangan yang aman dan meningkatnya ketahanan produk terhadap eksploitasi langsung, pihak lawan memiliki motivasi kuat untuk menargetkan vendor dalam rantai pasok, infrastruktur pembangunan, serta perangkat pendukung. Dalam skenario ini, satu penyusupan saja dapat berdampak luas pada berbagai produk dan varian. Bagi kelompok ancaman persisten tingkat lanjut (*Advanced*

Persistent Threat/APT) maupun aktor negara, model ini sangat menarik karena mampu melewati antarmuka produk yang telah diperkuat, dengan cara menyusup ke dalam alur rekayasa tempat perangkat lunak tepercaya diproduksi.

Insiden SolarWinds sering dijadikan rujukan karena mengungkap fakta yang tidak menyenangkan: organisasi yang canggih sekalipun dapat disusupi melalui saluran rantai pasok yang tepercaya, terutama jika perlindungan pada rantai perangkat (*toolchain*) dan pengendalian asal-usul (*provenance*) tidak memadai. Pelajarannya adalah bahwa kepercayaan terhadap vendor dan perangkat harus dikelola melalui proses yang ketat dan divalidasi secara berkelanjutan. Tidak cukup hanya memercayai hasil keluaran dari vendor; organisasi juga harus mengevaluasi praktik keamanan vendor tersebut, kebiasaan mereka dalam menangani pembaruan perangkat lunak, serta kemampuan mereka dalam menjaga integritas proses pembangunan dari waktu ke waktu.

Aspek lain yang sering terabaikan adalah banyaknya perangkat yang bersentuhan dengan sistem target sebelum sistem tersebut dirilis ke tahap produksi. Dalam konteks ini, perangkat sisi target (*target-side tools*) merujuk pada utilitas atau komponen perangkat lunak yang dijalankan pada ECU, HPC, atau perangkat tertanam (*embedded device*) selama fase pengembangan, manufaktur, pemuatan perangkat lunak (*flashing*), kalibrasi, pengujian, atau servis namun tidak dimaksudkan untuk tetap ada dalam sistem produksi akhir. Contohnya mencakup utilitas untuk memuat perangkat lunak (*flashing utilities*), perangkat lunak pengujian pabrik, alat bantu kalibrasi, perangkat pembuka akses diagnostik, skrip penyediaan (*provisioning scripts*), serta aplikasi layanan sementara.

Karena perangkat-perangkat ini dijalankan pada sistem target, penggunaannya harus dikendalikan dengan cermat. Jika perangkat tersebut tertinggal secara tidak sengaja, terpapar dalam citra (*image*) produksi, atau dapat digunakan tanpa otorisasi yang tepat, perangkat tersebut dapat menjadi vektor serangan terhadap produk akhir. Sebagai contoh, pertimbangkan perangkat pengujian khusus pabrik yang dimuat ke dalam sistem target sebelum kendaraan meninggalkan pabrik. Jika perangkat tersebut tidak ditandatangani secara digital, tidak memiliki kendali akses, dan tidak terikat secara kriptografis pada perangkat atau tahap produksi tertentu, siapa pun yang memperoleh salinannya dapat menggunakannya kembali pada sistem produksi untuk memfasilitasi rekayasa balik (*reverse engineering*), memintas kendali normal, atau mengubah status perangkat.

Oleh karena itu, pendekatan yang terkelola menuntut penerapan praktik pengelolaan perangkat yang setidaknya mencakup kebutuhan teknis berikut, baik untuk perangkat di lingkungan rekayasa maupun perangkat yang berada di dalam sistem target. Perlu dicatat bahwa perangkat di lingkungan rekayasa beroperasi di luar sistem target misalnya pada stasiun kerja pengembang, server pengembangan (*build server*), infrastruktur CI/CD, stasiun manufaktur, atau laptop layanan:

- **Integritas dan Keterlacakan Pengembangan (*build*)**: Organisasi harus mampu membuktikan apa yang dibangun, dari sumber mana, melalui alur (*pipeline*) apa, menggunakan versi perangkat yang mana, serta berdasarkan persetujuan siapa.

- **Kendali Akses dan Pemisahan Tugas**: Perangkat dan sistem pengembangan tidak boleh menjadi infrastruktur bersama dengan hak akses tinggi, di mana kompromi pada satu akun dapat menyebabkan kompromi pada artefak produksi. Akses pengembang, administrasi repositori kode sumber, wewenang penandatanganan, dan persetujuan rilis tidak boleh berada dalam batasan kepercayaan (*trust boundary*) yang sama.
- **Kendali Dependensi**: Dependensi pihak ketiga dan pembaruan perangkat harus diperlakukan sebagai perubahan yang terkendali, bukan sebagai perubahan yang tidak terpantau.
- **Pemeriksaan Integritas Kode Dan Kemampuan Audit**: Pengembangan yang ditandatangani secara digital, verifikasi integritas artefak yang dihasilkan, audit berkala terhadap lingkungan pengembangan, serta pemeriksaan statis terhadap perubahan dan keluaran, semuanya membantu mendeteksi modifikasi yang tidak wajar sebelum produk siap dirilis.
- **Manajemen Risiko Vendor**: Pemasok harus dievaluasi tidak hanya berdasarkan fungsionalitas dan biaya, tetapi juga berdasarkan tingkat kematangan praktik keamanan mereka. Memercayai vendor perangkat tanpa evaluasi berkelanjutan pada akhirnya akan menjadi risiko yang tidak terkelola.
- **Pengikatan Perangkat Ke Sistem Target**: Penandatanganan kode (*code signing*) tidak boleh terbatas hanya pada perangkat lunak produksi saja. Setiap perangkat (*tools*) di sisi target yang menawarkan kapabilitas yang dapat dimanfaatkan oleh penyerang selama salah satu tahapan siklus hidup perangkat harus ditandatangani secara kriptografis serta menyertakan pengidentifikasi unik perangkat, guna memastikan bahwa perangkat tersebut hanya dapat digunakan pada target spesifik yang memang dirancang untuknya.
- **Pelestarian Bukti**: Saat terjadi suatu insiden, organisasi harus mampu merekonstruksi kondisi rantai perangkat (*toolchains*), dependensi, persetujuan, dan artefak yang dihasilkan.

Pengadaan dan Manajemen Pemasok

Rantai pasok otomotif sangat luas dan mendalam. Baik saat membuat satu cip maupun ECU berkinerja tinggi, produk biasanya menyertakan komponen dan kekayaan intelektual (IP) dari berbagai pemasok. Dalam pendekatan yang terkelola, suatu sistem tidak dapat dianggap aman jika status keamanan komponen yang dipasok tidak diketahui atau tidak dikelola. Hal ini memunculkan dua persyaratan yang sering kali diremehkan:

- **Pendataan komponen yang dipasok**: program harus mampu memberikan visibilitas mengenai apa saja yang dipasok, bagaimana komponen tersebut diintegrasikan, serta asumsi keamanan apa yang diwariskan.
- **Penetapan pembagian tanggung jawab**: tanggung jawab keamanan harus dialokasikan secara eksplisit kepada pihak-pihak terkait.

Mekanisme praktis untuk alokasi ini adalah perjanjian antarmuka keamanan siber (*cybersecurity interface agreement*), yang memperjelas apa yang disediakan oleh masing-masing pihak dan apa yang diasumsikan oleh masing-masing pihak. Selama proses pengadaan,

pemasok juga harus dievaluasi terkait kapabilitas keamanan siber serta ekspektasi keamanannya, baik dari sisi proses maupun teknis. Hal ini harus dikomunikasikan sejak awal termasuk dalam dokumen permintaan penawaran (RFQ) jika relevan guna menghindari pemilihan pemasok yang hanya didasarkan pada kecocokan fitur, biaya, atau komitmen pengiriman. Dengan mendata komponen yang dipasok, menetapkan tanggung jawab, dan memverifikasi ekspektasi keamanan sejak dini, pihak lawan akan memiliki lebih sedikit celah seperti ketergantungan yang tidak diketahui, kerentanan bawaan, atau kesenjangan integrasi yang dapat mereka eksploitasi.

Perencanaan

Perencanaan kegiatan keamanan siber merupakan langkah awal dalam menetapkan ekspektasi mengenai peran Sistem Manajemen Keamanan Siber (CSMS) bagi suatu produk. Rencana keamanan siber berfungsi sebagai peta jalan yang memandu pelaksanaan; rencana ini mendefinisikan berbagai kegiatan, keputusan penyesuaian (*tailoring*), strategi penanganan pemasok dan perangkat, perlakuan terhadap komponen yang sudah tersedia di pasaran (*off-the-shelf*), serta pendekatan pengujian keamanan. Rencana ini juga menjadi acuan dalam penilaian. Saat produk mencapai tahapan kesiapan tertentu (*readiness gates*), rencana tersebut memungkinkan organisasi untuk menjawab dua pertanyaan krusial:

- Kegiatan keamanan apa saja yang telah disepakati untuk dilaksanakan?
- Apakah kegiatan-kegiatan tersebut telah dilaksanakan dengan kelengkapan dan kualitas yang memadai?

Dalam konteks ini, penyelarasan menjadi penting karena kegagalan keamanan siber biasanya berakar pada kegagalan proses seperti terlewatnya aktivitas keamanan siber, penetapan peran dan tanggung jawab yang tidak tepat, kurangnya perhatian terhadap pemantauan sejawat (*peer review*), tidak adanya atau tidak lengkapnya bukti pelaksanaan pekerjaan, hingga terlambatnya deteksi risiko berdampak tinggi tanpa perencanaan yang memadai untuk pengelolaan kerentanan dan penambalan (*patch management*).

Integrasi Dengan Siklus Hidup Rekayasa

Mari kita cermati bagaimana CSMS (*Cybersecurity Management System*) terintegrasi dengan keseluruhan siklus hidup rekayasa, sebagaimana diilustrasikan dalam Gambar 10.1, mulai dari TARA (*Threat Analysis and Risk Assessment*) pada fase konsep, berlanjut melalui fase pemeliharaan, hingga akhir masa pakai dan penonaktifan (*decommissioning*).

TARA

Salah satu ciri utama dari pendekatan terkelola ini adalah peran pemodelan ancaman (*threat modeling*) dalam mengendalikan risiko keamanan siber saat fitur-fitur baru diperkenalkan. Alih-alih menunggu terjadinya insiden atau temuan pengujian, TARA memastikan bahwa risiko diidentifikasi, diberi skor, ditangani, dan didokumentasikan sebelum keputusan desain menjadi terlalu mahal untuk diubah. Para insinyur sering kali menyampaikan kekhawatiran yang beralasan: pemodelan ancaman bisa memakan waktu, dan hasilnya dapat bervariasi tergantung pada tim serta metode yang digunakan. Masalah-masalah ini nyata, namun tidak meniadakan kebutuhan akan proses tersebut. Pemodelan ancaman tetap menjadi mekanisme paling sistematis untuk menunjukkan bahwa risiko keamanan telah

dipertimbangkan dan ditangani dengan bukti yang dapat dilacak. Tentu saja, banyak optimalisasi yang dapat dilakukan untuk meningkatkan konsistensi dan efisiensi, tetapi beroperasi tanpa TARA sama saja dengan memilih untuk mengelola risiko hanya berdasarkan upaya terbaik (*best effort*) semata. Tanpa penerapan sistematis dari pemeriksaan ancaman dan pendekatan manajemen risiko, mustahil untuk membuktikan kelengkapan serta ketepatan dari kegiatan manajemen risiko teknis tersebut.

Dalam praktiknya, TARA dapat diterapkan menggunakan berbagai metode. Pendekatan berbasis aset (*asset-driven approach*), sebagaimana ditetapkan dalam standar ISO/SAE 21434, digunakan secara luas, sementara pendekatan berbasis interaksi diterapkan secara selektif ketika interaksi sistem dan batasan kepercayaan (*trust boundaries*) menjadi faktor risiko utama. Keunggulan operasional dari metode berbasis aset adalah fitur-fitur baru dapat diperiksa dengan memetakannya ke aset dan properti keamanannya, sehingga memungkinkan penggunaan kembali ancaman dan kontrol yang sudah ada jika relevan.

Sebagai contoh, pertimbangkan penambahan fitur untuk merekam kejadian ke dalam perangkat perekam data elektronik (*Electronic Data Recorder*/EDR). Jika data yang direkam mencakup cuplikan kamera yang sebelumnya telah diidentifikasi sebagai aset yang memerlukan perlindungan kerahasiaan, maka kontrol kerahasiaan tersebut tetap relevan dan dapat dilacak. Namun, fitur ini mungkin menghadirkan properti tambahan yang juga memerlukan perlindungan, seperti *non-repudiation* (anti-penyangkalan) demi alasan hukum atau akuntabilitas. Hal ini mengubah penanganan risiko: program tersebut harus mengurangi risiko pemalsuan cuplikan kamera oleh penyerang yang berhasil mengakses jalur data atau penyimpanan EDR.

Inilah nilai praktis dari TARA. Metode ini mewajibkan pemeriksaan eksplisit terhadap properti aset yang terdampak, mencatat keputusan penanganan risiko, serta menghasilkan bukti yang dapat ditinjau oleh penilai independen. Dalam pendekatan yang bersifat *ad-hoc* (tidak terstruktur), risiko penyangkalan (*repudiation*) mungkin terlewatkan sepenuhnya atau ditangani secara informal tanpa bukti yang kuat, sehingga program tersebut tidak mampu memberikan justifikasi atas postur keamanannya. Manfaat lain dari TARA adalah tersedianya mekanisme terstruktur untuk memasukkan ancaman-ancaman baru yang ditemukan melalui insiden. Hal ini menjembatani siklus antara penemuan serangan (*offensive discovery*) dan rekayasa pertahanan (*defensive engineering*). Seiring dengan berkembangnya teknik serangan dan pemahaman yang lebih baik mengenai tingkat kelayakan serangan, pembaruan dalam TARA akan meningkatkan pemodelan jalur serangan serta penilaian risiko, sekaligus mendorong penerapan kontrol yang lebih tepat sasaran pada generasi produk berikutnya. Jika TARA tidak digunakan, program tersebut hanya dihadapkan pada pilihan biner: melakukan perbaikan (*patch*) atau tidak melakukan apa pun. Sering kali, tidak ada cara yang koheren untuk mengevaluasi dampak, menjustifikasi keputusan untuk menerima risiko (*risk retention*), atau mendokumentasikan alasan mengapa suatu kerentanan tidak dapat dieksploitasi secara praktis dalam desain sistem tertentu.

Dengan demikian, TARA menyediakan lapisan bukti yang selama ini kurang. Metode ini juga memungkinkan program untuk menjustifikasi risiko yang diterima (*retained risk*) jika

memang dianggap tepat. Sebagai contoh, masalah *cache-timing* mungkin dinilai tidak dapat dieksploitasi secara praktis dalam desain *single-tenant* yang tidak menyimpan data rahasia di dalam cache bersama, karena operasi kriptografi sensitif didelegasikan ke lingkungan keamanan yang terlindungi oleh perangkat keras (*hardware-protected security environment*). Tanpa TARA, kesimpulan tersebut hanyalah sekadar opini. Namun dengan TARA, kesimpulan itu menjadi argumen teknis yang dapat dievaluasi.

Kebutuhan Akan Konsep Keamanan Siber

TARA yang menyeluruh akan menghasilkan konsep keamanan siber berkualitas tinggi yang menjabarkan bagaimana risiko dikurangi melalui kontrol keamanan teknis. Konsep keamanan siber ini menyajikan narasi keamanan yang menjelaskan bagaimana tujuan keamanan dicapai dan mengapa kontrol yang dipilih dianggap memadai untuk menghadapi risiko yang telah diidentifikasi. Untuk perangkat sensor cerdas yang memproses frame kamera, konsep keamanan siber mungkin berfokus pada sejumlah kontrol tertentu, seperti manajemen rantai pasok, penyediaan identitas dan sertifikat, serta protokol autentikasi sensor.

Dengan menjabarkan asumsi konteks sistem, lingkungan operasional, dan kasus penggunaan (*use cases*) yang dipenuhi oleh produk, konsep ini berfungsi sebagai cetak biru yang mencakup ancaman-ancaman yang dipertimbangkan selama pemeriksaan, tujuan keamanan yang harus dipenuhi oleh produk, dan yang terpenting, kontrol serta persyaratan keamanan yang diperlukan untuk mencapai tujuan tersebut guna menekan risiko teknis secara keseluruhan ke tingkat yang dapat diterima. Nilai dari konsep ini terletak pada kemampuannya untuk mengaitkan pengendalian-pengendalian tersebut secara langsung dengan risiko dan aset yang telah diidentifikasi dalam TARA, sehingga menunjukkan kelengkapan dalam penanganan risiko. Semakin kaya fitur suatu produk, semakin rumit pula konsep keamanannya. Sebagai contoh, dalam platform HPC (*High-Performance Computing*), konsep keamanan siber biasanya mencakup beberapa kendali dasar, seperti:

- *Secure boot* (proses booting yang aman) dan inisialisasi
- Partisi dan isolasi yang aman
- Pengerasan sistem operasi (OS *hardening*) dengan manajemen identitas dan akses (*Identity and Access Management* – IAM)
- Komunikasi yang aman
- Manajemen pembaruan yang aman
- Layanan kriptografi dan manajemen kunci

Setiap kelompok kendali berkaitan dengan serangkaian ancaman, sekaligus diturunkan menjadi persyaratan keamanan teknis yang harus dipenuhi oleh produk untuk mengatasi ancaman tersebut. Singkatnya, konsep keamanan siber menyediakan peta yang koheren dari risiko menuju strategi pengendalian, serta menjadi landasan bagi penyusunan spesifikasi keamanan siber selanjutnya.

Perancangan dan Implementasi

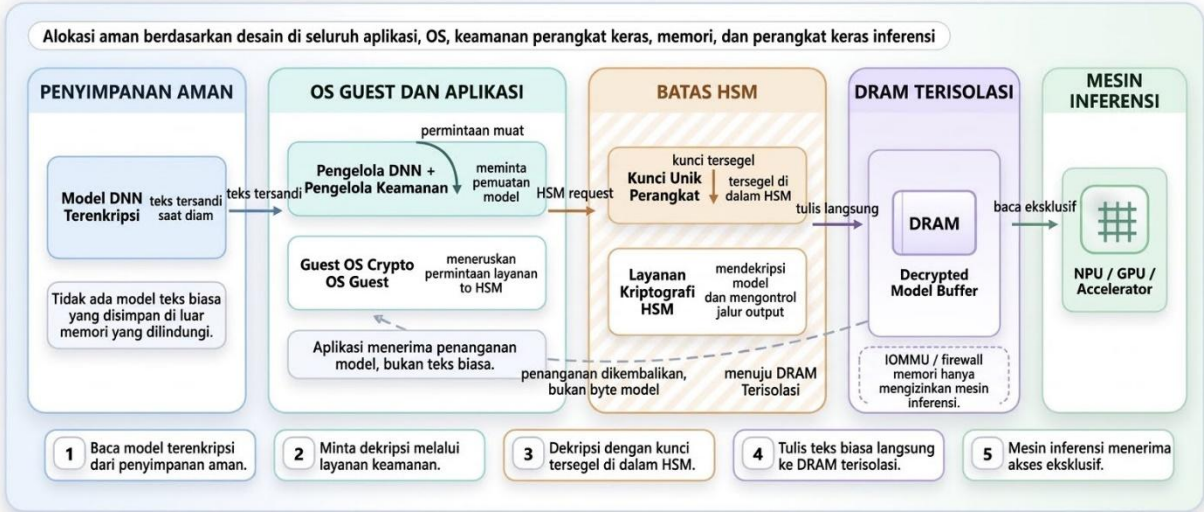
Produk yang dikembangkan di bawah kerangka kerja manajemen keamanan siber (*Cybersecurity Management System* – CSMS) diharapkan mengikuti prinsip *secure by design* (keamanan sejak tahap perancangan). Alih-alih menambahkan kendali pada produk yang

sudah tidak aman setelah produk tersebut selesai dibuat, prinsip *secure by design* menerapkan kendali secara terencana selama pengembangan, seiring dengan ditemukannya risiko dan ditetapkannya arsitektur sistem.

Setelah konsep keamanan siber ditetapkan, spesifikasi keamanan siber diturunkan untuk memenuhi konsep tersebut. Proses ini mencakup penerapan kendali di berbagai lapisan arsitektur serta perumusan persyaratan keamanan siber untuk setiap komponen. Persyaratan ini kemudian harus tercermin dalam artefak arsitektur perangkat lunak maupun perangkat keras, sehingga implementasinya tetap selaras dengan tujuan perancangan awal. Sebagai contoh, konsep keamanan siber mungkin menetapkan bahwa model *Deep Neural Network* (DNN) harus dienkripsi saat disimpan (*at rest*) dengan menggunakan kunci yang unik untuk setiap perangkat serta algoritma enkripsi yang telah disetujui. Untuk memenuhi persyaratan ini, diperlukan arsitektur yang memisahkan materi kunci dari logika aplikasi, serta mencegah data model dalam bentuk teks biasa (*plaintext*) terekspos ke luar batasan yang tepercaya. Persyaratan keamanan tersebut dapat diimplementasikan melalui arsitektur yang mengintegrasikan manajemen kunci berbasis perangkat keras, layanan kriptografi di dalam *Hardware Security Module* (HSM), penyimpanan aman, memori terisolasi, serta akses terkendali oleh mesin perangkat keras untuk inferensi, sebagaimana ditunjukkan pada Gambar 10.2.

Pada lapisan perangkat keras, HSM tertanam (*embedded* HSM) menyimpan dan mengelola kunci unik perangkat secara aman, memastikan bahwa materi kunci tidak keluar dari batasan HSM dalam bentuk teks biasa. Pustaka kriptografi pada Guest OS tidak menangani kunci secara langsung ataupun menjalankan operasi kriptografi sensitif itu sendiri; sebaliknya, pustaka tersebut menyediakan API yang digunakan oleh lapisan perangkat lunak yang lebih tinggi untuk meminta layanan kriptografi dari HSM. Pada lapisan aplikasi, manajer keamanan mengoordinasikan permintaan layanan keamanan, sementara manajer DNN mengatur siklus hidup model.

Ketika model DNN perlu dimuat, manajer DNN akan memanggil manajer keamanan, yang kemudian meminta dekripsi model melalui API kriptografi GuestOS. Permintaan tersebut diteruskan ke perangkat lunak kriptografi yang berada di dalam HSM, yang menggunakan kunci unik perangkat untuk mendekripsi model. Model yang telah didekripsi ditulis langsung ke dalam wilayah memori terisolasi yang hanya dapat diakses oleh mesin perangkat keras inferensi, dan tidak dikembalikan dalam bentuk teks biasa (*plaintext*) ke lapisan aplikasi umum. Dengan cara ini, persyaratan terpenuhi tanpa mengekspos kunci atau konten model yang telah didekripsi kepada perangkat lunak yang tidak tepercaya, sehingga menjaga prinsip *secure-by-design* (aman sejak perancangan) di seluruh lapisan perangkat keras, sistem operasi, dan aplikasi, sebagaimana diilustrasikan dalam Gambar 10.2:



Gambar 10.2 Arsitektur aman untuk memastikan model DNN didekripsi dengan cara yang aman

Prinsip dan Pola Desain Keamanan

Saat berupaya memenuhi persyaratan keamanan untuk solusi arsitektur tertentu, para insinyur sering kali kesulitan dalam memilih pendekatan arsitektur yang tepat. Salah satu cara yang andal untuk memperkuat desain keamanan adalah dengan memulai dari prinsip-prinsip keamanan serta pola desain yang aman, lalu menerapkannya secara konsisten. NIST SP 800-160 merupakan referensi umum untuk prinsip-prinsip desain; namun, prinsip-prinsip tersebut bisa terasa abstrak jika tidak diterjemahkan ke dalam keputusan teknis. Oleh karena itu, kita akan terlebih dahulu berfokus pada prinsip-prinsip yang paling berdampak langsung terhadap permukaan serangan (*attack surface*) dan batasan kepercayaan (*trust boundaries*) dalam sistem otomotif, kemudian beralih ke pola desain yang aman:

- **Pemisahan Domain (*Domain separation*)**: Pemisahan domain mengisolasi subsistem yang memiliki tingkat kepentingan keamanan berbeda. Sebagai contoh, memisahkan sistem yang krusial bagi keselamatan seperti ADAS dari sistem yang tidak krusial seperti sistem hiburan (*infotainment*) dapat membatasi risiko penyebaran serangan dalam desain komputer pusat. Teknik-teknik seperti isolasi berbasis perangkat keras, firewall IP, dan enkripsi memori memastikan bahwa kompromi pada satu domain tidak memengaruhi domain lainnya.
- **Hak Akses Minimum (*Least privilege*)**: Prinsip hak akses minimum memastikan bahwa setiap komponen sistem, proses, atau pengguna hanya memiliki izin yang diperlukan untuk menjalankan fungsi spesifiknya. Sebagai contoh, aplikasi hiburan yang dirancang untuk memutar media seharusnya tidak memiliki akses ke kunci kriptografis yang digunakan untuk menghasilkan pesan terautentikasi di dalam kendaraan, membuka layanan diagnostik, atau mengotorisasi pembaruan perangkat lunak. Jika seorang penyerang mengeksploitasi aplikasi media tersebut, hak akses yang berlebihan dapat memungkinkan mereka untuk mengirim pesan yang dianggap tepercaya, mengakses prosedur diagnostik yang terlindungi, atau mengganggu mekanisme pembaruan. Dengan membatasi izin, dampak kompromi kemungkinan besar akan tetap

terlokalisasi, sehingga mengurangi kemampuan penyerang untuk berpindah ke bagian lain (*pivoting*), meningkatkan hak akses, atau memengaruhi fungsi-fungsi yang berkaitan dengan keselamatan.

- **Pertahanan Berlapis (*Defense-in-depth*)**: Pertahanan berlapis melibatkan penerapan berbagai lapisan langkah keamanan untuk melindungi sistem otomotif dari kegagalan atau serangan. Saat merancang suatu kendali, penting untuk menyusun langkah-langkah keamanan secara berlapis guna memastikan bahwa kegagalan pada satu titik tidak menyebabkan pelanggaran total terhadap tujuan keamanan.
- **Pengurangan Kompleksitas (*Reduced complexity*)**: Kompleksitas dalam sistem otomotif dapat memicu kesalahan dan kerentanan keamanan. Menyederhanakan desain sistem akan mengurangi kemungkinan kesalahan konfigurasi sekaligus meningkatkan keamanan. Sebagai contoh, meminimalkan jumlah opsi konfigurasi yang diterapkan pada kontrol keamanan akan mengurangi kemungkinan diterapkannya kombinasi yang salah, yang dapat menurunkan efektivitas kontrol keamanan tersebut atau bahkan menonaktifkannya sama sekali.
- **Fungsionalitas Minimal (*Least functionality*)**: Prinsip ini menekankan bahwa komponen sistem hanya boleh mencakup fitur-fitur yang diperlukan untuk peran spesifiknya. Fungsionalitas yang berlebihan meningkatkan risiko eksploitasi. Sebagai contoh, layanan diagnostik yang ditujukan untuk mode pabrik harus dinonaktifkan saat produk beralih ke mode produksi. Membatasi fungsionalitas akan memperkecil permukaan serangan (*attack surface*) dan menyederhanakan pemeriksaan keamanan.
- **Akses Yang Dimediasi (*Mediated access*)**: Akses yang dimediasi memastikan bahwa semua interaksi dengan komponen sistem dikendalikan dan diotorisasi. Hal ini dapat dicapai dengan menggunakan API yang aman, kontrol akses berbasis peran (*role-based access control*), serta mekanisme autentikasi yang tangguh. Sebagai contoh, dalam sistem dengan banyak mesin virtual (VM), *hypervisor* akan memediasi akses ke memori bersama (*shared memory*) di antara sistem operasi tamu, alih-alih mengandalkan perangkat lunak dengan hak akses lebih rendah untuk melakukannya.
- **Aman Secara Bawaan (*Secure by default*)**: Sistem otomotif harus dilengkapi dengan konfigurasi bawaan yang aman dan memprioritaskan perlindungan. Sebagai contoh, jika autentikasi paket OTA bersifat opsional, maka pengaturan bawaannya harus berupa kewajiban autentikasi, bukan mengharapkan pengguna untuk mengaktifkannya sendiri. Menerapkan pengaturan bawaan yang bersifat protektif akan mengurangi risiko kerentanan yang disebabkan oleh konfigurasi yang tidak tepat.

Selain prinsip-prinsip perancangan keamanan, pola perancangan keamanan (*security design patterns*) dapat sangat berguna untuk mendekatkan konsep-konsep tersebut ke dalam ranah masalah yang sedang dihadapi oleh para insinyur. Ketersediaan sekumpulan pola perancangan yang relevan dengan produk tertentu dapat mempercepat proses perancangan keamanan, karena para insinyur dapat dengan mudah mengidentifikasi pola yang sesuai serta memeriksa seberapa baik pola tersebut telah diterapkan. Selanjutnya, kita akan meninjau beberapa pola perancangan keamanan umum yang dapat diterapkan pada sistem otomotif.

Dekomposisi Berbasis Ketidakpercayaan (*distrustful decomposition*)

Jika seluruh fungsionalitas berada dalam satu entitas perangkat lunak, kerentanan pada satu bagian dapat membahayakan keseluruhan sistem. Untuk menghindarinya, bagilah sistem menjadi komponen-komponen lebih kecil yang saling tidak memercayai satu sama lain. Setiap komponen menerapkan batasan antarmuka yang ketat dan beroperasi dengan hak akses minimal yang diperlukan untuk menjalankan fungsinya. Sebagai contoh, pertimbangkan sistem dengan beberapa mesin virtual (VM) yang semuanya memiliki akses ke perangkat penyimpanan bersama. Dengan merancang server penyimpanan khusus di luar batasan semua VM tersebut, tidak ada VM yang perlu memercayai VM lain terkait akses ke partisi penyimpanannya. Dengan memisahkan fungsi pengelolaan akses ke partisi penyimpanan ke dalam server penyimpanan tersendiri yang memiliki hak akses dan alokasi perangkat kerasnya sendiri kita telah berhasil melakukan dekomposisi fungsi sekaligus menjaga tingkat pemisahan akses penyimpanan yang tepat.

Rantai Kepercayaan (*chain of trust*)

Rantai proses *booting* yang kompleks terdiri dari berbagai lingkungan eksekusi *runtime* dengan citra perangkat lunak (*software image*) khusus dan tanda tangan digital yang terpisah. Jika komponen yang tidak tepercaya diizinkan untuk melakukan booting atau memuat perangkat lunak di sepanjang rantai ini tanpa verifikasi, kode berbahaya dapat dieksekusi sebelum pertahanan pada lapisan yang lebih tinggi aktif. Dalam situasi seperti ini, mulailah dari akar kepercayaan (*root of trust*) yang berlandaskan pada perangkat keras seperti kode ROM, HSM (*Hardware Security Module*), atau *bootloader* yang aman kemudian perluas kepercayaan tersebut langkah demi langkah ke setiap tahapan perangkat lunak berikutnya. Setiap tahapan wajib memverifikasi keaslian dan integritas tahapan selanjutnya sebelum menyerahkan kendali untuk eksekusi.

Mari kita ambil contoh HPC (*High-Performance Computing*) otomotif yang menjalankan *hypervisor*, beberapa sistem operasi guest, dan aplikasi persepsi berbasis AI. Jika boot ROM terlebih dahulu memverifikasi *bootloader* utama, kemudian *bootloader* memverifikasi citra *hypervisor*, dan *hypervisor* memverifikasi setiap citra VM guest sebelum dijalankan, maka tidak ada tahap yang akan dieksekusi kecuali tahap tepercaya sebelumnya memberikan persetujuan secara eksplisit. Dengan cara ini, kepercayaan diperluas secara bertahap mulai dari kode yang tertanam secara permanen di perangkat keras hingga ke seluruh tumpukan perangkat lunak HPC sehingga VM persepsi yang telah dimanipulasi tidak akan bisa dijalankan, meskipun VM tersebut tersimpan dalam perangkat penyimpanan yang secara keseluruhan dianggap sah.

Pihak Ketiga Tepercaya (*Trusted Third Party* – TTP)

Komponen atau layanan terpisah yang tidak saling memercayai secara langsung tetap memerlukan cara yang andal untuk mengautentikasi identitas dan membangun komunikasi yang aman. Tanpa adanya landasan kepercayaan (*trust anchor*) bersama, hubungan kepercayaan antar-entitas menjadi sulit untuk dikembangkan (*scalable*) dan dikelola. Untuk mengatasi masalah ini, digunakanlah pihak ketiga tepercaya yang menjamin identitas, menerbitkan kredensial, atau menjadi perantara hubungan kepercayaan di antara entitas-entitas yang berkomunikasi.

Sebagai contoh, bayangkan sebuah ECU telematika kendaraan yang harus menjalin komunikasi aman dengan sistem *backend* OEM, platform manajemen armada, dan perangkat servis yang digunakan di dealer. Alih-alih menyediakan materi kepercayaan yang dipertukarkan secara manual untuk setiap titik akhir (*endpoint*), OEM dapat mengoperasikan otoritas sertifikat (*certificate authority*) yang menandatangani kredensial bagi semua pihak yang disetujui. Dengan demikian, ECU telematika hanya perlu memercayai otoritas sertifikat OEM tersebut, sementara setiap layanan eksternal membuktikan identitasnya melalui sertifikat yang diterbitkan oleh otoritas tersebut. Melalui kehadiran pihak ketiga tepercaya ini, kendaraan tidak lagi bergantung pada hubungan kepercayaan bilateral yang bersifat ad hoc di antara setiap entitas yang terhubung.

Penyedia Identitas (*Identity Provider* – IdP)

Ketika berbagai layanan memerlukan informasi identitas pengguna atau perangkat, penyimpanan dan pengelolaan kredensial secara terpisah di setiap layanan akan menyebabkan duplikasi, ketidakkonsistenan dalam penerapan kebijakan, serta memperluas celah serangan. Untuk menghindari hal ini, pengelolaan identitas dapat dipusatkan pada satu penyedia identitas yang mengautentikasi subjek (*principal*) serta menerbitkan token atau pernyataan (*assertion*) yang dapat divalidasi oleh layanan lain.

Sebagai contoh, pertimbangkan sebuah HPC otomotif yang menjalankan aplikasi infotainment, manajemen profil pengemudi, kontrol fitur berbasis langganan, dan integrasi dengan aplikasi pendukung jarak jauh. Jika setiap aplikasi menyimpan nama pengguna, kata sandi, atau status sesi secara mandiri, maka pengelolaan kredensial akan menjadi terpecah-pecah dan rentan terhadap kesalahan. Dengan menghadirkan layanan penyedia identitas khusus di dalam ekosistem kendaraan yang terhubung, pengemudi cukup melakukan autentikasi satu kali dan menerima token yang dapat divalidasi oleh aplikasi hiburan (*infotainment*), layanan personalisasi kursi, serta pengelola fitur premium. Dengan demikian, pembuktian identitas dipusatkan, sementara layanan-layanan terkait tidak perlu lagi mengelola penyimpanan kredensial secara terpisah.

Autentikator

Jika setiap subsistem menerapkan logika autentikasinya sendiri, perilaku keamanan akan menjadi tidak konsisten, dan kesalahan pada satu subsistem dapat merusak kredibilitas seluruh ECU atau fungsi kendaraan. Memusatkan autentikasi ke dalam komponen autentikator khusus yang memverifikasi kredensial, sertifikat, token, atau permintaan yang ditandatangani sebelum modul lain memberikan akses akan membantu mencegah masalah tersebut. Pertimbangkan pengontrol zona (*zonal controller*) yang mendukung diagnostik lokal, pembaruan perangkat lunak yang aman (*secure flashing*), dan tindakan pemeliharaan melalui perangkat servis resmi. Jika setiap layanan diagnostik menerapkan logika penguraian sertifikat serta mekanisme challenge-response secara mandiri, ketidakkonsistenan kemungkinan besar akan muncul.

Dengan menghadirkan layanan autentikator khusus di dalam ECU, semua sesi diagnostik yang masuk akan diarahkan terlebih dahulu melalui komponen autentikasi yang sama. Komponen ini memvalidasi sertifikat penguji, memeriksa kebijakan otorisasi, dan baru

kemudian mengeluarkan token sesi untuk fungsi yang diminta. Pendekatan ini memastikan bahwa proses seperti *secure flashing*, pembukaan kunci untuk debug, dan pembaruan konfigurasi bergantung pada satu jalur autentikasi yang kokoh, bukan pada beberapa jalur yang diimplementasikan secara terpisah.

Serahkan Kepada Kernel

Menerapkan kembali penegakan keamanan di dalam kode aplikasi sering kali menyebabkan perilaku yang tidak konsisten, mediasi yang tidak lengkap, dan bug implementasi yang sebenarnya dapat dihindari. Pemeriksaan keamanan di lapisan aplikasi (*application layer*) mudah dilewati jika platform sistem operasi yang mendasarinya tidak menegakkannya. Untuk mencegah hal ini, serahkan penegakan keamanan yang krusial kepada kernel sistem operasi, *hypervisor*, atau mekanisme platform tepercaya tingkat rendah lainnya. Mekanisme ini mampu menegakkan kebijakan dengan isolasi yang lebih kuat dan meminimalkan celah untuk melakukan pintasan (*bypass*). Pertimbangkan sebuah *High-Performance Computer* (HPC) otomotif yang menjalankan beberapa aplikasi berbasis Linux, seperti navigasi, pemutaran media, dan pengaturan kendaraan. Jika setiap aplikasi secara mandiri menentukan berkas, socket, atau wilayah memori bersama (*shared memory*) mana yang boleh diakses, satu kesalahan pengodean saja dapat mengekspos aset sensitif.

Sebagai gantinya, platform dapat mengandalkan mekanisme yang ditegakkan oleh kernel, seperti kredensial proses, *namespace*, kontrol akses wajib (*mandatory access control*), dan partisi memori yang dikelola oleh *hypervisor*. Dalam desain ini, aplikasi navigasi tidak sekadar "memilih" untuk tidak mengakses data kontrol kendaraan; kernel dan *hypervisor* memastikan bahwa aplikasi tersebut tidak dapat mengakses sumber daya tersebut, bahkan jika aplikasi itu sendiri telah disusupi. Dengan menyerahkan penegakan keamanan kepada kernel dan lapisan virtualisasi, kebijakan keamanan menjadi lebih sulit untuk dilewati dan lebih mudah untuk dipahami atau diperiksa.

Pemisahan Hak Akses (*Privilege separation*)

Ketika berbagai fungsi dijalankan dengan tingkat hak akses tinggi yang sama, penyusupan pada satu fungsi akan memberikan akses yang tidak semestinya kepada penyerang untuk menjangkau sumber daya dan layanan lain yang tidak terkait. Untuk menghindari hal ini, bagilah sistem menjadi proses atau komponen terpisah dengan tingkat hak akses dan kemampuan yang berbeda, sehingga setiap fungsi hanya memperoleh izin yang diperlukan untuk perannya.

Pertimbangkan sebuah ECU gerbang (*gateway*) berbasis MCU yang mendukung perutean paket, diagnostik aman, pencatatan log (*logging*), dan penanganan pembaruan *firmware*. Jika semua fungsi ini dijalankan dalam satu proses dengan hak akses istimewa (*privileged process*), maka celah pada pengurai log (*logging parser*) dapat memungkinkan penyerang memperoleh wewenang yang sama dengan logika pembaruan *firmware*. Dengan memisahkan fungsi-fungsi tersebut ke dalam tugas-tugas (*tasks*) berbeda yang memiliki tingkat hak akses yang berbeda pula, tugas pencatatan log dapat dibatasi hanya untuk akses tulis (*write-only*) ke penyangga (*buffer*)-nya; sementara itu, hanya manajer pembaruan yang memiliki wewenang untuk memodifikasi memori *flash*, dan tugas perutean hanya dapat

mengakses antarmuka penerusan jaringan. Dengan cara ini, kompromi pada satu layanan tidak secara otomatis memberikan kendali atas seluruh operasi ECU yang sensitif terhadap keamanan.

Strategi Keamanan (faktor kebijakan)

Menanamkan (*hardcoding*) satu kebijakan keamanan ke dalam sistem akan menyulitkan adaptasi terhadap perubahan persyaratan regulasi, transisi kriptografi, atau batasan penerapan khusus pasar tertentu. Mengenkapsulasi pilihan kebijakan keamanan ke dalam strategi yang dapat dipertukarkan sehingga sistem dapat beralih antar kebijakan keamanan yang disetujui tanpa mengubah logika bisnis di sekitarnya akan membantu memitigasi masalah ini.

Pertimbangkan sistem *backend* OEM dan tumpukan telematika dalam kendaraan yang harus mendukung satu profil kriptografi untuk penerapan lama (*legacy*) dan profil lain untuk penerapan di masa depan yang menggunakan algoritma lebih kuat serta rantai sertifikat yang berbeda. Jika tumpukan komunikasi menyematkan satu kebijakan tetap di seluruh basis kode, migrasi ke kebijakan baru akan menjadi berisiko dan bersifat invasif. Dengan mengisolasi logika validasi sertifikat, metode pertukaran kunci, dan kebijakan tanda tangan ke dalam modul strategi yang terpisah, aplikasi telematika dapat memilih kebijakan yang disetujui untuk program kendaraan atau wilayah tertentu, tanpa perlu mengubah bagian perangkat lunak lainnya. Dengan demikian, kebijakan keamanan menjadi dapat dikonfigurasi dan dikelola, alih-alih tersembunyi di dalam logika aplikasi.

Pola Pembangun Keamanan (*Security builder factory*)

Subsistem yang kompleks sering kali memerlukan banyak parameter terkait keamanan selama proses konstruksi. Jika parameter-parameter ini disusun secara manual, kemungkinan terjadinya kombinasi yang tidak aman atau terlewatnya perlindungan yang diperlukan akan meningkat. Untuk mencegah hal ini, gunakan pola pembangun (*builder pattern*) yang menyusun subsistem langkah demi langkah, sembari menerapkan pilihan keamanan yang diwajibkan serta menolak konfigurasi yang tidak lengkap atau tidak aman.

Pertimbangkan komponen perangkat lunak ADAS pada HPC (*High-Performance Computer*) otomotif yang harus dikonfigurasi dengan saluran sensor yang aman, sumber pesan yang terautentikasi, penyimpanan model yang terenkripsi, serta jalur komunikasi antarproses yang telah disetujui. Jika pengembang melakukan instansiasi subsistem ini dengan mengatur puluhan kolom konfigurasi secara manual, ada risiko mereka melupakan mode autentikasi yang diwajibkan atau secara tidak sengaja mengizinkan pengiriman data dalam bentuk teks polos (*plaintext*) untuk saluran yang bersifat sensitif.

Dengan menggunakan *security builder factory*, setiap tahapan konstruksi akan memvalidasi bahwa opsi-opsi yang aman telah dipilih sebelum subsistem tersebut dapat diselesaikan. Sebagai contoh, builder dapat menolak untuk merampungkan runtime ADAS kecuali jika input sensor dikaitkan dengan sumber yang terautentikasi dan pemuatan model dikaitkan dengan penyimpanan yang terenkripsi. Hal ini memastikan bahwa subsistem yang dihasilkan tersusun dalam kondisi yang aman, sehingga tidak lagi bergantung pada ingatan pengembang untuk menerapkan setiap pengaturan yang diperlukan.

Rantai Tanggung Jawab yang Aman

Satu titik pemeriksaan keamanan sering kali tidak memadai ketika data atau perintah melewati berbagai lapisan, jaringan, atau ECU (*Electronic Control Unit*). Jika tahapan perantara tidak menerapkan pemeriksaan sendiri, pesan yang valid dalam satu konteks bisa menjadi berbahaya dalam konteks lain. Dalam situasi seperti ini, terapkanlah rantai tanggung jawab di mana setiap penangan (*handler*) dalam alur pesan menjalankan tindakan keamanan yang telah ditetapkan sebelum meneruskan pesan ke tahap berikutnya.

Sebagai contoh, pertimbangkan perintah membuka kunci pintu jarak jauh yang berasal dari cloud OEM dan melewati ECU telematika, *gateway* pusat, serta pengontrol bodi (*body controller*) sebelum mengaktifkan modul kunci. Jika sistem hanya mengandalkan satu pemeriksaan keamanan di titik masuk, maka asumsi pada tahap selanjutnya dapat dilanggar melalui serangan pengulangan (*replay*), penyalahgunaan perutean, atau ketidaksesuaian kebijakan.

Dengan menerapkan rantai tanggung jawab yang aman, ECU telematika akan memverifikasi tanda tangan pesan dari *cloud*; *gateway* pusat memverifikasi apakah perintah tersebut diizinkan dalam kondisi kendaraan saat ini dan apakah berasal dari domain sumber yang resmi; dan pengontrol bodi memeriksa batasan eksekusi lokal sebelum benar-benar membuka kunci. Setiap tahapan menambahkan keputusan keamanannya sendiri, sehingga memastikan bahwa tindakan akhir bergantung pada serangkaian tanggung jawab yang tervalidasi, bukan sekadar satu keputusan kepercayaan yang dibuat di tahap awal.

Implementasi yang Aman

Hingga tahap ini, pendekatan terkelola telah menetapkan tujuan keamanan, menerjemahkannya ke dalam persyaratan dan arsitektur, serta mengalokasikan kontrol di seluruh desain sistem. Namun, fase implementasi justru merupakan tahap di mana banyak kerentanan sering kali muncul. Hal ini berlaku baik untuk produk yang berbasis perangkat lunak, IP perangkat keras, *firmware*, maupun kombinasi dari ketiganya. Jika pendekatan terkelola hanya berhenti pada arsitektur yang baik tanpa mengatur bagaimana implementasi dilakukan, maka organisasi tersebut pada dasarnya hanya menghasilkan produk yang tidak aman meskipun memiliki dokumentasi yang lengkap. Oleh karena itu, implementasi yang aman harus diperlakukan sebagai aktivitas rekayasa yang terkelola, bukan sekadar bergantung pada preferensi pengembang. Pada tahap ini, tujuannya adalah mengurangi kemungkinan munculnya kelemahan saat menerjemahkan desain menjadi kode yang dapat dieksekusi, perangkat keras yang dapat disintesis, artefak konfigurasi, dan logika integrasi.

Dengan demikian, CSMS (*Cyber Security Management System*) mengharuskan agar implementasi dilakukan dengan mengikuti aturan penulisan kode (*coding rules*) yang ditetapkan, disiplin peninjauan, serta pemeriksaan otomatis. Dalam pengembangan perangkat lunak, hal ini biasanya berarti menerapkan pedoman pengkodean yang aman, melakukan pemantauan sejawat dengan pemeriksaan yang relevan dengan keamanan, dan menggunakan alat pemeriksaan statis untuk mengidentifikasi konstruksi dan pelanggaran berbahaya sebelum menyebar ke hilir. Dalam bahasa pemrograman yang tidak aman terhadap memori, praktik-praktik ini sangat penting karena cacat implementasi umum seperti penulisan di luar

batas, kondisi penggunaan setelah pembebasan memori, rantai luapan bilangan bulat, dan kondisi persaingan dapat menjadi primitif eksploitasi langsung. Dalam konteks jaminan keamanan yang lebih tinggi, organisasi juga dapat mengadopsi bahasa pemrograman yang aman terhadap memori secara selektif untuk komponen yang kritis terhadap keamanan guna mengurangi seluruh kelas kerentanan melalui konstruksi.

Logika yang sama juga berlaku untuk pengembangan perangkat keras dan sistem tertanam (*embedded system*) tingkat rendah. Proses implementasi yang lemah pada RTL, kode inisialisasi *firmware*, logika *secure boot*, kontrol akses debug, atau penegakan status siklus hidup produk dapat merusak konsep keamanan siber yang sebenarnya sudah dirancang dengan baik. Faktanya, arsitektur yang aman tidak akan bertahan jika implementasinya dilakukan secara ceroboh. Intinya, prinsip *secure by design* (aman sejak tahap perancangan) harus diikuti dengan *secure by implementation* (aman pada tahap implementasi). Prinsip pertama menentukan apa yang harus dilakukan produk untuk melawan serangan, sementara prinsip kedua mengatur bagaimana rancangan tersebut diwujudkan tanpa menimbulkan kerentanan yang tidak perlu.

Faktor lain yang mulai muncul adalah meningkatnya penggunaan perangkat pengembangan berbasis kecerdasan buatan (AI). Generator kode berbasis AI memang dapat meningkatkan produktivitas, namun di sisi lain, hal ini turut mengubah model risiko implementasi. Alat-alat tersebut dapat mereproduksi pola yang tidak aman, menerapkan API secara keliru, atau menghasilkan alur kode yang tampak meyakinkan padahal sebenarnya tidak aman terutama jika instruksi (*prompt*) yang diberikan ambigu atau asumsi keamanan yang mendasarinya tidak dirumuskan dengan jelas.

Oleh karena itu, kode yang dihasilkan oleh AI tidak boleh langsung dianggap aman hanya karena mampu mempercepat proses implementasi. Kode tersebut tetap harus tunduk pada aturan penulisan kode yang aman, standar peninjauan, dan ketentuan pemeriksaan yang sama seperti kode yang ditulis secara manual. Dalam praktiknya, kehadiran AI dalam rangkaian perangkat pengembangan justru memperkuat, bukan melemahkan, urgensi penerapan kontrol implementasi yang terkelola.

Di saat yang sama, perangkat berbasis AI untuk peninjauan kode dan pendeteksian kerentanan kini semakin menjadi bagian standar dalam lingkungan pengembangan. Perangkat ini mampu memeriksa basis kode yang luas, mengidentifikasi alur kode yang mencurigakan, mengaitkan kelemahan dengan pola eksploitasi yang sudah diketahui, serta membantu analis dalam membuktikan potensi eksploitasi. Jika digunakan dengan tepat, perangkat ini dapat membantu organisasi menemukan celah implementasi lebih dini dengan cakupan yang lebih luas. Namun, jika digunakan secara sembarangan, perangkat ini justru dapat menciptakan rasa aman yang semu. Oleh karena itu, pendekatan yang terkelola memandang perangkat tersebut sebagai sarana pendukung dalam sebuah proses yang memiliki tata kelola, bukan sebagai pengganti penilaian teknis (*engineering judgment*) ataupun bukti yang dapat ditelusuri.

Pengujian Verifikasi

Dalam konteks model-V (*V-model*), verifikasi merupakan fase di mana organisasi mengevaluasi apakah produk telah dibangun dengan benar sesuai dengan spesifikasi, artefak

desain, dan aturan rekayasa yang ditetapkan. Dalam keamanan siber, ini berarti memastikan bahwa kontrol keamanan yang telah ditentukan sebelumnya dalam siklus hidup pengembangan telah diimplementasikan sebagaimana mestinya, serta memastikan bahwa hasil kerja (*work products*) pada tahap-tahap perantara mencerminkan desain yang disyaratkan secara akurat. Pertanyaan utama untuk verifikasi sangatlah lugas: "Apakah kita membangun produk dengan cara yang benar?"

Verifikasi pada dasarnya berkaitan dengan kesesuaian (*konformitas*). Proses ini memeriksa apakah persyaratan sudah tepat, apakah artefak desain mewujudkan persyaratan tersebut dengan benar, apakah implementasi kode dan perangkat keras mengikuti desain yang ada, serta apakah aturan dan standar rekayasa telah dipatuhi selama pengembangan. Dengan kata lain, verifikasi memastikan bahwa produk yang dibangun selaras dengan tujuan yang ditetapkan dalam proses yang dikelola. Aktivitas verifikasi yang umum meliputi tinjauan persyaratan, tinjauan arsitektur, tinjauan antarmuka, tinjauan kode aman (*secure code review*), analisis statis, tinjauan kasus uji, pengujian integrasi, pengujian sistem, dan pemeriksaan konsistensi di seluruh hasil kerja. Verifikasi dimulai sejak tahap awal; persyaratan diperiksa untuk memastikan ketepatan dan kelengkapannya, arsitektur diperiksa kesesuaiannya dengan persyaratan, dan implementasi diperiksa terhadap desain serta aturan pengodean. Proses ini berlanjut hingga tahap eksekusi pengujian, di mana komponen yang terintegrasi dan sistem secara keseluruhan dipastikan telah memenuhi persyaratan yang telah diverifikasi serta tujuan desain yang ditetapkan.

Untuk komponen dengan tingkat jaminan (*assurance*) yang tinggi, organisasi dapat menerapkan bentuk verifikasi yang lebih ketat, seperti metode formal. Teknik-teknik ini tidak menggantikan tinjauan rekayasa konvensional, melainkan memperkuat jaminan untuk fungsi-fungsi yang krusial bagi keamanan terutama di mana pengujian biasa tidak mampu mencakup seluruh ruang keadaan (*state space*) secara menyeluruh. Metode semacam ini sangat bernilai dalam situasi di mana kegagalan dapat menimbulkan dampak besar, dan di mana properti keamanan dapat dirumuskan secara cukup presisi untuk dibuktikan atau diperiksa secara tuntas.

Pengujian Validasi

Fase terakhir sebelum peluncuran adalah validasi. Jika verifikasi berfokus pada apakah sistem dibangun sesuai dengan spesifikasi, maka validasi berfokus pada apakah sistem yang terintegrasi tersebut benar-benar mencapai tujuan keamanan siber yang diinginkan dalam konteks operasionalnya. Pertanyaan utama dalam tahap ini adalah: "Apakah kita membangun produk yang tepat?" Perbedaan ini penting karena suatu produk mungkin berhasil memenuhi spesifikasinya, namun tetap gagal memberikan keamanan yang berarti dalam kondisi nyata misalnya jika spesifikasinya tidak lengkap, asumsi yang digunakan tidak realistis, atau jika proses integrasi justru memunculkan jalur serangan baru. Oleh karena itu, standar ISO/SAE 21434 mengharuskan kegiatan validasi keamanan siber dilakukan pada tingkat kendaraan, di mana berbagai komponen diintegrasikan ke dalam produk utuh, termasuk dengan sistem pendukung dan infrastruktur *backend* yang relevan.

Validasi keamanan siber berfokus pada apakah sistem dapat beroperasi secara aman dalam praktiknya saat menghadapi ancaman yang mungkin terjadi. Hal ini mencakup pengujian dinamis terhadap sistem yang sedang berjalan, evaluasi permukaan serangan dalam konfigurasi yang realistis, serta pemeriksaan apakah sistem akhir mampu menahan akses tidak sah, manipulasi, kompromi data, eskalasi hak akses, dan serangan relevan lainnya. Aktivitas validasi yang umum meliputi pengujian penetrasi (*penetration testing*), fuzz testing, analisis dinamis, evaluasi kasus penyalahgunaan (*abuse case*) di tingkat sistem, dan pengujian operasional berdasarkan kondisi penggunaan yang diperkirakan. Berbeda dengan verifikasi yang sering kali memeriksa apakah artefak rekayasa sesuai dengan tujuan awal validasi menguji produk aktual untuk memastikan bahwa perlindungan yang direncanakan tetap tangguh terhadap serangan di dunia nyata.

Tentu saja, pengujian penetrasi merupakan metode ideal untuk validasi keamanan karena berupaya meniru perilaku penyerang nyata dalam lingkup penilaian yang terbatas. Metode ini tidak hanya membantu menentukan apakah kontrol keamanan tersedia, tetapi juga apakah kontrol tersebut benar-benar mampu mencegah eksploitasi dalam lingkungan yang terintegrasi. Inilah salah satu alasan mengapa teknik keamanan ofensif sangat berharga dalam program yang terkelola: teknik tersebut memberikan realisme yang diperlukan untuk menguji asumsi-asumsi yang dibuat pada tahap awal siklus hidup pengembangan.

Fuzz testing juga memegang peranan penting karena banyak permukaan serangan pada sistem otomotif yang digerakkan oleh parser, bersifat stateful (bergantung pada status), atau terpapar pada input yang tidak sesuai format (*malformed inputs*) di sepanjang batasan kepercayaan (*trust boundaries*) yang kompleks. Validasi yang mengabaikan perilaku sistem saat menerima input yang tidak sesuai format dianggap tidak lengkap. Analisis dinamis pun sama berharganya, karena beberapa kelemahan hanya akan terlihat ketika sistem dioperasikan dalam kondisi runtime yang realistis. Dalam praktiknya, verifikasi dan validasi saling melengkapi. Verifikasi mendeteksi penyimpangan dari persyaratan, arsitektur, dan aturan pengodean sebelum masalah tersebut meluas. Validasi memastikan bahwa produk yang telah terintegrasi sepenuhnya mampu bertahan terhadap ancaman nyata dan bahwa tujuan keamanan siber yang ditetapkan sejak awal benar-benar tercapai.

Peluncuran Produk

Sebelum suatu produk diluncurkan, kesiapan keamanan sibernya harus dinilai untuk memastikan apakah pengurangan risiko yang direncanakan sejak awal telah tercapai dengan bukti yang memadai. Dalam pendekatan yang terkelola, keputusan mengenai kesiapan ini tidak boleh didasarkan pada optimisme semata atau tekanan jadwal, melainkan harus didukung oleh *cybersecurity case* (argumen keamanan siber). *Cybersecurity case* merupakan argumen terstruktur yang menjelaskan mengapa produk tersebut siap untuk diluncurkan kepada pengguna jalan atau integrator di tahap selanjutnya. Dokumen ini menyatukan bukti berbasis proses maupun bukti teknis yang menunjukkan bahwa risiko keamanan siber telah dikelola hingga mencapai tingkat yang dapat diterima.

Dari sisi proses, kasus ini menunjukkan bahwa kegiatan keamanan siber yang telah direncanakan telah dilaksanakan, serta produk kerja yang dihasilkan bersifat lengkap,

konsisten, dan terkendali. Hal ini mencakup referensi ke berbagai artefak seperti TARA (*Threat Analysis and Risk Assessment*), persyaratan keamanan siber, catatan desain, perjanjian antarmuka, laporan verifikasi, laporan validasi, bukti dari pemasok, dan catatan rilis yang dikendalikan konfigurasinya. Setiap penyimpangan yang terjadi akan dicatat dan disertai dengan argumen risiko keamanan siber yang menjelaskan mengapa penyimpangan tersebut dapat diterima dalam konteks produk tertentu serta asumsi penggunaannya.

Dari sisi teknis, kasus ini menunjukkan bahwa penyimpangan, kelemahan, dan risiko residual yang teridentifikasi tidak sekadar diabaikan. Hal-hal tersebut diidentifikasi, dianalisis, ditentukan tindak lanjutnya, lalu diperbaiki atau secara eksplisit dipertahankan dengan disertai alasan yang mendasarinya. Pembedaan ini sangat penting karena setiap ketidaksesuaian atau kelemahan yang ditemukan harus berujung pada pengambilan keputusan teknis, bukan sekadar menjadi catatan dalam laporan pengujian.

Salah satu dokumen kerja yang sangat krusial pada tahap perilisan adalah manual keamanan siber beserta dokumentasi integrasi pendukungnya. Dokumen ini bertujuan untuk mengomunikasikan asumsi, batasan, persyaratan konfigurasi, dan langkah-langkah operasional yang harus dipatuhi agar produk tetap aman saat diintegrasikan oleh pelanggan atau pemasok di rantai pasok hulu. Sebuah produk mungkin aman saat dikembangkan, namun menjadi tidak aman saat diintegrasikan jika asumsi-asumsi tersebut tidak dikomunikasikan dan dijaga.

Ini bukan sekadar kekhawatiran teoretis, melainkan masalah nyata yang terbukti dapat menyebabkan kegagalan keamanan di dunia nyata. Faktanya, penyediaan (*provisioning*) yang tidak tepat, transisi status siklus hidup yang keliru, atau penguncian pengaturan perangkat keras yang terlalu dini dapat membuat sistem berada dalam kondisi yang tidak aman atau bahkan tidak dapat dipulihkan. Contoh umum yang sering terjadi adalah kesalahan dalam pemrograman fuse sebelum sistem dilengkapi sepenuhnya dengan data rahasia dan konfigurasi yang diperlukan. Begitu transisi siklus hidup yang salah terjadi, produk tersebut mungkin tidak dapat dipulihkan kembali kecuali melalui pengerjaan ulang yang memakan biaya besar atau penggantian unit. Terlepas dari apakah informasi ini dituangkan dalam manual keamanan siber formal atau dokumentasi terkendali lainnya, persyaratan utamanya tetap sama: asumsi dan ketentuan untuk pengaturan serta integrasi yang aman harus didokumentasikan secara eksplisit.

Perencanaan Produksi

Proses manufaktur, penyebaran, dan perilisan harus dipertimbangkan untuk memastikan bahwa sistem diproduksi dan diserahkan dengan aman. Meskipun tidak semua produk memerlukan tingkat perencanaan produksi yang sama, setiap produk yang mengalami transisi dari tahap pengembangan ke tahap perilisan di lapangan harus memperhitungkan implikasi keamanan yang muncul selama transisi tersebut. Masalah mendasarnya adalah keamanan tidak berhenti saat desain selesai. Sebuah sistem mungkin dirancang dengan karakteristik keamanan yang kuat, namun jika proses yang memindahkan produk dari lingkungan pengembangan ke lapangan tidak dikendalikan dengan baik, karakteristik keamanan tersebut dapat melemah bahkan sebelum sistem digunakan oleh pelanggan.

Bagi beberapa produk, tantangan perencanaan produksi relatif terbatas. Sebagai contoh, pertimbangkan pustaka perangkat lunak (*software library*) yang diintegrasikan oleh pengembang lain. Produk semacam ini mungkin tidak melibatkan manufaktur fisik, tetapi tetap melalui transformasi dari artefak pengembangan menjadi komponen perangkat lunak yang didistribusikan. Meskipun tidak ada lantai pabrik fisik, organisasi tetap harus menjawab pertanyaan seperti bagaimana perangkat lunak dikemas, bagaimana perilisan disetujui, bagaimana integritas versi dijaga, dan bagaimana distribusi dikendalikan. Jika artefak perangkat lunak dapat dimodifikasi antara tahap pengembangan dan penggunaan, maka ketiadaan manufaktur fisik tidak menghilangkan masalah keamanan produksi; masalah tersebut hanya berubah bentuk. Pengemasan yang aman, validasi integritas perilisan, dan saluran distribusi yang terautentikasi menjadi mekanisme untuk menjaga kepercayaan terhadap komponen yang diserahkan.

Untuk produk perangkat keras, tantangan perencanaan produksi menjadi lebih luas dan sering kali jauh lebih sensitif. Organisasi yang memproduksi cip, elemen keamanan (*secure elements*), atau bahkan unit kendali elektronik (*Electronic Control Unit* atau ECU) lengkap harus mempertimbangkan bagaimana karakteristik keamanan tetap terjaga di sepanjang proses manufaktur itu sendiri. Beberapa masalah berikut ini sangat krusial dalam lingkungan tersebut:

- Bagaimana mode pengujian manufaktur dikendalikan
- Bagaimana data rahasia dimasukkan (diprovisi) ke dalam perangkat
- Bagaimana transisi status siklus hidup perangkat dilakukan selama produksi
- Bagaimana pengaturan konfigurasi yang berkaitan dengan keamanan dikunci sebelum pengiriman

Lingkungan manufaktur sering kali memerlukan mekanisme akses yang tidak dapat diterima pada sistem yang telah digunakan oleh pengguna akhir. Sebagai contoh, antarmuka untuk penelusuran kesalahan (*debug*), port akses pengujian, dan pintu belakang (*backdoor*) teknis mungkin diperlukan untuk memvalidasi perangkat keras selama produksi. Namun, begitu produk meninggalkan pabrik, antarmuka yang sama dapat menjadi vektor serangan berisiko tinggi jika tetap aktif atau tidak dibatasi dengan semestinya.

Demikian pula, penanganan aset kriptografis selama manufaktur harus dikendalikan dengan sangat cermat. Kunci identitas perangkat, sertifikat akar (*root certificate*), dan materi provisi harus dimasukkan ke dalam perangkat dengan cara yang mencegah duplikasi atau ekstraksi oleh pihak yang tidak berwenang. Jika aset-aset ini ditangani secara tidak tepat selama produksi, perangkat yang dihasilkan mungkin tampak asli, padahal secara diam-diam melanggar asumsi kepercayaan yang tertanam dalam arsitektur sistem. Contoh umum adalah penanganan data rahasia dalam fuse (komponen perangkat keras) dalam bentuk teks polos (*plaintext*) di lingkungan pabrik. Kepercayaan yang berlebihan terhadap keamanan lingkungan pabrik merupakan masalah yang terus berulang dalam industri otomotif. Faktanya, fasilitas manufaktur otomotif telah berulang kali mengalami pembobolan dalam beberapa tahun terakhir terutama melalui serangan ransomware yang seharusnya mendorong kita untuk mengevaluasi kembali sejauh mana kepercayaan dapat diberikan kepada lingkungan pabrik yang dianggap "aman" tersebut. Bahkan produk yang sepenuhnya berbasis perangkat lunak

pun mungkin memerlukan kontrol keamanan khusus untuk tahap produksi. Contohnya meliputi:

- Infrastruktur penandatanganan kode (*code signing*) yang digunakan untuk mengautentikasi binary yang dirilis
- Alur kerja persetujuan rilis yang memastikan hanya artefak resmi yang dipublikasikan
- Kontrol integritas artefak untuk mendeteksi perubahan yang tidak sah dalam alur kerja pengembangan (*build pipeline*)
- Mekanisme distribusi aman yang mencegah modifikasi tidak sah selama proses pengiriman

Oleh karena itu, pendekatan keamanan terkelola tidak memandang keamanan dalam tahap pengembangan dan keamanan dalam tahap produksi sebagai hal yang terpisah. Sebaliknya, pendekatan ini menganggap keduanya sebagai fase yang saling terhubung dalam satu rantai kepercayaan (*chain of trust*). Pengendalian yang diterapkan selama pengembangan harus terus berlanjut melalui tahap manufaktur, pengemasan, perilisan, hingga distribusi, sehingga postur keamanan yang ditetapkan sejak tahap rekayasa tetap terjaga sampai produk tersebut digunakan oleh pengguna akhir.

Pertanyaan mendasar bagi setiap organisasi sangatlah sederhana: langkah apa yang diperlukan untuk beralih dari desain rekayasa menuju produk akhir tanpa menciptakan celah serangan (*attack surface*) baru? Jawaban spesifiknya akan bervariasi tergantung pada jenis produk dan lingkungan produksinya. Namun, prinsip dasarnya tetap sama. Jika tahap produksi menghadirkan celah yang tidak dikelola seperti risiko manipulasi, perubahan konfigurasi tanpa izin, kebocoran kunci akses, atau akses debug yang tertinggal maka jaminan keamanan yang telah dibangun selama pengembangan bisa hilang bahkan sebelum pengguna akhir berinteraksi dengan sistem tersebut.

Operasional dan Pemeliharaan

Apakah tugas kita sudah selesai? Jika jawaban Anda "belum", maka Anda benar. Pendekatan keamanan terkelola tidak berakhir saat produk dirilis. Kendaraan yang terhubung (*connected vehicles*), arsitektur berbasis perangkat lunak, dan penyediaan fitur melalui sistem langganan telah memperluas cakupan celah serangan operasional hingga jauh melampaui titik penjualan. Ketika kendaraan menerima pembaruan, membuka fitur setelah pembelian, serta menjalin hubungan jangka panjang dengan layanan *backend*, keamanan siber menjadi disiplin operasional yang berkelanjutan, bukan sekadar pencapaian teknis satu kali saja.

Oleh karena itu, pengelolaan pembaruan perangkat lunak merupakan bagian krusial dari keseluruhan aspek keamanan siber. Organisasi harus mampu mengembangkan, memvalidasi, menyetujui, mendistribusikan, dan memantau pembaruan dengan cara yang tetap menjaga keamanan serta integritas sistem. Hal ini mencakup tidak hanya penambalan kerentanan, tetapi juga pengelolaan aktivasi fitur, verifikasi keaslian perangkat lunak, penanganan pemulihan (*rollback*) pembaruan, serta interaksi antara keamanan siber dan keselamatan fungsional (*functional safety*).

Tantangan ini menjadi semakin nyata seiring dengan langkah produsen peralatan asli (OEM) dalam memonetisasi fitur perangkat lunak pascapenjualan. Dalam lingkungan seperti

ini, pembajakan perangkat lunak dan pengaktifan fitur tanpa izin menjadi ancaman yang nyata. Serangan tidak lagi terbatas pada pencurian data atau melumpuhkan layanan, melainkan meluas hingga mencakup upaya memintas pemeriksaan langganan, memalsukan paket pembaruan, melakukan *replay* (pengiriman ulang) terhadap respons *backend*, atau memanipulasi penerapan kebijakan lokal guna membuka akses ke fitur-fitur premium. Sebagaimana terlihat di sektor digital lainnya, begitu suatu metode untuk memintas kendali fitur ditemukan dan bocor, metode tersebut dapat menyebar dengan cepat, bahkan ke pihak-pihak di luar kelompok penyerang yang canggih.

Oleh karena itu, keamanan pembaruan tidak boleh dianggap sekadar sebagai fitur kenyamanan. Keamanan ini merupakan bagian dari model kepercayaan kendaraan. Organisasi harus memahami apa yang diubah, mengapa perubahan tersebut dilakukan, dependensi apa yang terdampak, bukti validasi apa yang tersedia, serta apakah pembaruan tersebut dapat diterapkan dengan aman pada armada kendaraan yang dituju. Dengan demikian, pembaruan melalui udara (OTA) yang aman bergantung pada sumber yang terautentikasi, verifikasi integritas, kontrol versi, pertimbangan pencegahan *rollback* (kembali ke versi sebelumnya) jika diperlukan, tata kelola penerapan, serta pemantauan pasca-penerapan.

Aspek operasional juga mencakup pemantauan keamanan siber dan pengelolaan kerentanan. Setelah produk digunakan di lapangan, kerentanan baru akan terus muncul baik melalui pengujian internal, penelitian eksternal, pengungkapan dari pemasok, respons insiden, maupun ekosistem keamanan publik. Oleh karena itu, program yang terkelola memerlukan proses untuk menerima informasi kerentanan, menilai relevansinya, memprioritaskan perbaikan, mendokumentasikan dampaknya, serta memutuskan apakah perlu dilakukan penambalan (*patching*), mitigasi, atau menerima risiko tersebut (dengan dokumentasi yang tepat). Maraknya penggunaan perangkat lunak sumber terbuka (*open-source*) dalam ekosistem otomotif membuat tugas ini menjadi sangat menantang. Kemampuan untuk merespons dengan cepat terhadap kerentanan baru dalam perangkat lunak sumber terbuka akan menjadi krusial bagi industri otomotif agar mampu mengimbangi kecepatan serangan siber berbasis kecerdasan buatan (AI) dalam menemukan kelemahan yang dapat dieksploitasi.

Ini adalah aspek lain di mana pendekatan terkelola sangat berbeda dengan pendekatan *ad-hoc* (*spontan*/tidak terencana). Dalam keamanan ad-hoc, kerentanan sering kali memicu penambalan yang bersifat reaktif tanpa disertai jejak analisis yang komprehensif. Sebaliknya, dalam pendekatan terkelola, setiap kerentanan yang ditemukan menjadi masukan ke dalam proses yang teratur, yang dapat menghasilkan tindakan teknis, bukti pendukung, dan jika memang beralasan keputusan risiko yang dapat dipertanggungjawabkan.

Akhir Masa Pakai dan Penonaktifan

Tahap akhir dalam pendekatan keamanan terkelola kami adalah penanganan akhir masa pakai produk, di mana produk tersebut atau beberapa komponennya tidak lagi mendapatkan dukungan. Faktanya, akhir masa pakai bukanlah berarti hilangnya masalah keamanan, melainkan sebuah fase keamanan tersendiri yang memiliki risiko-risiko khas. Ketika produk, unit kontrol elektronik (ECU), layanan *backend*, atau perangkat pendukung

dinonaktifkan, organisasi harus memastikan penanganan yang tepat terhadap berbagai elemen seperti data rahasia, kredensial, otoritas penandatanganan, kemampuan diagnostik, akses pemeliharaan, data pribadi yang tersimpan, serta komponen perangkat lunak yang tidak lagi didukung. Produk yang tidak lagi mendapatkan pemeliharaan namun masih beroperasi di lapangan dapat menimbulkan risiko bagi pelanggan maupun merek, terutama jika batasan dukungan tidak ditetapkan dengan jelas dan tidak ditegakkan secara aman.

Oleh karena itu, penghentian operasional (*decommissioning*) memerlukan strategi yang terencana dengan matang. Langkah ini dapat mencakup pencabutan kredensial, penonaktifan dependensi *backend*, pengakhiran hubungan kepercayaan untuk pembaruan, penghapusan akun yang tidak terpakai, pembersihan data sensitif, serta pendokumentasian asumsi keamanan yang tidak lagi berlaku setelah dukungan berakhir. Untuk perangkat keras, organisasi mungkin juga perlu mempertimbangkan pembuangan yang aman, daur ulang yang aman, atau penarikan komponen yang memuat kunci, sertifikat, atau algoritma eksklusif.

Pelajaran yang lebih luas adalah bahwa pendekatan terkelola memandang keamanan siber sebagai disiplin siklus hidup mulai dari tahap konsep hingga penghentian operasional. Sistem Manajemen Keamanan Siber (CSMS) menjadi berharga bukan karena menciptakan beban administratif, melainkan karena menghadirkan kendali atas bagaimana risiko diperkenalkan, dianalisis, dikurangi, diverifikasi, divalidasi, dipantau, dan pada akhirnya diakhiri. Inilah perbedaan mendasar antara sekadar berharap suatu produk aman dengan kemampuan untuk membuktikan melalui bukti nyata mengapa produk tersebut cukup aman untuk digunakan. Setelah memahami pentingnya pendekatan keamanan terkelola, kita akan membahas berbagai tantangan yang menghambat upaya pengurangan risiko secara sistematis serta peningkatan standar keamanan.

10.3 MENINGKATKAN STANDAR KEAMANAN DALAM SISTEM OTOMOTIF

Dalam banyak kasus, sekadar memenuhi standar keamanan minimum saja sudah terasa sangat berat. Namun, seperti yang telah kita bahas sepanjang buku ini, para penyerang terus menyempurnakan teknik penemuan dan eksploitasi mereka. Oleh karena itu, jika para insinyur keamanan defensif gagal mengimbangi kemajuan dalam ranah serangan, kekalahan menjadi sesuatu yang tak terelakkan. Akan tetapi, sebelum membahas cara meningkatkan standar keamanan, kita perlu memahami beberapa faktor yang menjadi kendala dalam upaya tersebut.

Tekanan Proses dalam Siklus Pengembangan Yang Cepat

Sepanjang bab ini, kita telah banyak membahas perlunya sistem manajemen keamanan siber yang mampu mengurangi risiko keamanan siber secara sistematis. Hal ini lebih mudah diucapkan daripada dilakukan, terutama dalam dunia kendaraan berbasis perangkat lunak (*Software-Defined Vehicles*/SDV). SDV modern berkembang berkat inovasi, dengan mengadopsi budaya "bergerak cepat, gagal dengan cepat" (*move fast, fail fast*) yang mempercepat pengembangan. Namun, pendekatan ini sering kali berbenturan dengan protokol keamanan dan keselamatan yang ketat, sebagaimana disyaratkan oleh standar seperti ISO/SAE 21434.

Ketika kepatuhan dianggap terlalu membebani, tim mungkin merasa terdorong untuk mengabaikan proses, yang pada akhirnya merusak efektivitas manajemen risiko. Dalam skenario terburuk, upaya yang awalnya tulus untuk mengurangi risiko melalui proses yang terstruktur justru berubah menjadi sekadar formalitas hanya mencentang daftar persyaratan (*checkbox exercise*). Tujuannya pun bergeser dari upaya nyata menangani risiko menjadi sekadar penyusunan dokumen yang rumit demi memenuhi penilaian pihak eksternal. Terlebih lagi, jika manfaat langsung dari manajemen risiko yang menyeluruh tidak terlihat jelas, tim mungkin lebih memprioritaskan pemenuhan tenggat waktu daripada menangani ancaman laten yang lebih mendalam.

Pendekatan dangkal seperti ini dapat menyembunyikan kerentanan yang nantinya akan sangat mahal untuk diperbaiki di tahap lanjut siklus pengembangan. Masalah ini diperumit oleh ketergantungan pada perangkat lunak sumber terbuka (*open-source*) dan rantai pasok yang kompleks, yang menciptakan tantangan tambahan dalam menjaga kelengkapan *Software Bill of Materials* (SBOM). Menjamin visibilitas di seluruh ekosistem tersebut serta menangani kerentanan pada modul siap pakai (*off-the-shelf*) yang sejak awal tidak dirancang untuk keamanan otomotif semakin membebani sumber daya yang ada.

Kesenjangan Sumber Daya Manusia

Evolusi pesat kendaraan berbasis perangkat lunak (*Software-Defined Vehicles* atau SDV) menuntut keahlian di bidang-bidang seperti jaringan tingkat lanjut, virtualisasi, dan kontrol kriptografis ranah yang secara tradisional dikaitkan dengan TI perusahaan (*enterprise IT*). Teknologi-teknologi ini mengharuskan tenaga kerja teknik otomotif untuk mendapatkan pelatihan ulang, beralih dari model pengembangan ECU klasik yang berbasis sistem operasi waktu nyata (*Real-Time Operating System* atau RTOS) menuju arsitektur berorientasi layanan (*service-oriented architectures*). Akibatnya, transisi ke arsitektur SDV memerlukan pelatihan tenaga kerja secara ekstensif sebuah tugas berat di tengah tekanan tenggat waktu proyek yang terus-menerus.

Salah satu tantangan umum saat membahas risiko keamanan dengan tim teknik adalah adanya kesenjangan antara apa yang dianggap dapat dilakukan dengan apa yang sebenarnya dapat dilakukan. Keinginan untuk meluncurkan produk tepat waktu cenderung membuat tim teknik meremehkan risiko. Menunda proyek karena ancaman yang bersifat hipotetis sering kali dianggap tidak beralasan jika dibandingkan dengan dampak nyata dari kegagalan memenuhi tenggat waktu. Pola pikir ini, ditambah dengan ketidakmampuan untuk mengantisipasi skenario serangan yang kompleks, menyebabkan kerentanan kritis tetap tidak tertangani. Selain itu, proses pengembangan SDV yang serba cepat dapat memicu kelelahan ekstrem (*burnout*) di kalangan pengembang. Mereka mungkin menganggap tugas-tugas keamanan sebagai hambatan tambahan yang tidak perlu, yang pada akhirnya menyebabkan kelelahan mental dan implementasi keamanan yang tidak optimal.

Tantangan Teknis Arsitektur Terpusat

Pergeseran dari arsitektur berbasis domain ke arsitektur terpusat menghilangkan pemisahan fisik yang sebelumnya membagi sistem otomotif. Jika sebelumnya penyerang harus menghadapi berbagai penghalang perangkat keras, kini mereka berpotensi mengeksploitasi

satu kesalahan konfigurasi saja untuk mengakses fungsi-fungsi krusial. Virtualisasi memang bertujuan untuk mereplikasi isolasi fisik tersebut, namun efektivitasnya sangat bergantung pada konfigurasi yang sempurna dan implementasi perangkat keras yang tanpa cela hal yang dalam praktiknya sering kali terbukti gagal.

Tantangan teknis unik lainnya adalah kebutuhan untuk menghasilkan log dan data debug dalam jumlah besar, yang dapat memberikan informasi berharga bagi pihak penyerang jika tidak diamankan dengan baik. Di saat yang sama, produsen kendaraan (*Original Equipment Manufacturers* atau OEM) menghadapi kendala logistik dalam mendistribusikan patch keamanan ke jutaan kendaraan. SDV juga sangat bergantung pada layanan cloud untuk augmentasi kecerdasan buatan (AI) dan pemrosesan data, yang menciptakan vektor serangan tambahan. Pelanggaran keamanan pada infrastruktur cloud dapat merembet hingga mengompromikan sistem di dalam kendaraan, yang menegaskan betapa eratnya keterkaitan antara keamanan kendaraan dan keamanan TI perusahaan.

Lanskap Penyerang

Arsitektur SDV yang menyerupai sistem TI perusahaan menurunkan hambatan bagi pihak penyerang untuk melancarkan aksinya. Perangkat eksploitasi (*exploit toolkits*) yang dikembangkan untuk lingkungan TI tradisional sering kali dapat diadaptasi untuk SDV dengan upaya minimal. Selain itu, meningkatnya standardisasi di antara berbagai OEM membuat serangan yang berhasil dapat diperluas cakupannya dengan lebih mudah. Minimnya eksploitasi yang menyita perhatian publik dapat menciptakan rasa aman yang semu, sehingga menghambat investasi penting dalam langkah-langkah pertahanan.

Meningkatkan Keamanan Otomotif

Mengingat tantangan-tantangan yang telah disebutkan di atas, diperlukan pendekatan holistik untuk meningkatkan standar keamanan pendekatan yang memadukan perbaikan proses, pengembangan sumber daya manusia, dan inovasi teknologi. Pertama, kita perlu menyederhanakan siklus hidup rekayasa yang aman (*secure engineering lifecycles*). Tugas-tugas keamanan harus diintegrasikan secara mulus ke dalam siklus pengembangan yang berjalan cepat. Hal ini mencakup penciptaan kerangka kerja adaptif yang mampu mendeteksi kerentanan sejak dini dan mengelola risiko residual secara sistematis. Otomatisasi dapat membantu menjaga kecepatan pengembangan sekaligus memastikan kepatuhan terhadap standar keamanan yang ketat.

Kedua, ada kebutuhan untuk memberdayakan sumber daya manusia. Peningkatan keterampilan para insinyur melalui pelatihan keamanan siber yang berkelanjutan sangatlah krusial. Budaya yang menghargai penilaian risiko komprehensif dan pemodelan ancaman berbasis skenario dapat membantu mengatasi kecenderungan terhadap "keamanan di atas kertas" yang bersifat dangkal. Ketiga, kita harus beralih ke arsitektur *Zero Trust* (kepercayaan nol). Contohnya adalah memperkuat platform komputasi berkinerja tinggi (*High-Performance Computing* atau HPC) dengan virtualisasi yang tangguh, perlindungan kriptografis, dan mekanisme *secure boot*. Contoh lainnya adalah penerapan isolasi sumber daya yang ketat untuk memastikan bahwa kerentanan di satu domain tidak merembet ke domain lainnya. Terakhir, kita harus mengadopsi alur kerja kecerdasan buatan yang bersifat agen (*agentic AI*

workflows) di sepanjang siklus hidup rekayasa yang aman hal yang akan menjadi fokus pembahasan kita selanjutnya.

10.4 INTEGRASI AI OFENSIF DAN DEFENSIF DALAM KEAMANAN SIBER OTOMOTIF

Di awal diskusi ini, kami telah memperingatkan tentang bahaya mengadopsi postur keamanan yang lazim di industri lain, di mana model operasionalnya telah merosot menjadi apa yang sering disebut sebagai pendekatan "tambal lalu berdoa" (*patch and pray*). Dalam lingkungan tersebut, perangkat lunak dirilis dengan asumsi bahwa kerentanan pasti akan ditemukan dan diperbaiki kemudian melalui tambalan (*patch*). Meskipun model ini mungkin dapat ditoleransi dalam sistem TI konsumen, model tersebut sama sekali tidak cocok untuk sistem yang berkaitan dengan keselamatan, seperti yang terdapat pada kendaraan modern. Jika Anda ragu dengan pernyataan tersebut, tanyakanlah pada diri sendiri: seberapa cepat vendor perangkat lunak perusahaan dapat mengeluarkan tambalan perangkat lunak dibandingkan dengan produsen kendaraan (OEM)? Bahkan jika kita berhasil mengatasi masalah kecepatan pemberian tambalan, fakta bahwa kerentanan dalam sistem otomotif dapat berdampak pada nyawa dan keselamatan manusia membuat kita tidak bisa menoleransi kegagalan keamanan sedikit pun.

Untuk menghindari terjerumus ke dalam model tersebut, sebelumnya dalam buku ini kami telah menyarankan pendekatan keamanan siber yang terkelola, di mana keamanan diperlakukan sebagai disiplin rekayasa sistematis yang terintegrasi di sepanjang siklus hidup produk. Namun, praktik keamanan yang bersifat defensif saja tidaklah cukup. Memenangkan perlombaan "kucing-kucingan" dengan pihak penyerang membutuhkan lebih dari sekadar membangun pertahanan. Hal ini menuntut antisipasi terhadap bagaimana pertahanan tersebut dapat ditembus.

Secara praktis, ini berarti berinvestasi tidak hanya pada kendali pertahanan, tetapi juga pada metode dan perangkat keamanan ofensif. Tujuannya bukan sekadar meniru penyerang karena rasa ingin tahu, melainkan untuk memahami kapabilitas mereka sebelum kapabilitas tersebut digunakan untuk menyerang sistem di dunia nyata. Hal ini memerlukan sinergi antara domain keamanan defensif dan ofensif. Keamanan defensif berfokus pada arsitektur, isolasi, dan pengurangan risiko. Keamanan ofensif berfokus pada penemuan kelemahan dan penyusunan rantai serangan. Ketika kedua disiplin ini beroperasi secara terpisah, masing-masing akan memiliki titik buta (*blind spot*). Sebaliknya, ketika keduanya beroperasi bersama, satu sama lain akan saling memperkuat.

Terlebih lagi, munculnya keamanan ofensif berbasis kecerdasan buatan (AI) membuat kolaborasi ini menjadi jauh lebih mendesak. Seperti yang akan kita bahas di bagian ini, AI secara signifikan menurunkan biaya untuk melakukan kampanye keamanan ofensif. Tugas-tugas yang sebelumnya membutuhkan waktu berminggu-minggu oleh para ahli, kini dapat diselesaikan dalam hitungan jam. Hambatan untuk mengembangkan eksploitasi semakin rendah, dan kemungkinan terjadinya serangan berskala besar terhadap sistem otomotif menjadi jauh lebih nyata. Secara paradoks, pergeseran teknologi ini justru menciptakan peluang bagi pihak pertahanan. Jika AI dapat digunakan untuk mempercepat penemuan celah

keamanan, AI juga dapat dimanfaatkan untuk mengurangi jumlah kelemahan yang bisa dieksploitasi secara sistematis sebelum sistem masuk ke tahap produksi. Untuk pertama kalinya dalam sejarah, gagasan untuk mencapai kondisi hampir nol kelemahan yang dapat dieksploitasi telah menjadi tujuan rekayasa yang realistis.

Untuk mencapai hasil ini, diperlukan integrasi bantuan AI di sepanjang siklus hidup rekayasa yang aman (*secure engineering lifecycle*). Namun, alur kerja yang dibantu oleh AI bukanlah pengganti bagi arsitektur keamanan yang kokoh. Arsitektur yang memprioritaskan isolasi, pembatasan (*containment*), dan batasan kepercayaan (*trust boundaries*) yang jelas tetap menjadi fondasi keamanan sistem. Faktanya, menerapkan alur kerja berbasis agen (*agentic workflows*) pada arsitektur yang sejak awal tidak aman hanyalah cara mahal untuk menghasilkan "sampah" (AI slop), karena kita gagal memperbaiki keputusan desain yang buruk.

Dalam situasi seperti ini, organisasi justru membuang sumber daya komputasi dan token model untuk mencoba menambal kerentanan dalam sistem yang seharusnya tidak pernah diterapkan dalam bentuk tersebut sejak awal. Oleh karena itu, alur kerja berbasis AI harus diterapkan secara sistematis sejak dini, dimulai dari analisis fitur, kemudian berlanjut ke tinjauan arsitektur, analisis implementasi, hingga pengujian penetrasi. Secara konseptual, hubungan antara domain defensif dan ofensif dapat divisualisasikan sebagai siklus umpan balik yang berkelanjutan:

```
System Definition → TARA → Security Architecture → Implementation
        ↓                                          ↑
  Offensive Testing (Pentest) ← Vulnerability Discovery
```

Dalam model ini, output TARA mengidentifikasi aset bernilai tinggi dan tujuan keamanan; pengujian penetrasi (*penetration testing*) berfokus pada fitur serta komponen yang terkait dengan aset-aset krusial tersebut; validasi konsep teknis mendeteksi kelemahan arsitektur dan kerentanan yang dapat dieksploitasi; kemudian temuan-temuan yang diperoleh digunakan untuk menyempurnakan desain dan menyusun strategi mitigasi. Hasilnya adalah sebuah proses rekayasa keamanan yang bersifat siklikal (*closed-loop*), di mana wawasan dari perspektif ofensif secara berkelanjutan memperkuat arsitektur defensif. Untuk memahami mengapa model terintegrasi ini diperlukan, kita harus terlebih dahulu menelaah bagaimana dinamika ekonomi dalam pengembangan eksploitasi terus mengalami perubahan.

Dari Serangan yang Terbatas oleh Manusia hingga Serangan Otonom

Selama bertahun-tahun, pengembangan eksploitasi dalam sistem otomotif sangat dibatasi oleh kapasitas manusia. Keamanan ofensif membutuhkan kesabaran, intuisi, dan spesialisasi teknis yang mendalam. Seperti yang telah kita lihat di bab-bab sebelumnya, kampanye tipikal seringkali melibatkan hal-hal berikut:

- Rekayasa balik citra *firmware* yang telah dihilangkan simbol dan dokumentasinya
- Menganalisis secara manual protokol transport CAN, LIN, Ethernet, atau protokol milik perusahaan

- Membangun rangkaian *fuzzing* buatan tangan berdasarkan dokumentasi dan inferensi parsial
- Membangun rantai eksploitasi multi-tahap di seluruh ECU, *gateway*, dan layanan *backend*
- Menerapkan pengetahuan domain yang mencakup batasan tertanam, arsitektur E/E otomotif, dan batasan keamanan perangkat keras

Masing-masing aktivitas ini menimbulkan beban kognitif nonlinier. Berkembang dari penemuan kerentanan hingga eksploitasi yang dipersenjatai membutuhkan waktu dan keahlian yang signifikan. Hambatan utama adalah fase penemuan, di mana para peneliti harus mengungkap kelemahan yang dapat dieksploitasi sebelum rantai serangan apa pun dapat dibangun. Agen ofensif AI secara fundamental mengubah persamaan ini, seperti yang ditunjukkan pada Tabel 10.1:

Tabel 10.1 Perbedaan upaya keamanan ofensif antara model yang dijalankan manusia dan model yang diperkuat oleh AI

Aktivitas	**Model upaya manusia**	**Model yang diperkuat AI**
Analisis biner	Minggu	Jam
Inferensi protokol	Rekayasa balik manual	Pengelompokan otomatis dan penalaran simbolis
Pembuatan pengujian fuzz	Berbasis heuristik	Pembuatan berbasis cakupan melalui perencanaan LLM
Sintesis eksploit	Pembuatan payload secara manual	Penyusunan payload dengan bantuan
Penyusunan rantai serangan	Pemodelan kognitif manusia	Perencanaan multi-agen

Peralihan dari model yang dijalankan manusia ke model yang diperkuat oleh AI tampak halus namun memiliki dampak yang mendalam. Faktor pembatasnya bukan lagi sekadar tenaga ahli, melainkan skala komputasi dan kapabilitas model.

Secara historis, peningkatan skala serangan bersifat linear terhadap keterampilan; tim kecil yang terdiri dari para ahli hanya dapat memeriksa sejumlah target terbatas dalam satu waktu. Dengan dukungan AI, peningkatan skala serangan menjadi berbasis komputasi. Seiring dengan bertambahnya siklus GPU, peningkatan kapabilitas model, dan semakin matangnya alur kerja orkestrasi, *throughput* (kapasitas keluaran) eksploitasi akan berskala mengikuti kapasitas otomatisasi, bukan lagi bergantung pada jumlah anggota tim ofensif. Ini bukan sekadar peningkatan bertahap, melainkan percepatan struktural dalam kapabilitas ofensif. Untuk memahami bagaimana transformasi ini terjadi dalam praktiknya, kita perlu mengklasifikasikan jenis-jenis sistem AI yang mendorong perubahan ini.

Taksonomi Kapabilitas Ofensif Berbasis AI

Untuk menganalisis ancaman berbasis AI secara cermat, kita mengelompokkan sistem ofensif ke dalam tiga kategori. Perbedaan ini penting secara operasional karena setiap kategori menghadirkan jenis risiko yang berbeda bagi program keamanan otomotif.

Penyerang Manusia dengan Bantuan LLM

Penggunaan awal *Large Language Model* (LLM) dalam riset keamanan termasuk dalam kategori ini. Dalam model ini, operator manusia tetap memegang kendali atas kampanye pengujian penetrasi (*penetration testing*), namun menggunakan LLM sebagai pengganda kekuatan (*force multiplier*). Tugas-tugas yang umum dilakukan meliputi:

- Menjelaskan kode yang kompleks dan mengidentifikasi potensi kerentanan dalam basis kode (*codebase*) yang besar
- Membantu proses transformasi dari CVE menjadi eksploitasi
- Menafsirkan binary hasil dekompilasi untuk mempercepat upaya rekayasa balik (*reverse engineering*)
- Menghasilkan *fuzzing* harness (perangkat untuk pengujian *fuzzing*) berdasarkan deskripsi antarmuka yang parsial

Dalam model ini, manusia tetap terlibat dalam prosesnya (*human-in-the-loop*). Tujuannya adalah untuk meningkatkan *throughput* dan menurunkan ambang batas keahlian yang diperlukan.

Sebagai contoh, alih-alih mengharuskan penguji penetrasi untuk menafsirkan bahasa rakitan (*assembly language*) secara manual, penguji dapat meminta LLM untuk menjelaskan logika dari fungsi yang telah dibongkar (*disassembled*). Demikian pula, alih-alih menugaskan tim untuk meninjau basis kode yang besar seperti keseluruhan rantai boot (*boot chain*) secara manual, penguji dapat memberikan perintah (*prompt*) kepada model untuk mencari pola yang sesuai dengan kelas kerentanan tertentu. Dampak utamanya adalah berkurangnya hambatan untuk memulai (*barrier to entry*). Tugas-tugas yang sebelumnya memerlukan insinyur senior kini dapat dicoba oleh praktisi tingkat menengah dengan dukungan AI. Batas atas keahlian (*skill ceiling*) tetap penting, namun standar keahlian minimum yang diperlukan (*skill floor*) menjadi jauh lebih rendah.

Alur Kerja Penemuan Kerentanan Secara Otonom

Seiring dengan berkembangnya eksperimen dalam kolaborasi berbasis AI, para peneliti segera menyadari bahwa keterlibatan manusia sering kali menjadi hambatan utama (*bottleneck*). Sistem-sistem awal menghasilkan temuan dalam jumlah besar, yang banyak di antaranya merupakan positif palsu (*false positive*) sehingga memerlukan peninjauan manual. Untuk mengurangi hambatan ini, diperlukan pergeseran dari sekadar bantuan menuju otomatisasi penuh. Alur kerja penemuan kerentanan secara otonom beroperasi secara berkelanjutan dan biasanya dipicu oleh peristiwa tertentu, seperti pengunggahan kode (*code commit*) baru atau siklus pengujian rutin harian. Komponen-komponen yang umum digunakan meliputi:

- Mesin analisis statis yang mengevaluasi pelanggaran praktik penulisan kode yang aman (*secure coding*) serta mengusulkan perbaikan (*patch*)
- Pembuatan kerangka pengujian (*harness*) secara otomatis untuk *fuzz testing*
- Sistem *fuzzing* berbasis cakupan (*coverage-guided fuzzing*) yang mengeksplorasi jalur kode secara otomatis

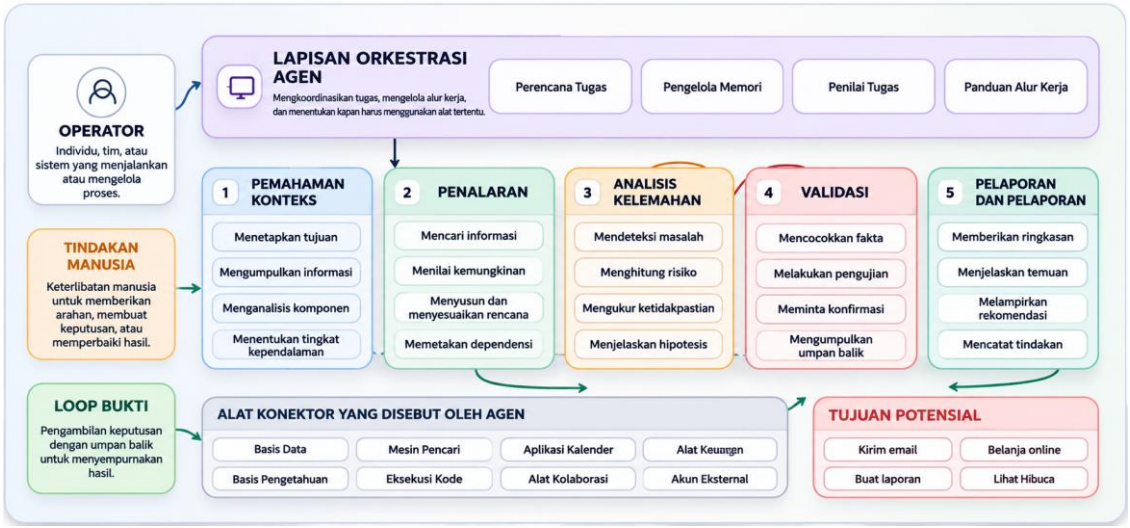
- Otomatisasi peninjauan crash yang mengelompokkan dan memprioritaskan kesalahan (*faults*)
- Klasifikasi akar penyebab (*root cause*) menggunakan pemetaan taksonomi CWE

Jika digunakan untuk tujuan ofensif, alur kerja ini mengubah penemuan kerentanan menjadi sebuah masalah komputasi. Alih-alih harus memilih kelemahan mana yang akan diselidiki, penyerang dapat memeriksa semuanya, selama tersedia sumber daya komputasi yang memadai.

Implikasi praktisnya adalah penemuan kerentanan menjadi sesuatu yang terjadi di mana-mana, bukan lagi sekadar selektif. Sistem dipindai secara terus-menerus, bukan hanya saat dilakukan pengujian penetrasi (*penetration testing*) secara berkala. Hal ini secara mendasar mengubah asumsi pihak pertahanan mengenai kerangka waktu yang tersedia.

Sistem Ofensif Berbasis Multi-Agen

Kategori ketiga ini mewakili perubahan yang paling signifikan dan dianggap sebagai teknologi mutakhir (*state of the art*) pada saat buku ini ditulis. Sistem ini mengoordinasikan berbagai agen khusus yang meniru cara kerja tim pengujian penetrasi lengkap. Dengan kemampuan untuk menggunakan berbagai perangkat (*tool calling*) dan keterampilan khusus, pendekatan multi-agen ini dapat mengotomatisasi seluruh tahapan mulai dari penemuan kerentanan, penyusunan rantai serangan (*attack chain*), hingga pengujian eksploitasi untuk menghasilkan eksploit yang siap digunakan sepenuhnya. Gambar 10.3 menampilkan contoh alur kerja berbasis agen yang merujuk pada temuan kampanye spionase berbasis AI oleh Anthropic:



Gambar 10.3 Alur keamanan ofensif yang diorkestrasi oleh AI

Sistem ofensif berbasis multi-agen ini beroperasi sebagai kampanye yang terorkestrasi, bukan sekadar upaya eksploitasi tunggal. Proses dimulai dengan operator manusia yang menentukan target dan tujuan. Selanjutnya, koordinator AI memecah misi tersebut ke dalam fase-fase yang bersifat iteratif. Pada fase pengintaian (*reconnaissance*), agen-agen khusus menjalankan alat pemindaian, pencarian, pengambilan data, dan analisis kode untuk

mengumpulkan intelijen terstruktur. Temuan-temuan tersebut dikumpulkan, diringkas, dan dimasukkan kembali ke dalam siklus perencanaan. Pada fase berikutnya, agen eksploitasi melakukan eksekusi eksploitasi terkendali dan memvalidasi callback terhadap layanan target, sementara agen lain menyempurnakan jalur serangan berdasarkan perilaku sistem yang teramati.

Setelah akses diperoleh, agen pengintaian internal melakukan pergerakan lateral (*lateral movement*), mengumpulkan kredensial, serta memetakan batasan kepercayaan (*trust boundaries*) yang baru. Sistem terus mengevaluasi ulang grafik serangan, mengarahkan penggunaan alat lebih lanjut, dan meningkatkan hak akses (*privilege escalation*) jika memungkinkan. Pada akhirnya, alur kerja ini berujung pada tindakan yang berorientasi pada tujuan, seperti eksfiltrasi data secara terkendali. Aspek arsitektural yang krusial di sini adalah bahwa penalaran, penggunaan alat, eksploitasi, dan umpan balik didistribusikan ke berbagai agen yang saling bekerja sama, sehingga memungkinkan pembentukan rantai serangan menyeluruh (*end-to-end attack chain*) yang beradaptasi secara dinamis terhadap respons sistem.

Ini adalah titik balik yang signifikan. Sistem AI kini tidak lagi terbatas pada identifikasi kerentanan kode, seperti cacat kerusakan memori (*memory corruption*). Sistem ini mampu menalar batasan kepercayaan, alur pembaruan (*update pipelines*), layanan bersama, hierarki hak akses, serta jalur komunikasi lintas domain. Hal ini mengubah tantangan yang sebelumnya bergantung pada waktu dan sumber daya manusia menjadi tantangan yang bergantung pada sumber daya komputasi dan biaya pembuatan token (*token generation costs*). Namun, berbeda dengan manusia, alur kerja AI dapat ditingkatkan skalanya secara efisien dan sesuai permintaan.

Manajemen Kerentanan Gagal Untuk Berskala

Model pertahanan tradisional mengikuti urutan reaktif:

1. Pemindaian (*Scan*)
2. Triase (*Triage*)
3. Penambalan (*Patch*)
4. Pemantauan CVE

Pada arsitektur ECU terdistribusi warisan (*legacy*) dengan kompleksitas perangkat lunak yang terbatas, model ini masih cukup dapat dipertahankan. Namun, pada platform komputasi berkinerja tinggi (HPC) yang terpusat, model ini mulai gagal. Beberapa faktor berikut berkontribusi terhadap kegagalan tersebut:

- Jumlah layanan yang sangat besar di dalam node komputasi terpusat
- Banyaknya pustaka pihak ketiga dan dependensi yang tidak transparan (*opaque dependencies*)
- Berkas biner dari pemasok yang tidak dapat dimodifikasi dengan mudah
- Domain keselamatan (*safety*) dan keamanan (*security*) yang bercampur dalam infrastruktur yang sama
- Alur penerapan melalui udara (OTA) yang mempersulit koordinasi pembaruan

Seiring dengan AI yang menurunkan biaya penemuan kerentanan oleh pihak penyerang, alur kerja triase dan penambalan manual oleh pihak pertahanan tidak dapat mengimbangi kecepatan tersebut. Hasilnya bukan sekadar penumpukan kerentanan yang belum ditangani, melainkan ketidaksesuaian kecepatan (*velocity mismatch*) yang sistematis antara otomatisasi penyerang dan respons pihak pertahanan. Hal ini memunculkan pertanyaan berikutnya: "Jika manajemen kerentanan tidak dapat berskala, lalu apa yang bisa?"

Eliminasi Kerentanan: Definisi yang Berorientasi Pada Pengurangan

Eliminasi kerentanan bukan berarti sistem akan sepenuhnya bebas dari cacat. Sebaliknya, kami mendefinisikan eliminasi sebagai pengurangan sistematis terhadap kepadatan kelemahan yang dapat dieksploitasi, melalui agen AI yang terintegrasi dalam siklus hidup pengembangan yang mampu mencegah berbagai jenis kerentanan bahkan sebelum sistem diterapkan.

Tujuannya bukanlah kesempurnaan, melainkan pengurangan yang terukur terhadap seluruh kategori kerentanan. Contohnya termasuk menghilangkan kategori kerusakan memori (*memory corruption*) melalui penggunaan subset bahasa pemrograman yang lebih aman atau runtime yang menjamin keamanan memori; menghilangkan cacat deserialisasi melalui pola validasi skema yang ketat; serta mencegah manipulasi pada rantai pembaruan melalui jaminan integritas yang dapat dibuktikan. Pergeseran strategis ini beralih dari penambalan reaktif menuju pencegahan struktural, karena pendekatan ini memiliki karakteristik skala yang berbeda dibandingkan dengan perbaikan (remediasi) biasa.

Model-V yang Diperkuat Oleh AI

Setelah memahami implikasi dari keamanan ofensif berbasis AI, kita harus meninjau sisi pertahanannya. Sama seperti agen AI yang mengotomatiskan penemuan kerentanan oleh penyerang, teknologi ini juga dapat memperkuat rekayasa pertahanan. Tujuannya sederhana: mengurangi jumlah kelemahan yang dapat dieksploitasi hingga sedekat mungkin dengan nol, sehingga menghilangkan keunggulan yang dimiliki oleh sistem serangan otomatis. Untuk mencapai tujuan ini, agen AI harus diintegrasikan ke dalam seluruh tahapan Model-V otomotif, alih-alih hanya diterapkan pada tahap akhir pengembangan. Berikut adalah lima kasus penggunaan yang menggambarkan bagaimana agen AI meningkatkan kapabilitas keamanan defensif kita.

Pemodelan Ancaman di Tingkat Fitur

Pada tingkat fitur, agen AI dapat membantu insinyur keamanan dengan cara terlebih dahulu menyerap spesifikasi fitur serta membangun konteks berdasarkan persyaratan yang ada, diagram arsitektur, dan dokumentasi produk, sebagaimana ditunjukkan dalam Gambar 10.4.

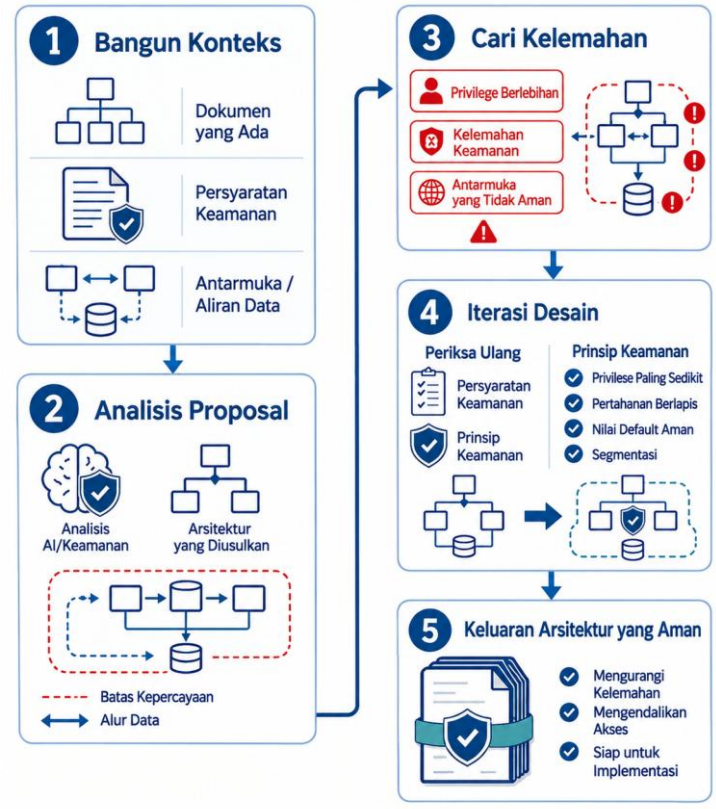


Gambar 10.4 Alur agen pembangkit ancaman pada tingkat fitur

Dengan menggunakan konteks ini, agen mengidentifikasi batasan kepercayaan (*trust boundaries*) yang diasumsikan serta aset-aset terkait, kemudian menghasilkan kandidat ancaman seperti akses tidak sah, penyamaran (*spoofing*), penolakan layanan (*denial of service*), atau paparan data. Selanjutnya, agen dapat menentukan langkah mitigasi dari katalog keamanan yang telah disetujui seperti autentikasi, otorisasi, komunikasi aman, validasi input, dan kontrol akses serta memetakan mitigasi tersebut ke dalam persyaratan dan kontrol keamanan yang sudah ada. Jika ditemukan kesenjangan, agen akan mengusulkan pembaruan pada persyaratan atau arsitektur, sehingga fitur tersebut dapat masuk ke tahap rekayasa hilir dengan seperangkat persyaratan dan kontrol keamanan yang jelas.

Tinjauan Keamanan Tingkat Arsitektur

Pada tingkat arsitektur, agen AI dapat membantu insinyur keamanan mengevaluasi apakah desain yang diusulkan mempertahankan batasan kepercayaan dan properti keamanan yang dimaksudkan sebelum implementasi dimulai. Seperti yang ditunjukkan pada Gambar 10.5, agen pertama-tama membangun konteks dari dokumen desain yang ada, persyaratan keamanan, antarmuka, dan aliran data, kemudian menganalisis arsitektur yang diusulkan terhadap prinsip-prinsip keamanan yang diketahui. Analisis ini dapat mengidentifikasi kelemahan seperti hak istimewa yang berlebihan, isolasi yang lemah, antarmuka yang tidak tepercaya, penyalahgunaan memori bersama, pelanggaran batasan hak istimewa, dan celah dalam rantai boot atau pembaruan yang aman:

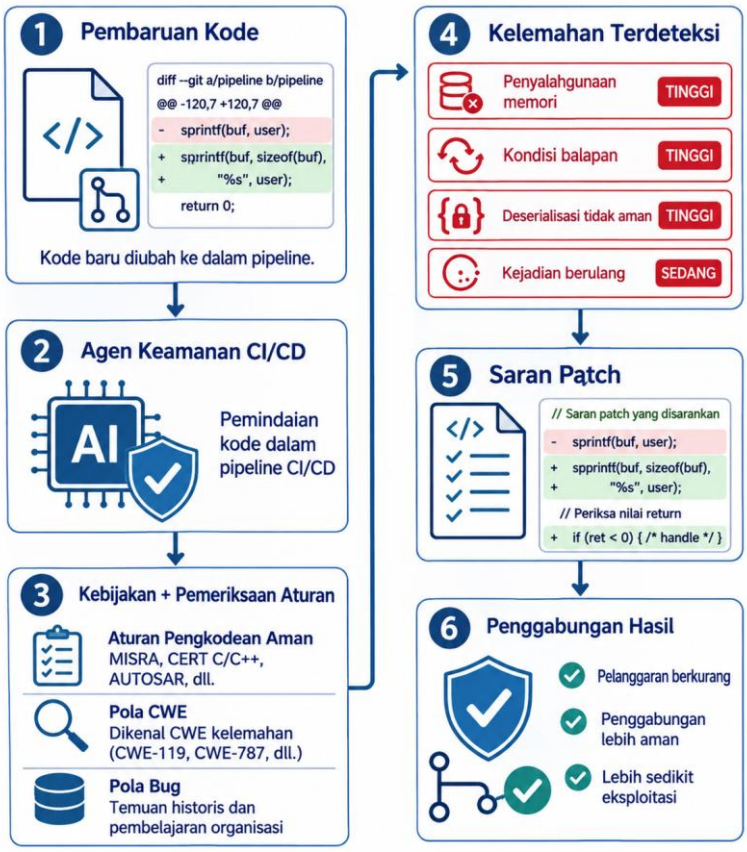


Gambar 10.5 Agen pendeteksi kelemahan arsitektur

Agen tersebut kemudian memeriksa desain terhadap berbagai persyaratan dan prinsip seperti hak akses minimum (*least privilege*), pertahanan berlapis (*defense in depth*), pengaturan bawaan yang aman (*secure defaults*), dan kompartementalisasi serta mengusulkan perubahan arsitektur jika diperlukan. Hasilnya adalah arsitektur yang lebih tangguh dengan kelemahan yang berkurang, kontrol yang lebih selaras, dan landasan yang lebih kuat untuk tahap implementasi selanjutnya.

Analisis Pada Tingkat Implementasi

Pada tingkat implementasi, agen AI dapat beroperasi secara berkelanjutan di dalam alur CI/CD, sehingga setiap perubahan kode yang baru dapat langsung dianalisis begitu masuk ke dalam proses build. Sebagaimana ditunjukkan dalam Gambar 10.5, agen tersebut memeriksa setiap pembaruan berdasarkan aturan penulisan kode yang aman seperti standar MISRA dan CERT C/C++ serta pola CWE, pola bug sebelumnya, dan temuan spesifik organisasi dari tinjauan terdahulu. Hal ini memungkinkan agen untuk mendeteksi berbagai kelemahan, seperti penyalahgunaan memori, *race condition*, deserialisasi yang tidak aman, pola kerentanan yang berulang, dan berbagai *exploit primitive* lainnya yang sudah diketahui, sebelum kode tersebut mencapai tahap build produksi.

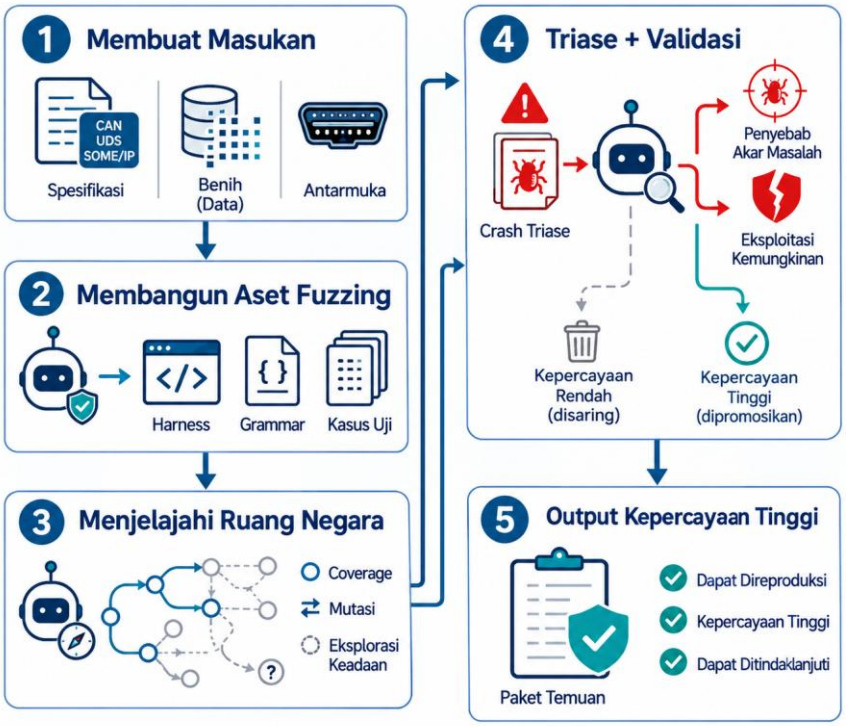


Gambar 10.6 Alur kerja pencari kelemahan dalam implementasi kode

Alih-alih hanya melaporkan pelanggaran, agen juga dapat memberikan saran perbaikan (*patch*) secara langsung, sehingga membantu pengembang memperbaiki kode yang tidak aman saat konteksnya masih segar. Hasilnya adalah latensi deteksi yang lebih rendah, berkurangnya pengulangan jenis kerentanan yang sama, keputusan penggabungan kode (*merge*) yang lebih aman, serta penurunan risiko masuknya metode eksploitasi yang sudah dikenal ke dalam perangkat lunak ECU yang dirilis.

Fuzzing Otomatis dan Pengujian White-Box

Agen AI dapat mengurangi salah satu hambatan terbesar dalam pelaksanaan *fuzzing* yang efektif, yaitu upaya manual yang diperlukan untuk membangun harness, tata bahasa (*grammar*), dan kasus uji yang berguna. Sebagaimana ditunjukkan pada Gambar 10.7, agen dapat memproses spesifikasi protokol dan input awal (*seed inputs*), lalu menghasilkan aset *fuzzing* termasuk harness, tata bahasa protokol, dan kasus uji yang terarah. Agen tersebut juga dapat menyimpulkan state machine protokol, menjelajahi ruang status (*state space*), memprioritaskan celah cakupan pengujian, serta menggunakan eksplorasi simbolik atau eksplorasi batas (*boundary exploration*) untuk mengarahkan eksekusi menuju jalur-jalur yang relevan dengan aspek keamanan.

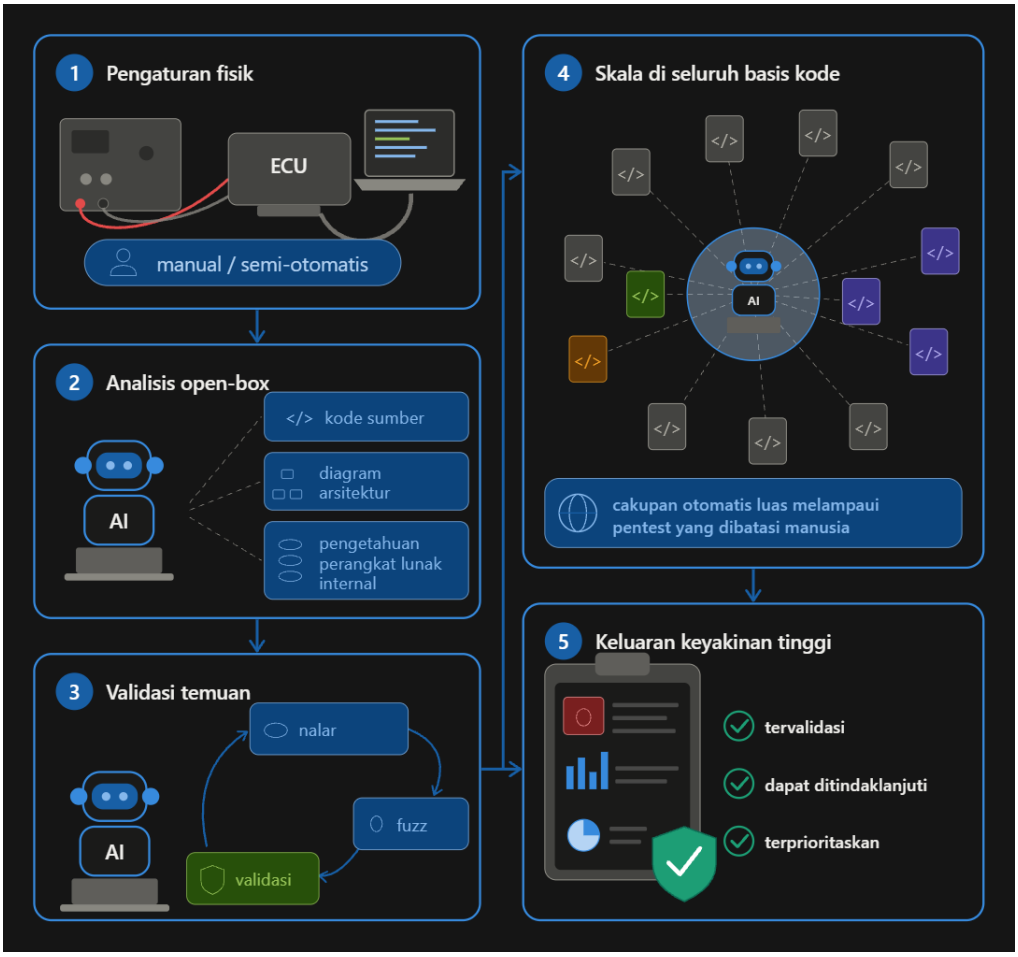


Gambar 10.7 Alur agen *fuzzing* otomatis

Setelah terjadi crash atau perilaku abnormal, agen dapat membantu dalam *triase crash*, analisis akar penyebab, penilaian kelayakan eksploitasi, dan pemeriksaan reprodusibilitas. Temuan dengan tingkat keyakinan rendah dapat disaring, sementara masalah dengan tingkat keyakinan tinggi akan ditindaklanjuti dengan menyertakan bukti pendukung; hal ini mengurangi gangguan (*noise*) dan membantu tim teknis untuk fokus pada kerentanan yang dapat ditangani. Dalam model ini, cakupan (*coverage*) menjadi metrik keamanan yang dioptimalkan secara berkelanjutan, bukan sekadar produk sampingan dari pengujian yang dirancang secara manual.

Agen Simulasi Ofensif

Agen AI red-team internal dapat menyimulasikan perilaku lawan dengan menyusun grafik serangan, memodelkan jalur eskalasi hak akses, mengevaluasi rantai serangan, serta menguji komponen target menggunakan input yang bersifat menyerang. Sebagaimana ditunjukkan pada Gambar 10.8, tidak semua aspek dalam pengujian penetrasi (*penetration testing*) dapat diotomatisasi sepenuhnya, terutama aktivitas yang memerlukan pengaturan fisik, pengabelan ECU, penggunaan instrumen laboratorium, atau penilaian terkait keselamatan manusia.



Gambar 10.8 Alur otomatis agen pengujian penetrasi (*pentesting*)

Namun, pengujian penetrasi *open-box* sangat cocok untuk otomatisasi karena agen dapat menganalisis kode sumber, diagram arsitektur, dan pengetahuan internal perangkat lunak dalam skala besar. Agen mampu menalar kelemahan yang ditemukan, membuat fuzz test atau pemeriksaan validasi, serta hanya menyajikan temuan yang dapat direproduksi, dapat ditindaklanjuti, dan telah diprioritaskan. Selain itu, agen pengujian penetrasi dapat meniru serangan jaringan jika diberikan akses ke perangkat jaringan. Hal ini memperluas cakupan pengujian keamanan melampaui bagian basis kode yang dapat diperiksa secara manual oleh tim penguji manusia.

Meskipun demikian, pengujian penetrasi yang diperkuat dengan kecerdasan buatan (AI) sebaiknya dipandang sebagai lapisan jaminan (*assurance layer*), bukan sebagai strategi pertahanan utama. Agen tidak akan menemukan setiap kerentanan yang ada, dan ketergantungan yang berlebihan pada pengujian penetrasi di tahap akhir dapat memicu siklus perbaikan (*patching*) yang mengganggu operasional. Tujuannya adalah menerapkan praktik keamanan yang kuat sejak awal (*upstream*), kemudian menggunakan simulasi serangan sebagai langkah antisipasi untuk menutup celah yang mungkin terlewatkan oleh sistem otomatis, sebelum celah tersebut dieksploitasi oleh pihak penyerang.

10.5 ARAH MASA DEPAN KEAMANAN SIBER OTOMOTIF

Dalam bab terakhir ini, kami memetakan arah ke depan untuk menyikapi realitas baru dalam keamanan siber otomotif, di mana kita tidak bisa lagi sekadar mengandalkan pertahanan yang bersifat reaktif. Model lama yang menemukan kerentanan di tahap akhir lalu memperbaikinya melalui patch sejak awal memang rapuh; namun, dalam konteks kendaraan berbasis perangkat lunak (*Software-Defined Vehicles* atau SDV) dan arsitektur komputasi berkinerja tinggi (*High-Performance Computing* atau HPC) yang terpusat, model tersebut menjadi sangat tidak berkelanjutan. Kendaraan kini menyerupai platform komputasi terdistribusi yang terhubung dengan ekosistem cloud, menjalankan jutaan baris kode di berbagai komponen yang saling terhubung erat. Dalam lingkungan seperti ini, asumsi bahwa kegagalan keamanan dapat diperbaiki begitu saja setelah kendaraan digunakan (*deployed*) tidak lagi sejalan dengan ekspektasi keselamatan maupun realitas operasional.

Kami telah membahas bahwa pendekatan yang bersifat *ad-hoc* (tidak terencana secara sistematis) gagal bukan karena kurangnya upaya dari organisasi, melainkan karena ketiadaan struktur. Tanpa metode sistematis untuk mengidentifikasi risiko, menetapkan kontrol, serta memverifikasi efektivitasnya, keputusan keamanan menjadi terfragmentasi dan reaktif. Contoh-contoh yang diulas dalam bab ini menunjukkan bagaimana fragmentasi tersebut memicu asumsi kepercayaan implisit, analisis ancaman yang tidak lengkap, kerentanan rantai pasok yang tidak terkelola, serta keterlambatan dalam menemukan celah keamanan. Ketika kelemahan-kelemahan ini terakumulasi, hasilnya adalah sistem yang tampak aman di atas kertas (dalam dokumentasi), namun sebenarnya sangat rapuh dalam praktiknya.

Wawasan mengenai pendekatan keamanan yang terkelola (*managed security approach*) memberikan kerangka perbaikan yang diperlukan. Dengan mengintegrasikan keamanan siber ke dalam seluruh siklus hidup rekayasa melalui Sistem Manajemen Keamanan Siber (*Cybersecurity Management System* atau CSMS) yang selaras dengan standar ISO/SAE 21434, organisasi dapat beralih dari penanganan insiden yang reaktif menuju manajemen risiko yang terkendali. TARA (*Threat Analysis and Risk Assessment*), konsep keamanan siber, prinsip desain yang aman, praktik implementasi yang disiplin, serta verifikasi dan validasi yang terstruktur, menciptakan rangkaian bukti yang menghubungkan identifikasi risiko dengan keputusan rekayasa yang konkret. Perspektif siklus hidup ini mencakup tahapan pengembangan, produksi, penerapan, operasional, hingga penonaktifan produk, sehingga memastikan bahwa asumsi keamanan tetap valid sepanjang masa pakai produk tersebut.

Bab ini juga menekankan bahwa keamanan siber bukanlah disiplin ilmu tunggal, melainkan sebuah ekosistem yang terdiri dari berbagai aktivitas yang saling bekerja sama. Arsitektur pertahanan, praktik pengembangan yang aman, pemantauan kerentanan, pengujian penetrasi (*penetration testing*), dan pemantauan operasional harus saling memperkuat, bukan berjalan secara terpisah. Metode keamanan ofensif, jika diterapkan secara konstruktif dalam kerangka kerja yang terkelola, dapat menjadi mekanisme validasi yang ampuh untuk menguji asumsi-asumsi yang dibuat selama tahap desain. Dengan mengungkap kelemahan lebih dini, organisasi dapat memperkuat sistem mereka sebelum pihak lawan (*adversaries*) sempat mengeksploitasinya.

Langkah ke depan tidak hanya mencakup peningkatan keamanan yang diperlukan untuk menaikkan standar keamanan dalam arsitektur kendaraan modern, tetapi juga menghadapi risiko-risiko baru yang muncul seiring dengan berkembangnya kemampuan ofensif berbasis kecerdasan buatan (AI). Secara paradoks, kita mendapati bahwa pergeseran teknologi yang sama justru memberikan kemampuan baru yang kuat bagi pihak pertahanan. AI generatif dapat mendukung rekayasa yang aman dengan meningkatkan skala proses keamanan, membantu para insinyur dalam menganalisis ancaman, serta menjaga konsistensi di seluruh sistem yang kian kompleks. Agen AI dapat membantu dalam pemodelan ancaman, analisis arsitektur, deteksi kerentanan, dan peninjauan implementasi yang aman, sehingga mengurangi kemungkinan masuknya kelemahan yang dapat dieksploitasi ke dalam sistem produksi. Meskipun AI tidak dapat menggantikan kecerdasan manusia dalam mengantisipasi serangan jenis baru, teknologi ini secara signifikan meningkatkan efisiensi dan memungkinkan pendekatan keamanan siber yang lebih terintegrasi serta proaktif.

Namun, otomatisasi saja tidak dapat menyelesaikan kelemahan rekayasa yang mendasar. Jika sektor otomotif ingin membangun kendaraan berbasis perangkat lunak (*Software-Defined Vehicles*/SDV) yang tangguh, maka diperlukan perbaikan struktural dalam cara sistem direkayasa. Kami juga membahas beberapa arah teknis yang menawarkan jalan menjanjikan ke depan. Salah satu perubahan yang paling berdampak adalah penerapan bahasa pemrograman yang aman terhadap memori (*memory-safe*), seperti Rust atau Ada/SPARK, untuk komponen-komponen yang krusial bagi keamanan. Dengan menghilangkan seluruh kategori kerentanan kerusakan memori (*memory corruption*), permukaan serangan dapat diperkecil bahkan sebelum eksploitasi dimungkinkan. Bab ini juga mengeksplorasi bagaimana alur kerja yang dibantu oleh AI dapat diintegrasikan di sepanjang siklus hidup pengembangan untuk mengurangi kepadatan kelemahan yang dapat dieksploitasi. Saat diterapkan pada analisis kebutuhan, peninjauan arsitektur, pembuatan kode, pengujian, dan penemuan kerentanan, sistem AI dapat membantu para insinyur mengidentifikasi kelemahan jauh lebih awal dalam proses rekayasa. Hasilnya adalah pergeseran dari pengelolaan kerentanan yang bersifat reaktif menuju penghapusan kerentanan secara proaktif.

Dengan demikian, arah ke depan sudah jelas. Mobilitas yang aman tidak akan tercapai hanya dengan merespons kerentanan secara lebih cepat. Keamanan akan tercapai dengan cara mengurangi secara sistematis jumlah kelemahan yang dapat dieksploitasi sebelum kendaraan tersebut digunakan di jalan raya. Keamanan harus dipandang sebagai disiplin yang mencakup seluruh siklus hidup dimulai dari arsitektur, berlanjut melalui implementasi dan validasi, serta terus berlangsung sepanjang masa operasional kendaraan. Dengan mengombinasikan proses rekayasa yang ketat, teknologi keamanan canggih, dukungan AI generatif, dan tenaga kerja yang terlatih dengan baik, industri otomotif dapat membangun sistem yang tangguh, tidak hanya terhadap ancaman saat ini, tetapi juga terhadap berbagai ancaman yang terus berkembang di masa depan.

DAFTAR PUSTAKA

Aiyanyo, I. D., Samuel, H., & Lim, H. (2020). A systematic review of defensive and offensive cybersecurity with machine learning. *Applied Sciences*, *10*(17), 5811.

Barletta, V. S., Caivano, D., Cellammare, G., del Vescovo, S., & Sciacovelli, A. L. (2025). Leveraging Trustworthy AI for Automotive Security in Multi-Domain Operations: Towards a Responsive Human-AI Multi-Domain Task Force for Cyber Social Security. *arXiv preprint arXiv:2507.21145*.

Benzaïd, C., & Taleb, T. (2020). AI for beyond 5G networks: A cyber-security defense or offense enabler?. *IEEE network*, *34*(6), 140-147.

Bharati, S., Podder, P., Mondal, M., & Robel, M. R. A. (2020). Threats and *counter*measures of cyber security in direct and remote vehicle communication systems. *arXiv preprint arXiv:2006.08723*.

Bhukya, C. R., Thakur, P., Mudhivarthi, B. R., & Singh, G. (2023). Cybersecurity in internet of medical vehicles: State-of-the-art analysis, research challenges and future perspectives. *Sensors*, *23*(19), 8107.

Das, P., Asif, M. R. A., Jahan, S., Ahmed, K., Bui, F. M., & Khondoker, R. (2024). STRIDE-based cybersecurity threat modeling, risk assessment and treatment of an in-vehicle infotainment system. *Vehicles*, *6*(3), 1140-1163.

Dash, S., & Azad, C. (2022). Cyber security and threat analysis in autonomous vehicles. *Autonomous Vehicles Volume 1: Using Machine Intelligence*, 161-180.

El-Rewini, Z., Sadatsharan, K., Sugunaraj, N., Selvaraj, D. F., Plathottam, S. J., & Ranganathan, P. (2020). Cybersecurity attacks in vehicular sensors. *IEEE Sensors Journal*, *20*(22), 13752-13767.

Gesteira-Miñarro, R., Gutiérrez, I., Palacios, R., & López, G. (2025). pwnobd: Offensive Cybersecurity Toolkit for Vulnerability Analysis and *Penetration testing* of OBD-II Devices. *IEEE Access*, *13*, 126925-126934.

Girdhar, M., Hong, J., & Moore, J. (2023). Cybersecurity of autonomous vehicles: A systematic literature review of adversarial attacks and defense models. *IEEE Open Journal of Vehicular Technology*, *4*, 417-437.

Jing, P., Cai, Z., Cao, Y., Yu, L., Du, Y., Zhang, W., ... & Wu, S. (2024, May). Revisiting automotive attack surfaces: a practitioners' perspective. In *2024 IEEE Symposium on Security and Privacy (SP)* (pp. 2348-2365). IEEE.

Knebel, S., Schultz, M. D., & Seele, P. (2022). Cyberattacks as "state of exception" reconceptualizing cybersecurity from prevention to surviving and accommodating. *Journal of Information, Communication and Ethics in Society*, *20*(1), 91-109.

Kumar, S., Daily, J., Ahmed, Q., & Arora, A. (2023). Cybersecurity vulnerabilities for off-board commercial vehicle diagnostics. *SAE International Journal of Advances and Current Practices in Mobility*, *5*(2023-01-0040), 2393-2404.

Li, C. P., Zhang, Q., Bian, X. D., & Zhang, S. X. (2023, April). Status quo and prospects of automotive cyber threat intelligence. In *Proceedings of the 2023 International Conference on Frontiers of Artificial Intelligence and Machine Learning* (pp. 214-219).

Mairaj ud din, Q., & Ahmed, Q. (2024, April). Automated TARA Framework for Cybersecurity Compliance of Heavy Duty Vehicles. In *WCX SAE World Congress Experience 288443*.

McLaughlin, K. L. (2023). Defense is the best offense: The evolving role of cybersecurity blue teams and the impact of soar technologies. *Edpacs*, *67*(6), 35-41. Santa Barletta, V., Caivano, D., De Vincentiis, M., Pal, A., & Volpe, F. (2023, October). Automotive knowledge base for supporting vehicle-SOC analysts. In *2023 IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE)* (pp. 960-965). IEEE.

Petrovic, N., Jeddou, O., Fraj, F. B., Zolfaghari, V., Pan, F., Schamschurko, A., & Knoll, A. (2026). CyberLLM: A Multi-Agent LLM Framework for Autonomous Detection and Guarded Response in Automotive Cybersecurity. *arXiv preprint arXiv:2608.06651*.

Qurashi, J. M., Jambi, K., Alsolami, F., Eassa, F. E., Khemakhem, M., & Basuhail, A. (2023). Resilient *counter*measures against cyber-attacks on self-driving car architecture. *IEEE transactions on intelligent transportation systems*, *24*(11), 11514-11543.

Rana, A., Saxena, K., & Anjum, S. (2026, February). Securing the On-Board Unit: a Multi-Layered Framework for Cybersecurity in Electric and Autonomous Vehicles. In *2026 International Conference on Green Technologies and Sustainable Solutions (GTSS)* (pp. 488-495). IEEE.

Roberts, A., Maennel, O., & Snetkov, N. (2021, September). Cybersecurity test range for autonomous vehicle shuttles. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (pp. 239-248). IEEE.

Rumez, M., Grimm, D., Kriesten, R., & Sax, E. (2020). An overview of automotive service-oriented architectures and implications for security *counter*measures. *IEEE access*, *8*, 221852-221870.

Santa Barletta, V., Caivano, D., Cellammare, G., Del Vescovo, S., Morga, M., & Sciacovelli, A. L. (2025, November). Adversarial Training for Battlefield Defensive Trustworthy AI: An Automotive Case Study for Multi-Domain Cyber Social Threat Intelligence. In *2025 IEEE International Workshop on Technologies for Defense and Security (TechDefense)* (pp. 499-504). IEEE.

Siddiqui, F., Khan, R., Tasdemir, S. Y., Hui, H., Sonigara, B., Sezer, S., & McLaughlin, K. (2023, June). Cybersecurity engineering: Bridging the security gaps in advanced automotive systems and ISO/SAE 21434. In *2023 IEEE 97th vehicular technology conference (VTC2023-spring)* (pp. 1-6). IEEE.

Udeshi, M., Shao, M., Xi, H., Rani, N., Milner, K., Putrevu, V. S. C., ... & Shafique, M. (2025). D-CIPHER: Dynamic collaborative intelligent multi-agent system with planner and heterogeneous executors for offensive security. *arXiv preprint arXiv:2502.10931*.

Usman, Y., Gyawali, P. K., Gyawali, S., & Chataut, R. (2024, October). The dark side of AI: Large language models as tools for cyber attacks on vehicle systems. In *2024 IEEE 15th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 169-175). IEEE.

Weaver, A., von Jouanne, A., Sicker, D., & Yokochi, A. (2025). Cybersecurity Dynamometer Testbed: A Review to Advance Vehicle-in-the-Loop Testing of Traditional, Connected and Autonomous Vehicles. *IEEE Open Journal of Vehicular Technology*.

Wu, J., Zhu, B., Tian, E., & Cao, Z. (2026). Offensive-Defensive Integration in Autonomous Vehicles: A Novel Data-Aware Attack and Resilient Control Framework. *IET Control Theory & Applications*, *20*(1), e70153.

Yan, M., Crane, G., Suillivan, D., & Shan, H. (2025). Driving into the Unknown: Investigating and Addressing Security Breaches in Vehicle Infotainment Systems. *Sensors*, *26*(1), 77.

Zachos, M., & Subke, P. (2020, April). Test method for the sae j3138 automotive cyber security standard. In *WCX SAE World Congress Experience 237384*.

Zheng, Y., Zhang, M., Li, X., Chen, X., Zhang, Z., Zhu, J., ... & Wu, G. (2024, August). Automotive security in the digital era: A comprehensive survey of attacks and defenses for keyless entry system. In *International Conference on Computational & Experimental Engineering and Sciences* (pp. 444-467). Cham: Springer Nature Switzerland.

KEAMANAN SIBER OTOMOTIF MODERN:

Serangan ECU (Electronic Control Unit) Kendaraan via IoT

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

BIODATA PENULIS



Dr. Joseph Teguh Santoso, M.Kom memiliki Jabatan Akademik Lektor Kepala dan praktisi industri yang berpengalaman. Saat ini menjabat sebagai Rektor Universitas Sains dan Teknologi Komputer (Universitas STEKOM), salah satu universitas terkemuka di Jawa Tengah, Indonesia. Dengan pengalaman lebih dari 13 tahun di dunia bisnis dan praktisi industri di China, beliau membawa perspektif global dan inovasi yang signifikan ke dalam dunia akademis.Sebagai seorang entrepreneur, penulis adalah pencipta TopLoker.com, sebuah platform inovatif yang merevolusi cara mencari dan menawarkan pekerjaan. TopLoker.com adalah portal lowongan bursa kerja terbesar di Indonesia, khusus untuk pendidikan SMA/SMK sederajat. TopLoker.com telah mendapatkan penghargaan sebagai juara 1 Startup4industry 2022 oleh Kementerian Perindustrian Republik Indonesia. Kontribusi Dr. Joseph dalam menyediakan akses pekerjaan yang luas bagi lulusan SMA/SMK telah membantu banyak individu menemukan peluang kerja yang sesuai dengan keahlian mereka. Selain itu, Dr. Joseph Teguh Santoso, M.Kom adalah pendiri dari dua organisasi yaitu (1) organisasi guru/pendidik PTIC (Perkumpulan Teacherpreneur Indonesia Cerdas) yang bertujuan untuk meningkatkan kualitas pendidikan dan kesejahteraan guru/pendidik dengan wawasan entrepreneurship, serta (2) organisasi industri PERKIVI (Perkumpulan Komunitas Industri dan Vokasi Indonesia) yang berfokus pada pengembangan link and match antara industri dan dunia pendidikan. Sebagai Rektor, Dr. Joseph Teguh Santoso, M.Kom memiliki kepemimpinan yang berorientasi pada hasil, dan berkomitmen untuk mendorong kemajuan Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Saat ini Universitas STEKOM telah mengalami transformasi positif dalam peningkatan kualitas pendidikan, perluasan fasilitas, serta penguatan kemitraan Perguruan Tinggi Nasional dan Internasional. Beliau memprioritaskan pengembangan sumber daya manusia dan penelitian, serta memastikan bahwa universitas berada di garis depan dalam inovasi dan teknologi untuk mencapai tujuan akhir, yaitu lulusan yang mampu bekerja dan sukses setelah lulus. Dr. Joseph Teguh Santoso, M.Kom sering diundang sebagai pembicara di berbagai konferensi nasional maupun internasional dan telah menerima berbagai penghargaan atas dedikasinya dalam bidang pendidikan, industri, dan kewirausahaan.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id