YAYASAN PRIMA AGUS TEKNIK

KEAMANAN SISTEM TERTANAM

(Embedded System Security)

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

KEAMANAN SISTEM TERTANAM

(Embedded System Security)

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

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

ISBN 978-623-8642-18-2 (PDF)

KEAMANAN SISTEM TERTANAM (Embedded System Security)

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

ISBN : 978-623-8642-18-2

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

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

KATA PENGANTAR

Puji Syukur kehadirat Tuhan Yang Maha Esa, Karna Berkat dan Rahmat-Nya, penulis dapat menyelesaikan buku yang berjudul ***"Keamanan Sistem Tertanam"***, dengan baik dan maksimal. Dalam buku ini pembaca akan memahami menggenai keamanan sistem tertanam yang memiliki manfaat signifikan. Dalam konteks ini pembaca akan mempelajarai keamanan sistem tertanam yang dapat membantu dalam mencegah serangan siber sebelum terjadi, yang dapat menyebabkan kerusakan fisik, pencurian data, atau bahkan ancaman terhadap keselamatan manusia. Dengan memahami keamanan istem tertanam, pihak individua atau pun organisasi dapat mempertahankan tingkat keamanan yang tinggi pada perangkat yang digunakan, yang penting untuk melindungi data sensitive dan menjaga integritas sistem.

Keamanan sistem tertanam (*embedded system security*) merupakan aspek krusial dalam pengembangan perangkat yang terintegrasi dengan fungsi tertentu, seperti perangkat IoT, kendaraan, dan sistem kontrol industri. Dengan meningkatnya konektivitas perangkat, risiko serangan siber juga semakin tinggi, sehingga penting untuk melindungi sistem ini dari potensi ancaman. Sistem tertanam sering kali memiliki sumber daya terbatas, baik dari segi pemrosesan maupun penyimpanan, yang membuat penerapan solusi keamanan yang kompleks menjadi tantangan. Selain itu, kesulitan dalam melakukan pembaruan dan patching perangkat lunak dapat menyebabkan kerentanan yang tidak teratasi. Oleh karena itu, strategi keamanan yang efektif harus diterapkan sejak tahap desain, termasuk penggunaan enkripsi untuk melindungi data, mekanisme autentikasi yang kuat, serta monitoring untuk mendeteksi intrusi. Pembaruan perangkat lunak secara berkala juga sangat penting untuk menutup celah keamanan. Dengan pendekatan yang proaktif dan komprehensif terhadap keamanan, pengembang dan pengguna dapat menjaga integritas dan keandalan sistem tertanam, serta melindungi mereka dari ancaman yang semakin kompleks.

Keamanan sistem tertanam atau embedded system security merujuk pada perlindungan terhadap ancaman terhadap sistem yang terintegrasi ke dalam perangkat keras (hardware) dan perangkat lunak (software) yang khusus dibuat untuk melakukan tugas tertentu. Sistem ini sering ditemukan dalam berbagai aplikasi, mulai dari kendaraan pintar hingga peralatan medis, dan memainkan peran penting dalam keamanan data, privasi pengguna, dan operasional yang stabil.

Dengan demikian Mempelajari keamanan sistem tertanam memberikan mahasiswa pemahaman mendalam tentang ancaman potensial seperti serangan fisik dan perangkat lunak berbahaya yang dapat mempengaruhi kinerja sistem dan data sensitif. Manfaat utamanya adalah mempersiapkan mahasiswa untuk mendesain sistem yang lebih aman, mengembangkan solusi keamanan yang efektif, serta memahami praktik terbaik dalam mengatasi risiko keamanan yang terkait dengan teknologi terkini. Ini adalah kompetensi krusial dalam era digital yang semakin terhubung dan rentan terhadap serangan cyber. Terima Kasih.

Semarang, Juli 2024
Penulis

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

DAFTAR ISI

BAB 1
PENGANTAR KEAMANAN SISTEM TERTANAM

1.1 APA ITU KEAMANAN?

Buku apa pun tentang keamanan harus dimulai dengan beberapa definisinya. Jika sepuluh profesional keamanan diminta untuk mendefinisikan istilah tersebut, sepuluh hasil berbeda akan diperoleh. Untuk mencapai validitas berbagai sistem tertanam dan fungsinya yang tak terhitung banyaknya, kuas kami menggunakan goresan yang luas:

> *Keamanan adalah kemampuan suatu entitas untuk melindungi sumber daya yang menjadi tanggung jawab perlindungannya.*

Dalam sistem tertanam, tanggung jawab perlindungan ini mungkin berlaku untuk sumber daya di dalam atau sumber daya sistem keseluruhan yang terhubung atau dimasukkan ke dalam sistem tertanam. Seperti yang akan kita bahas nanti di bab ini, sifat perlindungan suatu komponen atau sistem diwujudkan dalam kebijakan keamanannya.

1.2 APA YANG DIMAKSUD DENGAN SISTEM TERTANAM?

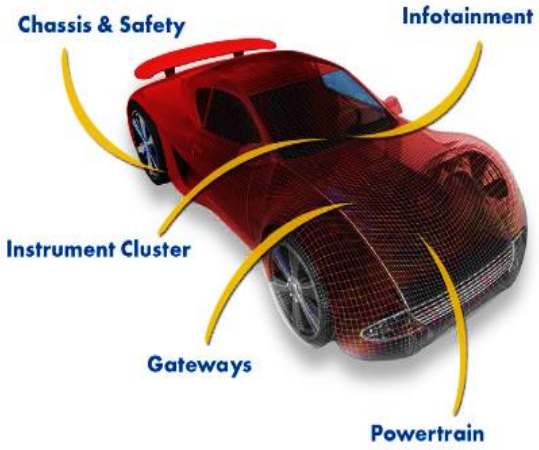
Upaya untuk mendefinisikan "sistem tertanam" juga sering kali penuh dengan kontroversi. Untuk keperluan buku ini, kami mendefinisikan sistem tertanam sebagai berikut:

> *Sistem tertanam adalah produk elektronik yang berisi mikroprosesor (satu atau lebih) dan perangkat lunak untuk menjalankan beberapa fungsi penyusun dalam entitas yang lebih besar.*

Setiap definisi sistem tertanam harus dilengkapi dengan contoh-contoh. Kami tidak mengklaim bahwa pesawat terbang merupakan sistem yang tertanam, namun merupakan sistem kendali penerbangannya; sistem penghindaran tabrakan lalu lintas (TCAS); sistem komunikasi, navigasi, dan pengawasan (CNS); sistem tas penerbangan elektronik (EFB); dan bahkan sistem hiburan dalam penerbangan merupakan contoh sistem tertanam di dalam pesawat (lihat Gambar 1.1).

Gambar 1.1 Sistem tertanam dalam pesawat komersial modern.

Kami tidak mengklaim mobil adalah sistem yang tertanam. Namun "head-unit" infotainment, sistem anti-lock, unit kontrol mesin powertrain, kluster instrumen digital, dan lusinan subsistem elektronik lainnya pada kartu modern umumnya merupakan contoh sistem tertanam (lihat Gambar 1.2).



Gambar 1.2 Beberapa sistem tertanam dalam mobil pada umumnya.

Sistem yang tertanam sering kali memiliki ciri yang berbeda: kebalikan dari sistem yang tertanam adalah komputer pribadi desktop yang mikroprosesor utamanya berbasis Arsitektur Intel (IA) mendukung antarmuka manusia dan lingkungan aplikasi yang berfungsi sebagai satu-satunya tujuan entitas. Demikian pula, mikroprosesor utama server yang dipasang di rak melakukan layanan khusus, seperti menghosting situs web.

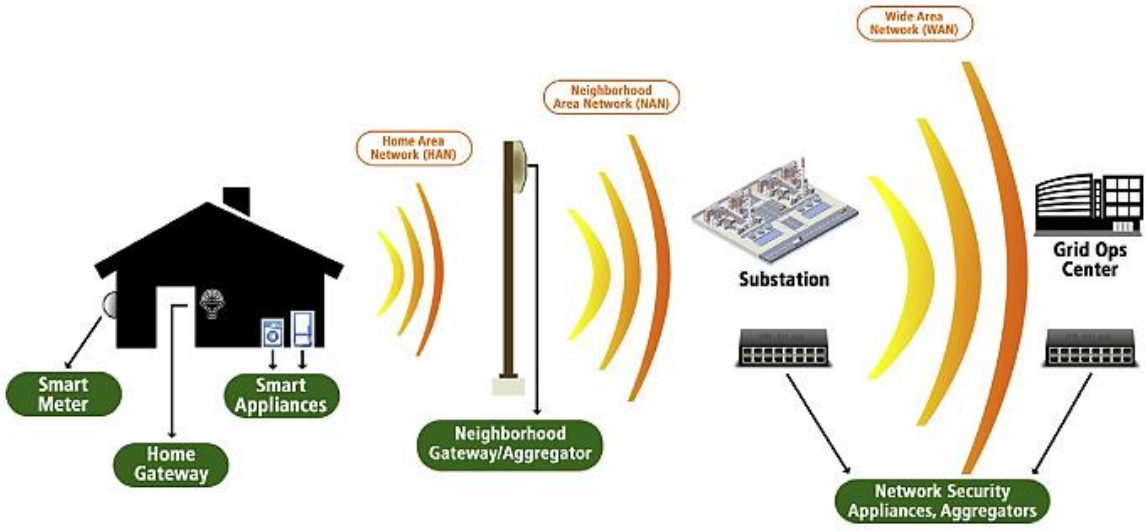
Area abu-abu menyebabkan kontroversi tersebut di atas. Ada yang berpendapat apakah ponsel cerdas merupakan sistem tertanam atau sekadar komputer desktop mini. Namun demikian, terdapat sedikit perdebatan bahwa masing-masing komponen di dalam telepon, seperti radio dengan mikroprosesor baseband dan perangkat lunaknya sendiri, merupakan sistem yang tertanam. Demikian pula, beberapa server berisi kartu anak tambahan yang melakukan pemantauan kesehatan dan manajemen jarak jauh untuk meningkatkan ketersediaan secara keseluruhan. Setiap kartu berisi mikroprosesor dan perangkat lunak sehingga memenuhi definisi kami tentang sistem tertanam.

Ruang lingkup buku ini secara luas mencakup telepon pintar yang keamanan keseluruhannya sangat bergantung pada perangkat keras dan perangkat lunak yang tertanam. Tentu saja, buku ini membahas tentang sistem tertanam yang terlibat dalam beberapa fungsi penting keamanan, dan beberapa sistem tertanam tidak memiliki persyaratan keamanan sama sekali. Buku ini secara umum tidak membahas tentang termostat mandiri bertenaga baterai yang dijalankan oleh mikrokontroler 8-bit dan beberapa kilobyte perangkat lunak yang diprogram dalam kode perakitan. Tantangan keamanan terbesar dalam sistem tertanam terletak pada produk elektronik canggih yang terhubung ke jaringan yang dikelola oleh sistem operasi tertanam yang menjalankan aplikasi perangkat lunak penting yang ditulis dalam bahasa pemrograman tingkat tinggi seperti C, C++, Ada, dan Java.

1.3 TREN KEAMANAN TERTANAM

MP944, yang oleh banyak orang dianggap sebagai mikroprosesor pertama di dunia, menjalankan sistem kontrol penerbangan di atas jet tempur F-14 Tomcat Angkatan Laut A.S. dan memulai kemajuan dalam teknologi sistem tertanam selama lebih dari 40 tahun. Bergantung pada pertanyaan analis tertentu, komputer tertanam mencakup 94% hingga 98% komputer di dunia. Praktis setiap perusahaan multinasional besar seperti Lockheed Martin, Exxon, General Motors, Hewlett Packard, dan Johnson & Johnson membangun dan bergantung pada sistem tertanam dalam produk-produk terpentingnya. Dan, tentu saja, rata-rata konsumen bergantung pada aplikasi yang tertanam dalam pesawat terbang, mobil, permainan, peralatan medis, dan sebagainya, secara terus-menerus.

Pada saat yang sama, kompleksitas perangkat lunak dan perangkat keras, konektivitas jaringan, dan ancaman serangan berbahaya terus tumbuh di sistem tertanam, yang semakin diandalkan untuk keselamatan dan keamanan konsumen. Jaringan cerdas dengan peralatan dan sensor cerdasnya, pengukur cerdas, dan gateway jaringan (semua sistem tertanam) merupakan contoh yang baik, namun hanya satu dari banyak contoh lainnya. Kumpulan kompleks sistem dan jaringan tertanam dalam smart grid ditunjukkan pada Gambar 1.3.



Gambar 1.3: Jaringan pintar, konten sistem tertanam, dan contoh topologi jaringan.

1.3.1 Kompleksitas Sistem Tertanam

Salah satu sistem tertanam pertama di dalam mobil adalah komputer perjalanan Cadillac Seville tahun 1978, dijalankan oleh mikroprosesor Motorola 6802 dengan RAM 128 byte dan ROM dua kilobyte. Kode sumber yang dicetak tidak boleh lebih dari beberapa halaman.

Sebaliknya, bahkan mobil kelas bawah saat ini memiliki setidaknya selusin mikroprosesor; mobil kelas atas diperkirakan berisi sekitar 100 mikroprosesor. Dengan sistem infotainment yang menjalankan sistem operasi canggih seperti Microsoft Windows dan Linux, total konten perangkat lunak yang tertanam dapat dengan mudah melebihi 100 juta baris kode. Avionik F-35 Joint Strike Fighter diperkirakan menampung sekitar 6 juta baris kode, yang digerakkan oleh kontrol fly-by-wire, kemampuan kesadaran situasional yang kompleks, pemrosesan sensor, dan tampilan grafis resolusi tinggi untuk pilot. Switch dan router jaringan

perusahaan secara rutin berisi jutaan baris kode untuk pemrosesan protokol jaringan, manajemen dan konfigurasi, pembatasan laju anti-virus, dan kontrol akses.

Singkatnya, kompleksitas didorong oleh permintaan yang tak terhindarkan akan kemampuan yang lebih baik, digitalisasi fungsi manual dan mekanis, serta interkoneksi dunia kita. Meskipun pertumbuhan konten elektronik ini bermanfaat bagi masyarakat, pertumbuhan tersebut juga merupakan sumber utama masalah keamanan.

> *Banyak masalah yang berkaitan dengan hilangnya kualitas, keselamatan, dan/atau keamanan produk elektronik disebabkan oleh meningkatnya kompleksitas yang tidak dapat dikelola secara efektif.*

Telah diketahui dengan baik bahwa kelemahan operasional, seperti buffer overflows (ketika perangkat lunak gagal memvalidasi panjang input, sehingga memungkinkan input untuk menimpa di luar akhir area memori yang dialokasikan yang digunakan untuk menampung input), sering kali merupakan penyebab utama. sarana dimana penyerang dapat menghindari kebijakan keamanan sistem. Kompleksitas tentu saja tidak dapat diukur hanya dengan ukuran kode atau jumlah transistor.

> *Pertumbuhan linier dalam konten perangkat keras/perangkat lunak menciptakan lebih dari sekedar pertumbuhan linier dalam keseluruhan kompleksitas karena peningkatan eksponensial dalam interaksi antara fungsi dan komponen.*

Kompleksitas melahirkan kelemahan, dan kelemahan dapat dieksploitasi untuk melanggar keamanan sistem. Mengontrol kompleksitas dari perspektif keamanan adalah salah satu perhatian utama buku ini.

Studi Kasus: Linux Tertanam

Untuk membantu lebih memahami ruang lingkup masalah kompleksitas ini dan memotivasi informasi di Bab 2 dan 3 mengenai keamanan perangkat lunak, mari kita lihat lebih dekat penggunaan Linux dalam sistem tertanam. Linux tertanam semakin populer karena lisensinya yang bebas royalti, aksesibilitas sumber terbuka, dan ketersediaan driver perangkat dan aplikasi yang luas. Meskipun memiliki ribuan kontributor di seluruh dunia, proses manajemen perubahan yang dikontrol ketat untuk Linux (terutama kernel Linux) sangat baik dibandingkan dengan standar kualitas perangkat lunak komersial pada umumnya. Steve McConnell, dalam bukunya Code Complete, memperkirakan rata-rata industri perangkat lunak memiliki sekitar 30 bug per 1.000 baris kode produksi.[1] Namun kernel Linux memiliki rekam jejak yang jauh lebih baik, yaitu antara 1 dan 5 bug per 10.000 baris kode.

Penggunaan Linux dalam sistem yang memerlukan tingkat keamanan tinggi sering menjadi topik kontroversi. Para pendukungnya mengklaim bahwa pendekatan open source Linux meningkatkan keamanan karena paparan komunitas pengembang dan pengguna di seluruh dunia (terkadang disebut teori “banyak mata”). Para pengkritiknya berpendapat bahwa kompleksitas dan arsitektur Linux membuatnya tidak cocok untuk aplikasi dengan tingkat kritikal tinggi.

Dua peristiwa baru-baru ini menyoroti perdebatan ini. Pada bulan Agustus 2009, Linux Foundation menerbitkan sebuah makalah, Linux Kernel Development (Pengembangan Kernel Linux), yang merinci perkembangan dan penyebaran Linux yang masif dan berkembang pesat

[1] McConnell S. *Code Complete. 2nd ed.* Redmond, WA; Microsoft Press; 2004.

dalam segala hal mulai dari ponsel hingga pesawat televisi dan kamera video.[2] Sekitar setahun kemudian, para peneliti menerbitkan rincian tentang dampak buruk yang parah. kerentanan kernel, yang telah ada di Linux selama delapan tahun sebelumnya.

Dengan umur Linux yang lebih dari 15 tahun, kini terdapat banyak statistik publik yang dapat digunakan untuk menganalisis ketahanan sistem operasi. Studi kasus ini membahas peristiwa-peristiwa terkini yang disebutkan di atas serta sumber informasi publik lainnya untuk menyimpulkan keadaan saat ini dan prospek Linux dalam sistem dengan keamanan tinggi.

Linux dalam Sistem Pemerintahan

Beberapa organisasi berpengaruh telah menjadi pendukung Linux dalam sistem komputer pemerintah yang kritis terhadap keamanan. Linux adalah sistem operasi tepercaya di NetTop HP, Raytheon/TCS Trusted Thin Client, dan General Dynamics Trusted Virtual Environment (TVE) yang merupakan produk program High-Assurance Platform (HAP) NSA. Semua produk ini dirancang untuk mengkonsolidasikan komputer yang digunakan oleh pegawai pemerintah untuk mengakses jaringan rahasia dan tidak rahasia. Komputer khusus ini menyediakan beberapa desktop "virtual" dan dipercaya untuk melindungi informasi sensitif. Untuk mempersiapkannya dengan lebih baik dalam tugas menjadi "titik kontak" antara jaringan-jaringan yang berbeda secara fisik, Linux ditingkatkan oleh Laboratorium Riset Jaminan Informasi Nasional NSA dengan kontrol keamanan tambahan, yang dikenal sebagai Security-Enhanced Linux (SELinux). Ekstensi SELinux telah diadopsi oleh komunitas Linux perusahaan dan digunakan dalam sistem komputer yang disebutkan di atas.

Seiring dengan investasi mereka di Linux, para pemasok militer ini telah membuat klaim yang berani tentang produk-produk ini yang dapat dipercaya. Menurut General Dynamics, TVE memberikan "ketangguhan tinggi" dan "lompatan kuantum dalam cara mengakses tingkat keamanan militer dan pemerintah."[3]

Menarik untuk dicatat, bagaimanapun, bahwa para pengembang NSA berhati-hati untuk tidak mengklaim kesesuaian untuk sistem dengan tingkat kritisitas tinggi, dengan menyatakan bahwa SELinux "sangat tidak mungkin memenuhi definisi sistem aman yang menarik." Selain itu, upaya SELinux juga mencakup "tidak ada upaya yang berfokus pada peningkatan jaminan Linux itu sendiri."[4]

Meskipun banyak diskusi mengenai keamanan Linux telah dikaburkan oleh hiperbola dan agenda komersial, sejumlah sumber independen, banyak yang diterbitkan oleh komunitas Linux, memberikan gambaran yang lebih lengkap dan tidak memihak tentang keamanan Linux.

Pengembangan Linux mengikuti praktik komersial umum, tidak mematuhi standar keselamatan atau keamanan yang ketat. Meskipun eksposur Linux terhadap sumber terbuka memungkinkannya mencapai tingkat kerusakan yang rendah dibandingkan dengan sebagian besar perangkat lunak komersial, ukuran kernel menjamin tingkat kerusakan yang besar dan berkelanjutan. Pada tahun 2004, alat analisis statis otomatis menemukan hampir 1.000 bug di kernel Linux.

---

[2] Kroah-Hartman G, Corbet J, McPherson A. *Linux Kernel Development: How Fast It Is Going, Who Is Doing It, What They Are Doing, and Who Is Sponsoring It: An August 2009* Update. Linux Foundation, 2009.

[3] General Dynamics. website: http://www.gdc4s.com/content/detail.cfm?item1/4570bc11a-1d6d-4acf-a68e-c256a615808d&page1/22.

[4] NSA SELinux Frequently Asked Questions: http://www.nsa.gov/research/selinux/faqs.shtml.

Institut Standar dan Teknologi Nasional AS dan Divisi Siber Keamanan Nasional Departemen Keamanan Dalam Negeri AS menerbitkan katalog, National Vulnerability Database (NVD), tentang cacat keamanan pada produk perangkat lunak komersial.[5] Pada 16 Agustus 2009, pencarian di Linux menghasilkan 1.288 entri, 457 di antaranya dianggap "Keparahan Tinggi." Seratus tiga puluh empat kerentanan tingkat tinggi dikaitkan dengan kernel Linux. NVD melaporkan 91, 77, 87, 111, dan 115 kerentanan kernel Linux masing-masing pada tahun 2006, 2007, 2008, 2009, dan 2010. Secara statistik dipastikan bahwa jumlah serupa akan ditemukan di tahun-tahun mendatang, yang menyiratkan bahwa terdapat banyak kerentanan dalam versi pengiriman saat ini. Angka-angka ini tentu saja belum memperhitungkan kerusakan yang tidak dilaporkan.

Pada 10 Agustus 2009, kebocoran memori pada ekstensi keamanan SELinux dipublikasikan di NVD. Beberapa hari kemudian, lima kerentanan lagi dipublikasikan. Salah satunya, CVE-2009-2692, melaporkan kerusakan kernel parah yang dapat dengan mudah dieksploitasi oleh pengguna untuk mengambil kendali penuh atas sistem. Kerentanan ini tersembunyi di kernel Linux selama delapan tahun!

Tingkat Perubahan Linux

Beberapa komponen perangkat lunak penting mendapatkan jaminan seiring berjalannya waktu. Hal ini terjadi ketika perangkat lunak relatif sederhana, hanya mengalami sedikit perubahan (kecuali mungkin untuk perbaikan bug), dan diterapkan dalam jangka waktu lama di berbagai lingkungan. Kernel Linux, bagaimanapun, mengalami modifikasi terus-menerus, termasuk di lapangan (misalnya patching over-the-air). Versi utama Linux terbaru, 2.6, telah berubah lebih cepat dibandingkan versi sebelumnya dan secara teratur mengalami modifikasi besar pada "cabang stabil" kernel. Sebagai contoh, pengembang Linux Greg Kroah-Hartman melaporkan bahwa kernel 2.6.24 melihat sekitar 5.000 baris kode ditambahkan per hari selama periode tiga bulan, hal ini memicu keluh kesahnya, "Sungguh menakjubkan bahwa semuanya masih berfungsi sama sekali."[6] Tingkat perubahan semakin cepat. Kroah-Hartman melaporkan bahwa rata-rata lebih dari 12.000 baris kode ditambahkan per hari selama siklus pengembangan 2.6.30. Sejak tahun 2005, kernel Linux telah dimodifikasi oleh lebih dari 5.000 orang dengan kecepatan melebihi enam perubahan per jam. Pada rilis pertama Linux 2.6 (2.6.0), kernel Linux terdiri dari lebih dari 5 juta baris kode. Pada 2.6.30, jumlah tersebut meningkat menjadi lebih dari 11 juta baris.

Contoh bagus lainnya diberikan oleh Jim Ready, pendiri vendor Linux tertanam MontaVista, yang membahas cacat NVD CVE-2006-1528, yang telah ditambal di Linux versi 2.6.13. Untuk mendapatkan perbaikan bug dalam rilis yang didukung, pengguna yang menjalankan 2.6.10 akan dipaksa untuk menggunakan 846.233 baris kode baru, yang mewakili perubahan antara versi 2.6.10 dan 2.6.13.[7]

CVE-2009-2692dIlustrasi Kehilangan Jaminan Total

Berikut ini adalah penjelasan rinci tentang kerentanan kernel parah yang disebutkan di atas. Kehalusan kelemahan tersebut, ditambah dengan bentuk program yang sangat sederhana yang digunakan untuk mengeksploitasinya, memberikan gambaran kepada

[5] http://web.nvd.nist.gov/view/vuln/search.
[6] http://kerneltrap.org/mailarchive/linux-kernel/2008/2/2/700024.
[7] Corbet J. An Interview with Jim Ready (June 11, 2008), LWN.net.

pembaca tentang kesulitan yang dihadapi dalam keamanan Linux. Pembaca yang ingin melewatkan detailnya harus melanjutkan ke bagian berikutnya.

Peneliti keamanan Tavis Ormandy dan Julien Tinnes menemukan serangkaian dereferensi penunjuk fungsi NULL di kernel Linux. Penunjuk fungsi berasal dari kode jaringan Linux. Linux menggunakan struktur, yang disebut proto_ops, untuk menampung serangkaian operasi prototipikal yang terkait dengan soket jaringan. Implementasi setiap operasi bervariasi antar kelompok soket (misalnya, AppleTalk, Bluetooth, Inframerah, Protokol Transmisi Kontrol Aliran untuk IPv6). Penunjuk fungsi yang terkait dengan operasi ini tidak diinisialisasi atau divalidasi secara konsisten di situs panggilan kernelnya. Ketika panggilan layanan soket dijalankan oleh suatu aplikasi, kernel Linux menentukan struktur proto_ops yang tepat untuk digunakan berdasarkan jenis soket. Kernel mengeksekusi penunjuk fungsi untuk operasi yang diminta dalam struktur ini.

Contoh kernel Linux berikut menunjukkan pointer proto_ops (operasi splice_read) yang dicentang NULL di situs panggilan:

```
static ssize_t sock_splice_read(struct file *file. loff_t
  *ppos, struct pipe_inode_info *pipe, size_t len, unsigned int
  flags)
{
struct socket *sock = file->private_data;
  if (unlikely(!sock->ops->splice_read))

return -EINVAL;
  return sock->ops->splice_read(sock. ppos. pipe. len. flags);
}
```

Dan berikut adalah contoh di mana cek yang sama tidak ada:

```
static ssize_t sock_sendpage(struct file *file, struct page
  *page, int offset, size_t
  size loff_t *ppos. int more)
{
struct socket *sock;
  int flags;
  sock - file->private_data:
  flags = !(file->f_flags & 0_NONBLOCK) ? 0 :
MSG_DONTWAIT;
  if (more)
  nags | - MSG_MORE:
return sock->ops->sendpage(sock, page, offset, size, flags);
}
```

Metode lain yang digunakan di Linux untuk menghindari dereferensi NULL proto-ops adalah dengan melakukan pra-inisialisasi penunjuk ke fungsi stub. Jika penunjuk halaman kirim pada contoh sebelumnya diinisialisasi ke stub Linux sock_no_sendpage, maka panggilan yang tidak dilindungi dalam kode sebelumnya akan menjalankan fungsi stub dan menghindari

penghormatan NULL. Banyak proto-op keluarga soket yang diinisialisasi dengan cara ini, misalnya yang digunakan untuk datagram inframerah:

```
static const struct proto_ops
     SOCKOPS_WRAPPED(irda_dgram_ops) = {
   .family =          PF_IRDA,
   .owner =           THIS_MODULE,
   .release =         irda_release,
   .bind =            irda_bind.
   .connect =   irda_connect,
   .socketpair =      sock_no_socketpair,
   .accept =          irda_accept.
   .getname =   irda_getname.
   .poll =            datagram_poll,
   .ioctl =           irda_ioctl,
#ifdef CONFIG_COMPAT
   .compat_ioctl =    irda_compat_ioctl.
#endif
   .listen =          irda_listen,
   .shutdown =  irda_shutdown,
   .setsockopt =      irda_setsockopt,
   .getsockopt =      irda_getsockopt.
   .sendmsg =   irda_sendmsg_dgram.
   .recvmsg =   irda_recvmsg_dgram,
   .mmap =            sock_no_mmap,
   .sendpage =  sock_no_sendpage.
};
```

Namun, contoh lain tidak diinisialisasi dengan benar; misalnya, proto-ops Bluetooth Network Encapsulation Protocol (BNEP) berikut:

```
static const struct proto_ops bnep_sock_ops = {
   .family           = PF_BLUETOOTH,
   .owner       = THIS_MODULE,
   .release          = bnep_sock_release,
   .ioctl       = bnep_sock_ioctl,
#ifdef CONFIG_COMPAT
   .compat_ioctl     = bnep_sock_compat_ioctl,
#endif
   .bind        = sock_no_bind,
   .getname          = sock_no_getname,
   .sendmsg          = sock_no_sendmsg,
   .recvmsg          = sock_no_recvmsg.
   .poll        = sock_no_poll,

   .listen           = sock_no_listen,
   .shutdown    = sock_no_shutdown,
   .setsockopt  = sock_no_setsockopt.
   .getsockopt  = sock_no_getsockopt.
   .connect          = sock_no_connect,
```

```
    .socketpair   = sock_no_socketpair,
    .accept            = sock_no_accept,
    .mmap         = sock_no_mmap
→MISSING SENDPAGE STUB INITIALIZATION
};
```

Oleh karena itu, penyerang hanya perlu menjalankan operasi sendpage pada soket BNEP terbuka untuk memaksa dereferensi NULL.

Eksekusi Kernel NULL dapat mempunyai berbagai efek. Misalnya, eksekusi halaman 0 dapat menyebabkan sistem crash sepenuhnya (serangan penolakan layanan yang sederhana namun total).

Namun, beberapa kelemahan yang tidak terkait pada kernel Linux memungkinkan penyerang untuk menyuntikkan kode eksekusi sewenang-wenang ke dalam kernel di alamat 0. Selain itu, seperti yang dilaporkan dalam kandidat NVD CVE-2009-2695 dan dirinci oleh Red Hat, ekstensi SELinux memperburuk alamat tersebut. 0 kerentanan eksekusi.

Kombinasi dari pointer NULL dan kerentanan injeksi kode kernel memungkinkan penyerang mengambil kendali penuh atas sistem dengan memaksa kernel untuk mengeksekusi kode penyerang.

Beberapa eksploitasi kelemahan proto-ops segera dipublikasikan. Eksploitasi ini adalah program sederhana yang dapat dijalankan oleh pengguna Linux untuk menunjukkan kegagalan keamanan total. Alur umum eksploitasi adalah sebagai berikut:

```
/* Code the attacker wants to execute at kernel */
/* Privilege to do absolutely anything!
Attacker_code(void)
{
   kprint(" *** You've Been Hacked !! \n");
   ...
*/
...
/* Service call that causes address 0 to be mapped */
unsigned char *Kernel0 = mmap(0. 0x1000. flags. 0, 0);
/* Install instructions at 0 to run Attacker's code */
Kernel0[0] = '\xff'; /* JMP DWORD -Jump indirect */
Kerne10[1] = '\x25';
*(unsigned long *)(Kerne10+2) = 6;/* Jump target is @6 */
*(unsigned long *) (Kerne10+6) = (unsigned long)&Attacker_code;
/* Create socket of type containing flawed proto_ops */

fd=socket(PF_BLUETOOTH, SOCK_DGRAM, 0);
/* Use socket operation to cause NULL dereference */
/* This triggers the attacker's code to run */
sendfile(fd, ... );
```

Linux memiliki “fitur” yang membuat eksploitasi ini sedikit lebih mudah: untuk meningkatkan kinerja panggilan sistem, Linux tetap mengaktifkan pemetaan memori aplikasi pengguna selama layanan panggilan sistem. Hal ini menghilangkan kebutuhan untuk melakukan

terjemahan alamat manual pada layanan parameter panggilan yang terletak di memori aplikasi. Dengan demikian, halaman memori aplikasi di alamat 0 menjadi halaman kernel di alamat 0, dan Attacker_code secara otomatis dipetakan dan dieksekusi oleh kernel.

Penutupan Studi Kasus

Meskipun Linux memenuhi visi penciptanya sebagai sistem operasi serba guna berkinerja tinggi, Linux tidak pernah dirancang untuk tingkat ketahanan yang dituntut oleh sistem dengan tingkat kritisitas tinggi. Ini tidak berbeda dengan sistem operasi tujuan umum lainnya. Bayangkan menggunakan Windows untuk menjalankan sistem kendali penerbangan di pesawat modern dan tindakan yang bisa memberi arti baru pada istilah layar biru kematian.

Solaris adalah sistem operasi serba guna lainnya yang telah digunakan untuk konsolidasi jaringan (digunakan di Stasiun Kerja Tepercaya DoDIIS militer). CVE-2007-0882 (keparahan tinggi) menggambarkan cacat yang memalukan pada Solaris 10 dan 11, yang memungkinkan penyerang mana pun untuk masuk dari jarak jauh, sepenuhnya melewati otentikasi.

Jadi, untuk lebih jelasnya, meskipun statistik dalam studi kasus ini mengacu pada Linux, kesimpulannya bersifat umum untuk produk perangkat lunak berskala besar, monolitik, dan bertujuan umum. Selain itu, masalah keamanan dalam sistem operasi tujuan umum tidak hanya disebabkan oleh kompleksitas perangkat lunak; arsitektur kernel adalah faktor penting lainnya, seperti yang kita bahas di Bab 2.

Linux yang tertanam adalah contoh yang baik dari perangkat lunak yang umumnya dibuat dengan baik, karena kompleksitas kodenya yang sangat besar dan laju perubahan yang cepat, penuh dengan kerentanan dan karenanya tidak cocok untuk melindungi sumber daya bernilai tinggi dari ancaman serangan yang canggih.

1.3.2 Konektivitas Jaringan

Tren lain yang jelas dalam sistem tertanam adalah penambahan konektivitas jaringan. Sistem tertanam secara tradisional dianggap sebagai entitas mandiri yang kebal terhadap risiko Internet. Ada banyak alasan mengapa sistem tertanam dijadikan jaringan. Konektivitas memungkinkan manajemen jarak jauh dari sistem tertanam. Tugas manajemen mungkin mencakup peningkatan perangkat lunak untuk mengatasi kelemahan. Konektivitas jaringan menambah kemampuan baru. Misalnya, oven rumah yang dibuat oleh TMIO memungkinkan pemiliknya terhubung dari ponsel atau Internet untuk menghidupkan dan mematikan pemanas dari jarak jauh. Tiba-tiba, peralatan rumah tangga menjadi sangat penting bagi keamanan.

Pada tahun 2010, General Motors memperkenalkan fitur yang memungkinkan pemilik mobil memanipulasi kunci dan menyalakan mesin dari mana saja di dunia menggunakan ponsel pintar. Koneksi jarak jauh ini mendukung sistem telematika OnStar GM, yang menjadi standar di semua model GM pada tahun 2007. Apakah konsumen harus khawatir tentang dampak keamanan dari konektivitas ini?

Sesaat sebelum GM mengumumkan fitur ponsel pintar, tim peneliti universitas menerbitkan sebuah penelitian yang menunjukkan bagaimana sistem penting mobil seperti

rem, pelambatan mesin, dan sebagainya dapat dirusak secara jahat dengan mengeksploitasi kerentanan dalam sistem yang tertanam pada mobil.[8]

Para peneliti menyita rem mobil, mesin, dan kunci pintu melalui port diagnostik. Mereka belajar bagaimana menjembatani jaringan dengan keamanan rendah ke sistem kritis menggunakan teknik fuzzing. Para peneliti menunjukkan tekad yang mengagumkan; praktis setiap subsistem penting utama pada mobil ditemukan, dipelajari, dan kemudian ditumbangkan sepenuhnya. Rem dan mesin dinonaktifkan saat mobil sedang melaju, hal ini menunjukkan bahwa serangan tersebut dapat membahayakan penumpang. Makalah penelitiannya luar biasa, wajib dibaca oleh para profesional dan penggemar keamanan tertanam.

Banyak artikel dan blog telah ditulis sebagai tanggapan terhadap penelitian ini, namun reaksi keseluruhannya tidak terdengar, hampir seperti mengantuk. Hal ini mungkin disebabkan oleh upaya tekun penulis untuk mencegah kepanikan:

- "Kami tidak tertarik untuk mengambil nada yang mengkhawatirkan."
- "Kami tidak punya alasan untuk percaya bahwa hal ini merupakan masalah saat ini."
- "Saat ini semua orang fokus pada keamanan Web dan botnet. Kami ingin memastikan bahwa dalam 5 atau 10 tahun kami tidak menambahkan mobil ke daftar tersebut."

Tidak adanya alarm merupakan hal yang mengejutkan dan memprihatinkan. Apakah para peneliti menganjurkan keamanan karena ketidakjelasan? Mereka menolak untuk mengungkapkan merek dan model mobil yang diretas dan tidak merilis alat "car shark" yang digunakan untuk melakukan subversi. OnStar selalu menyediakan koneksi jarak jauh. Melampirkan ke jaringan seluler akan membuka lebih banyak peluang serangan. Beberapa orang mungkin bertanya mengapa ada orang yang ingin menyerang jaringan mobil. Itu seperti menanyakan mengapa ada orang yang ingin menyerang jaringan listrik? Apa cara yang lebih baik untuk menjamin terjadinya bencana selain menonaktifkan rem jutaan mobil secara bersamaan? Orang-orang jahat juga memiliki peneliti yang sangat cerdas dan berdedikasi.

Kita perlu mengambil nada yang mengkhawatirkan. Penelitian menunjukkan bahwa kita mempunyai jutaan mobil yang rentan di jalan. Kita sekarang tahu bahwa penyerang cukup canggih untuk menonaktifkan rem pengemudi saat ia melaju di jalan raya. Satu-satunya pertanyaan adalah apakah penyerang cukup canggih untuk menemukan jalan masuk dari jarak jauh.

Para peneliti menyatakan: "Di dalam mobil kami, kami mengidentifikasi tidak kurang dari lima jenis antarmuka radio digital yang menerima masukan dari luar, beberapa hanya dalam jarak pendek dan lainnya dalam jarak tidak terbatas. Secara keseluruhan, kendali komputer di mana-mana, konektivitas internal terdistribusi, dan antarmuka telematika semakin banyak yang bergabung untuk menyediakan platform perangkat lunak aplikasi dengan akses jaringan eksternal."

Ironisnya, layanan konektivitas darurat, seperti OnStar milik GM, kini dapat menyediakan sarana untuk serangan jarak jauh yang terdistribusi. Penumpang menginginkan internet di dalam dan aplikasi ponsel cerdas untuk mengontrol fungsi kenyamanan, namun mereka tidak pernah mengharapkan antarmuka ini terhubung ke drivetrain. Para peneliti

[8] Koscher K, Czeskis A, Roesner F, Patel S, Kohno T. Experimental Security Analysis of a Modern Automobile. Oakland, CA: 2010 IEEE Symposium on Security and Privacy; May 19, 2010.

melanjutkan: "CLS [*Central Locking System*] juga harus terhubung dengan sistem penting keselamatan seperti deteksi tabrakan untuk memastikan bahwa kunci mobil terlepas setelah kantung udara dipasang untuk memfasilitasi keluar atau penyelamatan."

Apa yang tidak dibicarakan oleh para peneliti adalah apa yang dapat kita lakukan mengenai keamanan otomotif tertanam saat ini. Kemungkinan besar perubahan kecil dapat dilakukan untuk mengisolasi subsistem jaringan dengan lebih baik.

Otentikasi kriptografi yang kuat harus digunakan untuk semua koneksi jaringan. Platform tepercaya dan pengesahan jarak jauh harus digunakan untuk mencegah pemasangan firmware palsu memaparkan jaringan mobil kepada penyerang. Unit Kontrol Elektronik (ECU) dengan fungsionalitas kritis campuran harus menggunakan partisi dan kontrol akses dengan jaminan tinggi: kamera tampak belakang tidak boleh terpengaruh oleh iTunes.

Pabrikan mobil dan OEM tingkat 1 mungkin tidak terlalu memikirkan keselamatan saat mereka merancang mobil yang akan digunakan di jalanan saat ini, namun hal tersebut jelas harus diubah. Produsen harus bekerja sama dengan spesialis keamanan tertanam sejak awal dalam desain dan arsitektur elektronik dan jaringan di dalam mobil. Banyak dari topik ini dieksplorasi secara rinci di seluruh buku ini.

1.3.3 Ketergantungan pada Sistem Tertanam untuk Infrastruktur Kritis

Sebelumnya, kami menyebutkan smart grid sebagai sumber sistem tertanam yang penting dan muncul dengan persyaratan keamanan. Salah satu kekhawatiran utama dalam kasus ini adalah masalah finansial: penyerang dapat memanipulasi informasi pengukuran dan menumbangkan perintah kontrol untuk mengalihkan rabat listrik konsumen ke akun palsu.

Jaringan pintar (smart grid) menyiratkan penambahan konektivitas jarak jauh, dari jutaan rumah, ke sistem back-end yang mengontrol pembangkitan dan distribusi listrik. Sistem back-end ini dilindungi oleh teknologi keamanan yang sama (firewall, otentikasi akses jaringan, deteksi intrusi, dan sistem perlindungan) yang saat ini melindungi bank dan pemerintah dari serangan yang ditularkan melalui Internet. Intrusi yang berhasil ke dalam sistem ini terjadi setiap hari. Kemampuan untuk mempengaruhi distribusi tenaga listrik mempunyai konsekuensi keselamatan yang jelas, dan potensi untuk mempengaruhi populasi yang besar meningkatkan daya tarik dari target tersebut. Jaringan pintar (smart grid), jika tidak dirancang dengan baik demi keamanan, dapat memberikan jalur serangan bagi negara-negara yang bermusuhan dan teroris dunia maya dari kenyamanan ruang keluarga mereka. Setiap sistem tertanam di jalur ini, mulai dari peralatan pintar, meteran pintar, hingga konsentrator jaringan, harus aman.

> *Perusahaan utilitas dan pemasoknya masih dalam tahap awal pengembangan strategi keamanan dan arsitektur jaringan untuk jaringan pintar; kini ada peluang emas untuk membangun keamanan sejak awal.*

Sistem transportasi, pesawat terbang, kereta api, elevator, dan kendaraan industri mungkin merupakan contoh paling nyata dari teknologi tertanam yang sangat penting bagi keselamatan. Pada tahun 2008, Administrasi Penerbangan Federal AS (FAA) menyatakan kekhawatirannya mengenai aksesibilitas sistem kontrol penerbangan pada pesawat komersial terbaru Boeing, 787 Dreamliner, dari sistem hiburan penumpang dalam penerbangan melalui jaringan internal pesawat. Pemberitahuan FAA kepada Boeing untuk mengatasi potensi akses berbahaya terhadap avionik penting yang tertanam dari penumpang kemungkinan merupakan yang pertama dalam sejarah pesawat komersial.

Infrastruktur keuangan, termasuk bank dan pemroses kartu kredit, telah menjadi target peretas sejak Internet menjadi populer, dan sistem tertanam (embedded system) mengambil peran lebih besar dalam transaksi keuangan bernilai tinggi. Misalnya, kios titik penjualan yang canggih di perusahaan ritel memberikan target yang menarik: kemampuan untuk menumbangkan sistem tertanam dapat memberikan akses ke nomor kartu kredit yang disimpan secara lokal serta jalur ke jaringan pembayaran back-end yang menampung ratusan ribu orang. hingga jutaan catatan keuangan sensitif disimpan.

Beberapa perangkat medis memiliki persyaratan keamanan. Sistem tertanam yang penting bagi kehidupan dengan konten perangkat lunak yang signifikan mencakup sistem pemantauan pasien, peralatan pencitraan, dan perangkat nirkabel yang dapat ditanamkan. Sistem radiografi digital berisi basis kode sumber jutaan baris dan konektivitas jaringan untuk manajemen jarak jauh dan pengarsipan data. Peneliti keamanan baru-baru ini mendemonstrasikan kemampuan untuk mengendalikan alat pacu jantung melalui jaringan nirkabel.[9] Rekam medis elektronik (EMR) menimbulkan ancaman terhadap rekam medis yang sensitif; dokter menggunakan perangkat seluler untuk mengakses informasi pasien melalui Internet.

Pengendalian proses industri adalah contoh penting lainnya dari sistem tertanam yang kritis terhadap keamanan. Kami membahas hal ini lebih lanjut di bagian berikut.

1.3.4 Penyerang Canggih

Meningkatnya ketergantungan pada sistem tertanam dalam perdagangan, infrastruktur penting, dan fungsi-fungsi penting membuat mereka menarik bagi penyerang yang memiliki pendanaan besar dan gigih. Sistem kendali industri yang tertanam dalam mengelola reaktor nuklir, kilang minyak, dan infrastruktur penting lainnya menghadirkan peluang terjadinya kerusakan yang luas.

Stuxnet menyusup ke sistem kendali proses Siemens di pembangkit listrik tenaga nuklir dengan terlebih dahulu menumbangkan stasiun kerja Microsoft Windows yang digunakan operator untuk mengkonfigurasi dan memantau elektronik kendali tertanam (lihat Gambar 1.4).

Gambar 1.4 Infiltrasi Stuxnet pada sistem kontrol proses penting melalui PC operator.

[9] Halperin D, Heydt-Benjamin TS, Ransford B, Clark SS. Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses. Oakland, CA: 2008 IEEE Symposium on Security and Privacy; March 12, 2008.

Worm Stuxnet kemungkinan merupakan malware pertama yang secara langsung menargetkan sistem kontrol proses tertanam dan menunjukkan potensi kecanggihan yang luar biasa dalam serangan keamanan tertanam modern.

Sebagian besar diskusi komunitas keamanan tentang Stuxnet adalah spekulasi mengenai identitas dan motif penyerang serta tingkat kecanggihan serangan yang belum pernah terjadi sebelumnya, yang mencakup konstruksi rootkit yang cerdas dan penggunaan tidak kurang dari empat kerentanan Windows zero-day. Kerentanan ini memungkinkan Stuxnet mendapatkan akses dan mengunduh malware ke pengontrol Siemens itu sendiri, yang menyiratkan bahwa penyerang memiliki pengetahuan mendalam tentang perangkat lunak dan perangkat keras yang tertanam di dalamnya.

Stuxnet menunjukkan perlunya peningkatan keterampilan keamanan dalam komunitas pengembangan tertanam, tetapi juga menjelaskan persyaratan untuk tingkat jaminan yang lebih tinggi dalam infrastruktur penting dibandingkan dengan praktik TI komersial standar. Pada Bab 3, kami membahas tingkat jaminan keamanan yang sesuai dengan tingkat ancaman dan bagaimana pendekatan standar Kriteria Umum internasional terhadap pemetaan ini.

Stuxnet juga menunjukkan bahwa sistem tertanam dan sistem TI seringkali saling bergantung. Jaringan SCADA dikendalikan oleh PC umum. Sebagai tanggapan terhadap Stuxnet, Kepala Komando Siber Departemen Pertahanan AS dan Direktur NSA, Jenderal Keith B. Alexander, merekomendasikan pada bulan September 2010 pembuatan jaringan terisolasi untuk infrastruktur penting. Hal ini mungkin terdengar seperti pendekatan yang berat, namun hal ini menunjukkan betapa banyak pemerintah yang melindungi jaringan rahasia mereka yang paling sensitif dan terkotak-kotak. Isolasi fisik menimbulkan beberapa inefisiensi yang dapat diperbaiki dengan penerapan solusi akses jaminan tinggi yang memungkinkan komputer klien mengakses beberapa desktop virtual terisolasi dan jaringan back-end dengan aman. Sistem kontrol akses ini menggunakan antarmuka manusia-mesin Windows atau Linux terbaru dan terhebat tetapi tidak bergantung pada Windows atau Linux untuk keamanannya. Sebaliknya, mereka mengandalkan teknik yang disebut MILS Virtualization. Virtualisasi MILS dibahas dalam studi kasus yang melibatkan radio yang ditentukan perangkat lunak (SDR) di Bab 6.

1.3.5 Konsolidasi Prosesor

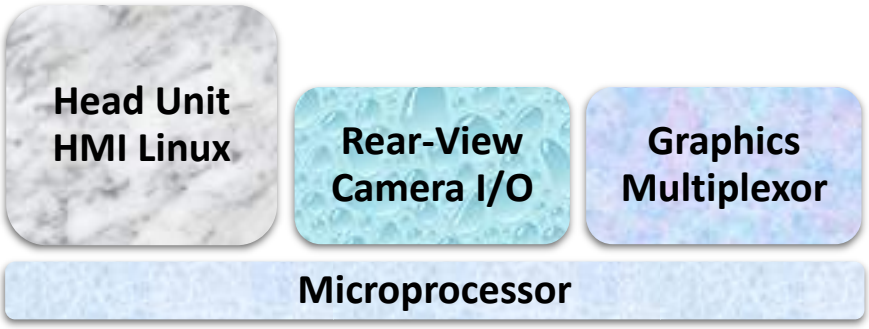
Tren lain yang berdampak pada keamanan adalah konsolidasi prosesor dalam sistem tertanam. Mobil memberikan contoh bagus lainnya. Ketika mobil melanjutkan transformasinya yang tak terhindarkan menjadi sistem sistem elektronik, jumlah komponen elektronik dan konten kabel terkait di dalam mobil telah meroket. Pertumbuhan barang elektronik ini menimbulkan biaya produksi, jejak fisik, dan tantangan waktu pemasaran yang signifikan bagi produsen otomotif. Responsnya adalah membalikkan tren pertumbuhan dan menggabungkan fungsi-fungsi yang berbeda ke dalam jumlah komponen elektronik yang lebih sedikit. Konsolidasi memerlukan arsitektur sistem yang tepat untuk memastikan bahwa komponen-komponen ini tidak berinteraksi dengan cara yang tidak terduga, sehingga menimbulkan risiko keandalan pada sistem kritis. Bab 2 membahas inti arsitektur ini: lingkungan operasi partisi yang mampu memberikan jaminan sumber daya yang ketat, eksekusi deterministik, kontrol akses yang ketat, pengerasan aplikasi, dan jaminan yang tinggi.

Contoh penerapan konsolidasi yang muncul dapat ditemukan nanti di bab ini dan khususnya di Bab 6.

Konsolidasi prosesor sangat selaras dengan tren ke arah sistem kekritisan campuran yang mana keselamatan, keamanan, atau komponen penting secara real-time harus hidup berdampingan dengan komponen yang kurang penting.

> *Berdasarkan survei informal terhadap pengembang sistem tertanam yang menghadiri Konferensi Sistem Tertanam baru-baru ini, kami memperkirakan bahwa sistem kritis campuran akan mencakup setidaknya 60% dari seluruh proyek tertanam pada tahun 2020.*

Misalnya, menggabungkan head-unit infotainmen dengan komponen kamera pandangan belakang menghasilkan sistem tertanam kekritisan campuran (lihat Gambar 1.5).



Gambar 1.5: Sistem tertanam otomotif kekritisan campuran.

Karena dapat berbagi kemampuan audio dan video dari komputer tumpukan tengah, modul kamera pandangan belakang merupakan kandidat alami untuk konsolidasi. Meskipun kamera belakang masih bersifat opsional pada banyak model mobil, kamera ini masih dianggap sebagai fungsi penting bagi keselamatan karena kemampuannya memperingatkan pengemudi akan bahaya (seperti sepeda roda tiga dan anak kecil) yang mungkin luput dari perhatian hanya dengan penglihatan manusia.

1.4 KEBIJAKAN KEAMANAN

Hingga saat ini, kita telah membicarakan keamanan dalam pengertian yang agak umum, tanpa memformalkan properti keamanan spesifik yang perlu diterapkan oleh sistem tertanam kami. Properti dan persyaratan komponen atau sistem yang mendukung keamanan tercakup dalam apa yang disebut kebijakan keamanan komponen. Kebijakan keamanan dapat memiliki rincian yang sangat bervariasi, bergantung pada tingkat detail yang diperlukan. Misalnya, pembeli produk firewall mungkin tertarik dengan kebijakan keamanan tingkat tinggi yang menyatakan, “Paket ingress yang menunjukkan format yang ditemukan pada daftar hitam lalu lintas yang tidak diizinkan yang dapat dikonfigurasi tidak boleh diteruskan melalui port egress firewall.” Administrator firewall tertarik dengan kebijakan keamanan khusus penerapan yang dikonfigurasi dalam daftar hitam. Kebijakan ini mungkin terdiri dari ratusan aturan individual yang memblokir port, protokol, alamat asal dan tujuan tertentu, serta permutasi parameter tersebut.

Kebijakan keamanan diperlukan untuk melakukan evaluasi praktis terhadap keamanan produk. Dalam evaluasi keamanan, evaluator menilai jaminan yang dimiliki pengguna produk

sehubungan dengan kebijakan keamanan yang diklaim. Kebijakan keamanan dibuat untuk melawan ancaman.

1.4.1 Keamanan Sempurna

Pembaca mungkin pernah mendengar pernyataan ini sebelumnya: "Tidak ada keamanan yang sempurna," atau konsekuensinya, "Satu-satunya hal yang benar-benar aman adalah batu bata." Sentimen ini datang dari pengguna teknologi, pengembang teknologi, dan bahkan pakar keamanan di seluruh dunia.

Sayangnya, sikap ini merugikan. Dengan memulai dengan pernyataan negatif yang luas, pengembang diberikan alasan untuk tidak mengupayakan keamanan terbaik. Lebih jauh lagi, pernyataan tersebut salah. Faktanya, adalah mungkin untuk mencapai keamanan yang sempurna. Namun penting untuk disadari bahwa kesempurnaan hanya dapat diterapkan pada kebijakan keamanan tertentu, seperti yang baru saja kami jelaskan. Meskipun tidak mungkin untuk membuktikan kebijakan keamanan mikroprosesor multiinti miliaran transistor, sejumlah kecil perangkat lunak penting, seperti sistem operasi mikrokernel tertanam, telah terbukti secara matematis kebijakan keamanannya. Dan bukti formal ini diperkuat dengan pengujian fungsionalitas, cakupan, dan platform perangkat keras yang menyeluruh serta sejumlah besar artefak jaminan lainnya yang telah dievaluasi dan diafirmasi oleh para ahli terkemuka dalam sertifikasi keamanan (seperti Badan Keamanan Nasional AS). Di Bab 3, pembaca dapat menemukan studi kasus terperinci tentang kebijakan keamanan sistem operasi dengan jaminan tinggi.

1.4.2 Kerahasiaan, Integritas, dan Ketersediaan

Sebagian besar kebijakan keamanan dapat dipetakan ke satu atau lebih elemen perlindungan kerahasiaan, integritas, dan ketersediaan (CIA). Kerahasiaan adalah pencegahan pengungkapan informasi yang tidak sah. Integritas adalah pencegahan modifikasi atau korupsi sumber daya. Ketersediaan adalah pencegahan serangan yang akan membuat sumber daya tidak dapat diakses atau digunakan sesuai dengan fungsi yang dimaksudkan.

Bayangkan sebuah kebijakan keamanan yang memerlukan otentikasi kunci publik yang kuat untuk menjaga akses jaringan jarak jauh ke sistem infotainment mobil. Kebijakan ini dapat diterapkan untuk mencegah paparan aset yang dikelola hak digital (kerahasiaan) serta untuk mencegah penyerang jarak jauh menjembatani jaringan infotainment untuk menonaktifkan sistem penting mobil (ketersediaan).

Saat menentukan kebijakan keamanan yang tepat untuk diterapkan dalam sistem tertanam, pengembang harus mulai dengan memikirkan perlindungan kerahasiaan, integritas, dan ketersediaan (CIA) tingkat tertinggi yang harus ditegakkan.

Pada Bab 2, kami membahas keamanan perangkat lunak sistem secara rinci, namun untuk membantu menjelaskan konsep dan penerapan kebijakan keamanan dengan lebih baik, berikut kami merangkum beberapa kebijakan keamanan yang relevan untuk komponen perangkat keras dan perangkat lunak, seperti sistem operasi dan unit manajemen memori, dalam mengelola sumber daya fisik platform tertanam (memori, waktu CPU, akses I/O).

1.4.3 Isolasi

Pada mikroprosesor tertanam yang menyediakan perangkat keras perlindungan memori (misalnya, unit manajemen memori), sistem operasi atau hypervisor menggunakan fasilitas perangkat keras ini dan teknik perangkat lunaknya sendiri untuk menerapkan kebijakan isolasi antar komponen perangkat lunak.

Isolasi komponen adalah kebijakan keamanan sistem tertanam dasar yang diperlukan untuk dapat mewujudkan kebijakan keamanan sistem tingkat tinggi.

Misalnya, kernel sistem operasi harus diisolasi dari aplikasi yang di-host sehingga sistem operasi dapat menerapkan kebijakan kontrol akses untuk sumber daya I/O yang berada di bawah kendalinya. Tanpa isolasi ini, aplikasi yang salah atau berbahaya dapat merusak kernel dan mencegahnya menyediakan layanan keamanan lainnya. Isolasi, dalam konteks ini, mengacu pada isolasi temporal dan spasial; aplikasi yang melarikan diri tidak boleh menolak eksekusi aplikasi lain atau kernel itu sendiri.

Contoh bagus lainnya dari kekuatan komposisi kebijakan isolasi dapat dilihat dalam penerapan layanan kriptografi yang efektif dalam suatu sistem. Aplikasi seperti IPsec atau sistem file enkripsi, yang menggunakan kriptografi untuk mewujudkan kebijakan keamanannya, bergantung pada isolasi subsistem kriptografi (terutama kunci privat) dari aplikasi atau subsistem lain. Tanpa isolasi ini, kunci pribadi dapat terekspos, sehingga menjadikan kebijakan kerahasiaan yang dimaksudkan oleh perlindungan data-in-motion (DIM) atau data-at-rest (DAR) tidak berdaya. Data-in-motion, terkadang juga disebut sebagai data-in-transit (DIT), keamanan diaktifkan oleh protokol keamanan jaringan, dan data tidak aktif dengan enkripsi media penyimpanan. Keduanya dibahas di Bab 5.

Isolasi juga dapat membatasi kerusakan akibat kegagalan keamanan. Misalnya, jika sistem file rusak dan gagal menyediakan layanan file, tumpukan jaringan yang terisolasi dapat terus menyediakan layanan komunikasinya. Pembatasan kerusakan dapat dianggap sebagai kebijakan keamanan itu sendiri. Bab 2 membahas keunggulan keamanan relatif dalam arsitektur sistem operasi yang mendukung (atau mencegah) isolasi komponen antara layanan sistem operasi umum seperti sistem file dan jaringan.

1.4.4 Pengendalian Arus Informasi

Layanan sistem operasi, seperti permintaan untuk memunculkan thread atau memanipulasi perangkat I/O, adalah contoh aliran informasi yang dikontrol secara ketat antara aplikasi dan kernel sistem operasi. Untuk menjamin integritas layanannya sendiri dan kebijakan keamanan tingkat tinggi, kernel harus mengkonfigurasi aplikasi dan memvalidasi parameter panggilan layanan sesuai dengan kebijakan kontrol aliran informasi sistem.

Kebijakan Arus Informasi		Penerima		
		Alice	Bob	Charlie
Pengirim	Alice	Ya	Ya	Tidak
	Bob	Ya	Ya	Tidak
	Charlie	Tidak	Tidak	Ya

Gambar 1.6. Contoh kebijakan keamanan aliran informasi.

Misalnya, kebijakan arus informasi mungkin mengizinkan komponen Alice dan Bob untuk berkomunikasi, tetapi tidak mengizinkan Alice dan Charlie. Sistem operasi dapat menerapkan kebijakan ini dengan mengizinkan permintaan runtime oleh Alice untuk mengirim pesan ke Bob tetapi menolak permintaan Alice atau Bob untuk mengirim pesan ke Charlie (dan sebaliknya), seperti pada Gambar 1.6.

Alternatifnya, perancang sistem dapat mengkonfigurasi Alice dan Charlie secara statis sedemikian rupa sehingga mereka tidak memiliki deskriptor komunikasi untuk membuat permintaan semacam itu ke sistem operasi. Kontrol akses sistem operasi tertanam serta kebijakan dan mekanisme keamanan utama lainnya dibahas lebih lanjut di Bab 2.

1.4.5 Kebijakan Keamanan Fisik

Berbeda dengan serangan berbasis perangkat lunak digital, beberapa sistem tertanam harus dilindungi dari serangan fisik. Misalnya, agregator jaringan pintar, yang dipasang pada tiang listrik di lingkungan sekitar, menjaga akses (melalui koneksi jaringan terenkripsi yang diautentikasi) ke jaringan pintar dari sejumlah besar meteran pintar di lingkungan sekitar. Seorang penyerang yang ingin mendapatkan akses tidak sah ke dalam infrastruktur back-end dapat menggunakan serangan fisik untuk merusak agregator utilitas, mencoba mencuri kunci pribadi yang akan menghasilkan akses jaringan yang diinginkan. Radio yang digunakan di lingkungan medan perang untuk komunikasi rahasia harus dilindungi dari serangan fisik yang mungkin terjadi jika musuh menangkap radio tersebut.

Serangan fisik datang dalam dua bentuk: invasif dan non-invasif. Serangan invasif pada agregator jaringan pintar mungkin termasuk membuka penutup agregator untuk mendapatkan memori flash dan mencuri kunci pribadi dan perangkat lunak yang tersimpan.

Serangan non-invasif mungkin termasuk menempatkan monitor energi di dekat agregator untuk melakukan analisis daya diferensial (DPA) sistem; pola konsumsi daya dapat digunakan untuk memperoleh informasi tentang fungsionalitas perangkat lunak seperti eksekusi algoritma kriptografi terkenal seperti Advanced Encryption Standard (AES). DPA dan teknik lainnya telah berhasil diterapkan untuk memulihkan kunci rahasia. TEMPEST adalah nama kode yang diciptakan pada tahun 1960-an, mengacu pada penyadapan non-invasif terhadap pancaran elektromagnetik.

Kelas serangan fisik lainnya melibatkan penggunaan komponen berbahaya dalam sistem tertanam. Komponen berbahaya dapat dipasang di titik lemah mana pun dalam rantai pasokan atau proses manufaktur produk tertanam atau subsistemnya. Misalnya, pada tahun 2008, pemerintah AS menyita sejumlah besar router Cisco palsu buatan Tiongkok, yang menimbulkan spekulasi bahwa router tersebut dimaksudkan untuk melemahkan keamanan internal dan mencuri informasi sensitif. Desainer tertanam harus memperhatikan asal semua bahan yang digunakan untuk membuat suatu produk, termasuk mikroprosesor, kunci kriptografi, dan perangkat lunak. Selain itu, keamanan rantai pasokan harus mencakup transportasi dan pengiriman komponen dan produk antara pemasok dan konsumen.

1.4.6 Kebijakan Khusus Aplikasi

Serangkaian kebijakan keamanan yang praktis tidak terbatas mungkin diperlukan untuk melindungi sumber daya spesifik aplikasi dari berbagai potensi ancaman. Perancang sistem harus menentukan kebijakan spesifik aplikasi yang masuk akal untuk implementasi, komponen, dan/atau proses sistem tertanam tertentu. Salah satu contoh kebijakan khusus aplikasi adalah kebijakan kontrol akses berbasis peran (RBAC) yang mendefinisikan (dan karenanya membatasi) akses sumber daya yang tepat yang diizinkan oleh proses perangkat lunak dan/atau pengguna manusia berdasarkan tanggung jawab pekerjaan mereka dan hal yang perlu diketahui. Misalnya, agregator jaringan pintar mungkin memerlukan akses jaringan jarak jauh untuk meningkatkan perangkat lunak di lapangan. Agregator berisi proses

manajemen yang perannya memungkinkan kemampuan memprogram ulang memori flash internal untuk tujuan peningkatan. Tidak ada proses lain dalam sistem yang memiliki peran yang mengizinkan akses tulis ini. Peran manajemen juga memerlukan akses ke jaringan untuk menerima permintaan manajemen. Aplikasi manajemen kemudian dapat menerapkan kebijakan khusus aplikasinya sendiri yang mengizinkan pengguna jarak jauh tertentu yang diautentikasi untuk membuat permintaan peningkatan.

1.5 ANCAMAN KEAMANAN

Seperti yang dibahas sebelumnya, kebijakan keamanan ada untuk melawan ancaman keamanan. Perancang sistem pertama-tama harus menentukan ancaman apa yang mungkin terjadi dan kemudian kebijakan keamanan apa yang masuk akal secara ekonomi dibandingkan dengan nilai sumber daya yang terkena ancaman tertentu.

Sistem yang tertanam di dalam Fort Knox dan pasukan penjaga keamanannya mungkin tidak memerlukan perlindungan terhadap serangan fisik. Sistem tertanam tanpa koneksi jaringan tidak memerlukan perlindungan terhadap serangan yang ditularkan melalui jaringan seperti badai paket, serangan pemutaran ulang protokol, dan pemeriksaan port.

Pemutar MP3 seseorang kemungkinan besar tidak akan ditargetkan oleh negara atau penyerang canggih lainnya sehingga tidak memerlukan firewall yang mahal serta sistem operasi dan kriptografi bersertifikasi NSA. Di sisi lain, sistem infrastruktur penting harus mempertimbangkan ancaman canggih seperti serangan perangkat lunak saluran samping dan serangan penolakan layanan memori/CPU.

Sistem tertanam mungkin tidak aman jika perancang sistem salah dalam menilai ancaman atau bagaimana sistem rentan terhadap ancaman tersebut. Penyimpangan tersebut terjadi karena berbagai alasan, termasuk ketidaktahuan tentang kemampuan perangkat keras yang mendasarinya, sistem operasi tertanam, atau middleware pihak ketiga. Apakah perancang sistem mengetahui jenis kriptografi (algoritma, panjang kunci, perlindungan penyimpanan untuk kunci pribadi) yang digunakan implementasi protokol Keamanan Lapisan Transportasi (TLS)? Apakah perancang mengetahui port apa yang mungkin dibiarkan terbuka oleh sistem operasi tertanam? Apakah perancang mengetahui bagaimana mikroprosesor dan boot firmware dikunci untuk menjamin pembentukan keadaan awal yang aman?

Studi Kasus: Kerentanan Port Debug VxWorks

Pada tahun 2010, kerentanan kritis dan meluas ditemukan pada sistem tertanam yang menjalankan VxWorks, sebuah sistem operasi real-time tertanam (RTOS). Contoh produk yang terkena dampak termasuk konsentrator DSL, sistem otomasi industri SCADA, sistem konferensi video, sakelar Fibre Channel, dan router Wi-Fi. Peneliti H.D. Moore menggunakan port komunikasi debug, yang tidak memiliki otentikasi kuat, untuk mengambil alih sistem yang menjalankan VxWorks. Dengan menggunakan antarmuka debug, penyerang jarak jauh dapat membaca atau menulis lokasi memori fisik mana pun: kata sandi admin dapat diekstraksi dan malware mode supervisor dapat dipasang dengan mudah. Moore mengungkapkan pada konferensi keamanan B-Sides bahwa seperempat juta perangkat yang dapat diakses langsung dari Internet ternyata rentan. Kemungkinan besar, jumlah total perangkat yang terkena dampak jauh lebih tinggi.

Sistem tertanam, yang secara historis dianggap sebagai sistem yang otonom dan tertutup, semakin menjadi bagian dari Internet of Things, dengan persyaratan untuk peningkatan di lapangan, debugging dan diagnostik jarak jauh, serta fungsi manajemen lainnya.

Pengembang tertanam yang gagal menilai ancaman secara akurat, membuat sistem, sumber daya, misi, dan jaringan mereka terkena risiko serangan. Keamanan tertanam bukan hanya tentang memilih teknologi perangkat keras dan perangkat lunak yang tepat. Yang tidak kalah pentingnya adalah desain dan arsitektur keamanan, termasuk konfigurasi, isolasi fungsi keamanan, penilaian kerentanan platform, pembentukan keadaan awal yang aman, keamanan kegagalan, dan sebagainya. Meskipun jawaban atas peretasan VxWorks ini mungkin tampak sederhana seperti menonaktifkan koneksi diagnostik atau menambahkan kemampuan otentikasi yang kuat, sering kali diperlukan pendekatan yang lebih sistemik. Port terbuka apa lagi, kelemahan buffer overflow, data yang tidak terlindungi, atau kelemahan lain yang mungkin ada di lingkungan perangkat lunak lama?

1.6 PENUTUP

Meskipun peran sistem tertanam semakin meningkat, postur keamanannya masih lemah, dan sebagian besar profesional sistem tertanam hanya memiliki pengetahuan dasar tentang masalah keamanan tertanam. Banyak profesional sistem tertanam telah bekerja dengan basis kode yang relatif kecil dan arsitektur perangkat keras yang sederhana dan berjuang untuk menghadapi tren ke arah desain yang lebih canggih, konektivitas jaringan, konsolidasi sistem, dan penyerang yang memiliki tekad dan pendanaan yang besar. Selain itu, sebagian besar profesional perangkat lunak tertanam tidak terlatih dalam seni dan ilmu merancang sistem untuk keamanan. Keamanan hanyalah sebuah renungan dan hampir tidak pernah dapat diperbaiki secara layak.

Hasilnya adalah sistem tertanam dilengkapi dengan kerentanan yang membuat infrastruktur penting terekspos. Para profesional sistem tertanam perlu dididik tentang keamanan sistem tertanam yang canggih sehingga mereka dapat mengintegrasikan dan menerapkan praktik-praktik ini untuk meningkatkan keamanan. Harapan besar penulis adalah bahwa buku ini dapat membawa komunitas pengembangan tertanam (embedded development community) untuk mengambil langkah maju yang signifikan dalam mencapai tujuan tersebut.

BAB 2
PERTIMBANGAN PERANGKAT LUNAK SISTEM

2.1 PERAN SISTEM OPERASI

Karena sistem operasi mengontrol sumber daya (misalnya memori, CPU) dari sistem tertanam, sistem operasi mempunyai kekuatan untuk mencegah penggunaan sumber daya ini tanpa izin. Sebaliknya, jika sistem operasi gagal mencegah atau membatasi kerusakan akibat akses yang tidak sah, bencana dapat terjadi. Dalam konteks sistem operasi, aktor yang tidak berwenang dapat merujuk pada aplikasi/proses, kumpulan aplikasi, dan/atau pengguna manusia yang mencoba mengakses sumber daya komputer yang tidak diizinkan oleh kebijakan keamanan sistem.

Sistem operasi menanggung beban yang sangat besar dalam mencapai keselamatan dan keamanan.

Keamanan sistem operasi bukanlah bidang penelitian baru. Namun secara praktis semua sistem operasi tertanam yang diterapkan tidak mampu memenuhi sertifikasi keamanan tingkat tinggi. Salah satu alasan kurangnya sistem operasi yang aman adalah pendekatan historis yang diambil untuk mencapai keamanan. Dalam kebanyakan kasus, keamanan hanya dikesampingkan. Namun, bahkan sistem operasi yang dirancang untuk keamanan berupaya menyediakan layanan perlindungan dan partisi, kontrol akses perangkat, sistem file yang aman, dan layanan jaringan yang aman. Akibatnya, sistem ini terlalu besar dan rumit untuk dievaluasi pada tingkat keamanan yang tinggi.

Kita semua setuju bahwa mempercayakan sistem tertanam penting kita pada sistem operasi yang tidak aman adalah ide yang buruk. Sayangnya, sebagian besar sistem komputer di dunia yang digunakan untuk memantau dan mengendalikan pabrik dan peralatan di industri seperti pengendalian air dan limbah, energi, dan penyulingan minyak menjalankan sistem operasi tersebut, sama seperti sistem operasi yang menjalankan desktop biasa. komputer. Sebagaimana dinyatakan oleh Michael Vatis, direktur eksekutif Satuan Tugas Keamanan Nasional di Era Informasi Markle Foundation, “Kerentanan ini mewabah karena kita memiliki seluruh jaringan dan infrastruktur yang dibangun di atas perangkat lunak yang tidak aman. Begitu orang luar mendapatkan akses root, dia bisa melakukan apa saja. Setiap hari, kerentanan baru akan muncul.”

2.2 BERBAGAI TINGKAT KEAMANAN INDEPENDEN

Baru-baru ini, sejumlah kecil teknologi sistem operasi tertanam telah mengambil pendekatan baru yang berupaya memecah belah dan mengatasi masalah keamanan sistem operasi. Sistem operasi ini mengadopsi arsitektur Multiple Independent Levels of Security (MILS) yang menetapkan pendekatan keamanan berlapis.

Fondasi dari sistem tertanam berbasis MILS adalah kernel pemisahan, sebuah mikrokernel kecil yang mengimplementasikan serangkaian kebijakan keamanan fungsional penting yang terbatas, termasuk isolasi data, kontrol aliran informasi, pembatasan kerusakan, dan periode pemrosesan.

Arus Informasi

Informasi tidak dapat mengalir antar aplikasi yang dipartisi kecuali diizinkan secara eksplisit oleh kebijakan keamanan sistem.

Isolasi Data

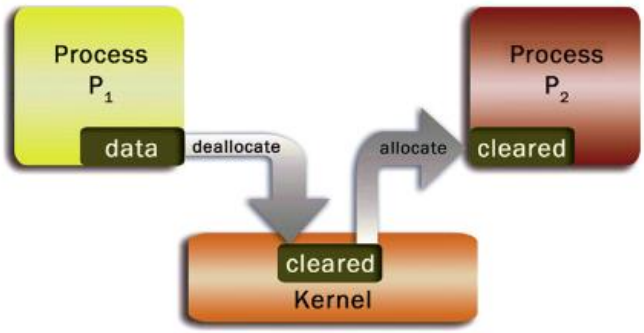
Data dalam aplikasi yang dipartisi tidak dapat dibaca atau diubah oleh aplikasi lain.

Batasan Kerusakan

Jika bug atau serangan merusak aplikasi yang dipartisi, kerusakan tersebut tidak dapat menyebar ke aplikasi lain.

Periode Pengolahan

Pemrosesan periode adalah kebijakan yang memastikan bahwa informasi dalam satu komponen tidak bocor ke komponen lain melalui sumber daya, seperti buffer memori yang dikelola kernel dan register CPU, yang dapat digunakan kembali sepanjang periode eksekusi. Misalnya, jika komponen A menyimpan informasi pribadi di halaman memori dan kemudian melepaskan halaman memori tersebut kembali ke kernel sistem operasi, kernel harus memastikan bahwa halaman tersebut dibersihkan sebelum dapat digunakan kembali oleh komponen B lain yang meminta alokasi halaman memori (lihat Gambar 2.1). Demikian pula, jika register mikroprosesor ditulis dengan data selama eksekusi A, sistem operasi harus memastikan bahwa nilai register ini dihapus ketika konteks beralih ke B pada inti yang sama. Tanpa periode pemrosesan, kerahasiaan informasi A akan dilanggar dengan pengungkapan informasi kepada B melalui sumber daya komputer tersebut.



Gambar 2.1 Contoh pemrosesan periode: sanitasi memori mencegah kebocoran informasi di seluruh domain keamanan.

Kernel pemisahan MILS mewujudkan kebijakan MILS ini dengan menggunakan perangkat keras manajemen memori mikroprosesor untuk mencegah akses tidak sah antar partisi dan dengan menerapkan mekanisme alokasi sumber daya yang mencegah operasi satu partisi mempengaruhi partisi lainnya (misalnya, dengan menghabiskan sumber daya seperti memori atau waktu CPU). Kebijakan arus informasi mencegah akses tidak sah ke perangkat dan sumber daya sistem lainnya dengan menggunakan model objek berbasis kemampuan yang efisien yang mendukung pembatasan dan pencabutan kemampuan ini ketika kebijakan keamanan sistem menganggapnya perlu.

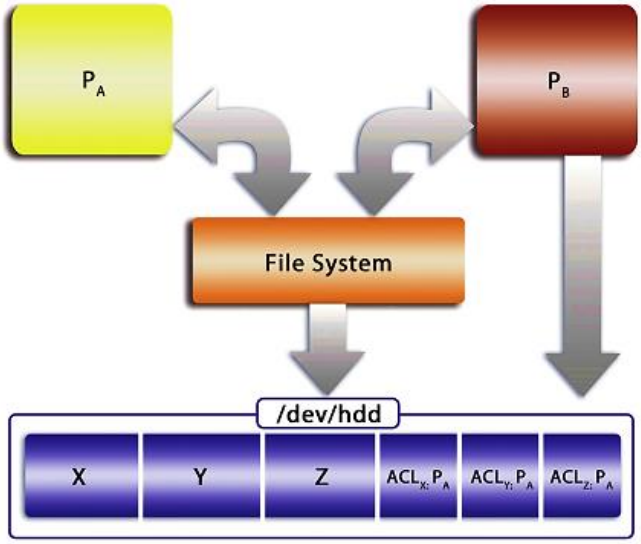
Kemampuan dan sifat pencabutan dan pengurungan yang terkait akan dibahas nanti dalam bab ini.

Arsitektur MILS juga mengadopsi monitor referensi atau mekanisme validasi referensi, yang pertama kali dijelaskan dalam laporan keamanan komputer pemerintah AS pada tahun 1970an.[1]

Kernel pemisahan dianggap sebagai monitor referensi ketika mekanisme penegakan kebijakan MILS kernel selalu digunakan, tahan terhadap kerusakan, dan dapat dievaluasi. Ketiga properti ini dijelaskan lebih lanjut pada bagian berikut.

Selalu Dipanggil

Aplikasi yang meminta sumber daya atau layanan dari sistem operasi tidak boleh melewati kebijakan keamanan sistem. Sebagai contoh, mari kita pertimbangkan sistem operasi yang menerapkan kebijakan kontrol akses berikut untuk proses A dan B pada file X, Y, dan Z: proses A diizinkan mengakses file X, Y, dan Z, namun proses B tidak diizinkan untuk mengakses file X, Y, dan Z. mengakses salah satu dari X, Y, atau Z. Namun, misalkan proses B dapat memperoleh akses ke /dev/hdd, perangkat hard disk fisik yang digunakan oleh sistem file. B dapat menggunakan perangkat fisik ini untuk melewati sistem file secara keseluruhan (lihat Gambar 2.2) dan mengakses file apa pun di disk.



Gambar 2.2 Melewati kebijakan sistem file melalui akses media langsung.

Bukti Kerusakan

Aplikasi tidak boleh merusak kebijakan keamanan atau mekanisme penegakannya. Dengan menggunakan contoh sistem file sebelumnya, kebijakan kontrol akses untuk suatu file dikodekan dalam metadata file atau dalam file kebijakan khusus yang dikelola oleh sistem file. Sekali lagi, jika diberikan akses langsung ke hard drive yang mendasarinya, B dapat merusak data kebijakan kontrol akses dan memberikan dirinya atau sekutunya akses ke file X, Y, dan Z.

Dapat Dievaluasi

Klaim keamanan sangat sedikit. Siapa pun dapat membuat klaim tentang keamanan. Pertanyaannya adalah: seberapa yakinkah pengguna akhir dan pemangku kepentingan lainnya bahwa suatu produk benar-benar memenuhi klaimnya? Keyakinan ini disebut sebagai jaminan keamanan.

[1] Andersen JP. Computer Security Planning Study, http://csrc.nist.gov/publications/history/ande72.pdf; 1972.

Arsitektur MILS memerlukan penggunaan komponen penegakan keamanan yang persyaratan fungsionalnya memenuhi tingkat jaminan yang tinggi.

Perusahaan-perusahaan bereputasi tinggi membuat klaim-klaim yang tidak berdasar mengenai keamanan namun gagal ketika menghadapi kerentanan yang ditemukan. Misalnya, mari kita lihat sertifikasi keamanan terbaru untuk hypervisor perusahaan VMware.

Seiring dengan berkembangnya penerapan virtualisasi VMware di pusat data, para pakar keamanan telah menyuarakan keprihatinan mengenai implikasi "VM sprawl" dan kemampuan teknologi virtualisasi untuk menjamin keamanan. Pada tanggal 2 Juni 2008, VMware berusaha menghilangkan kekhawatiran ini dengan mengumumkan bahwa produk hypervisornya telah mencapai sertifikasi keamanan Common Criteria Evaluated Assurance Level (EAL) 4. Siaran pers VMware mengklaim bahwa produk virtualisasinya kini dapat digunakan "untuk lingkungan pemerintahan yang sensitif dan menuntut keamanan paling ketat."

Pada tanggal 5 Juni, hanya tiga hari kemudian, kerentanan parah pada hypervisor VMware bersertifikat telah diposting ke Database Kerentanan Nasional Tim Kesiapan Darurat Komputer AS (US CERT). Di antara kendala lainnya, kerentanan tersebut "memungkinkan pengguna sistem operasi tamu untuk mengeksekusi kode arbitrer." Pada Bab 3, kita membahas standar keamanan Kriteria Umum dan tingkat jaminan relatif; namun cukuplah dikatakan bahwa EAL 4 tidak memenuhi tingkat jaminan tinggi yang diinginkan oleh MILS. Bab 3 membahas Kriteria Umum secara lebih rinci, dengan fokus pada perbedaan tingkat jaminan yang dievaluasi.

MILS menetapkan bahwa penegakan kebijakan keamanan kernel inti dapat dievaluasi hingga tingkat jaminan tertinggi. Jaminan ini sangat penting dan merupakan alasan mengapa kernel pemisahan menerapkan serangkaian kebijakan yang terfokus dan tidak menyediakan kebijakan keamanan tingkat yang lebih tinggi seperti kontrol akses berbasis peran untuk file atau keamanan jaringan. Karena evaluasi keamanan dengan jaminan tinggi memerlukan model formal dari sistem, bukti formal korespondensi antara model dan implementasi aktual, dan bukti teorema keamanan terhadap model ini, sistem yang terdiri lebih dari 10.000 baris kode menjadi terlalu sulit dan mahal. untuk mengevaluasi. Kebijakan keamanan MILS dapat diimplementasikan dengan mikrokernel yang cukup kecil untuk dievaluasi pada tingkat jaminan tertinggi.

Di bawah konsep MILS, perangkat lunak aman tingkat tinggi, seperti mekanisme komunikasi aman, server web, atau sistem file, dapat ditempatkan di atas kernel pemisahan. Kebijakan keamanan MILS bersifat rekursif: sistem file MILS, menggunakan fakta bahwa kernel pemisahan yang mendasarinya menerapkan kebijakan keamanan partisi, dapat digunakan untuk memastikan isolasi data sistem file, aliran informasi, dan properti pembatasan kerusakan. Perancang sistem dapat memilih, sesuai kebutuhan, komponen MILS yang membentuk sistem sebenarnya. Jika sistem tidak memerlukan server web yang aman, maka tidak perlu bersusah payah mengevaluasinya. Komponen MILS dapat dievaluasi secara independen pada tingkat jaminan tertinggi dan dapat berasal dari banyak vendor. Pendekatan modular terhadap keamanan ini adalah alasan utama mengapa sistem berbasis MILS telah banyak digunakan dengan biaya siklus hidup keseluruhan yang lebih rendah dibandingkan upaya awal untuk menciptakan sistem operasi terverifikasi.

Keuntungan utama lainnya dari kernel pemisahan adalah memungkinkan perangkat lunak pada berbagai tingkat kekritisan dijalankan pada satu mikroprosesor. Misalnya, aplikasi yang berisi data rahasia dan algoritme dapat menempati satu partisi sementara partisi lain tersambung ke jaringan tidak rahasia seperti Internet. Kebijakan keamanan MILS, jika sangat terjamin, memungkinkan hal ini. Hal ini dapat menghasilkan penghematan biaya yang sangat besar dalam pengembangan produk karena aplikasi multi-fungsi yang rumit dapat berjalan pada satu mikroprosesor yang kuat tanpa mengharuskan semua aplikasi tersebut dievaluasi pada tingkat jaminan tertinggi.

2.3 MIKROKERNEL VERSUS MONOLIT

Dalam keamanan komputer, istilah Basis Komputasi Tepercaya (TCB) digunakan untuk merujuk pada bagian sistem (perangkat lunak dan perangkat keras) yang penting bagi keamanan dan oleh karena itu harus dapat dipercaya.

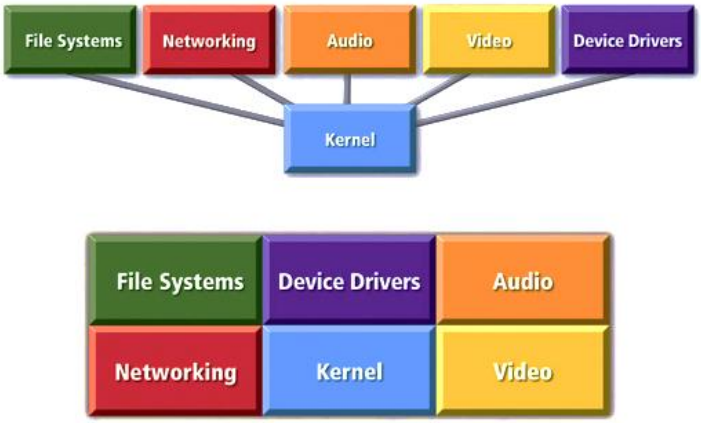
Sistem operasi monolitik berisi perangkat lunak sistem, seperti tumpukan jaringan, sistem file, dan driver perangkat kompleks yang berbagi ruang memori tunggal dan mengeksekusi dalam mode istimewa (penyelia). Hal ini menghasilkan TCB yang besar, memberikan banyak peluang bagi peretas untuk menemukan dan mengeksploitasi kerentanan. Contoh sistem operasi monolitik termasuk Windows, UNIX, Linux, dan VxWorks.

Sistem operasi mikrokernel memberikan arsitektur keamanan yang lebih baik dibandingkan sistem operasi monolitik.

Sistem operasi mikrokernel hanya menjalankan serangkaian layanan sistem penting yang minimal, seperti manajemen thread, penanganan pengecualian, dan komunikasi antar-proses, dalam mode supervisor dan menyediakan arsitektur yang memungkinkan perangkat lunak sistem yang kompleks untuk berjalan dalam mode pengguna, jika diizinkan. akses hanya ke sumber daya yang dianggap sesuai oleh perancang sistem. Kegagalan pada satu komponen tidak dapat menyebabkan kerusakan pada hard drive karena komponen yang terinfeksi tidak memiliki akses ke sumber daya tersebut. Karena mikrokernelnya sederhana, keamanannya dapat lebih mudah diverifikasi dan terjamin. Jika driver jaringan tidak menjalankan fungsi keamanan, maka driver tersebut tidak dianggap sebagai bagian dari TCB dan tidak perlu memiliki standar keamanan yang sama dengan mikrokernel atau komponen penting lainnya. Dalam sistem monolitik, driver jaringan adalah bagian dari kernel dan oleh karena itu keamanan sangat penting menurut definisinya. Gambar 2.3 menggambarkan perbedaan mencolok antara pendekatan mikrokernel dan monolitik di mana layanan sistem operasi pada umumnya terisolasi satu sama lain (dan kernel), atau tidak.

Kebanyakan sistem operasi tujuan umum bersifat monolitik karena fokus utamanya pada kinerja. Aplikasi pengguna dapat mengakses sebagian besar layanan jaringan TCP/IP, file, dan perangkat I/Od dengan panggilan sistem yang sederhana dan efisien ke dalam kernel monolitik. Sebaliknya, mikrokernel mengimplementasikan layanan ini dalam proses terpisah, memerlukan komunikasi antar-proses (IPC) antara proses permintaan dan satu atau lebih proses layanan sistem yang relevan. Misalnya, aplikasi yang ingin mengakses sistem file residen jaringan jarak jauh menggunakan Network File System (NFS) mungkin memerlukan komunikasi antara aplikasi, proses TCP/IP, proses driver perangkat antarmuka jaringan, dan proses NFS. Pekerjaan ini terjadi di balik layar: aplikasi pengguna menggunakan antarmuka

pemrograman aplikasi (API) read() atau write() yang sama dengan yang disediakan oleh sistem operasi monolitik, dan mikrokernel menangani perutean data antara proses sistem yang sesuai.



Gambar 2.3 Mikrokernel (atas) versus monolit (bawah).

Keluhan historis utama terhadap mikrokernel adalah hilangnya kinerja karena peralihan konteks ekstra dan overhead penyampaian pesan dari arsitektur proses layanan ini. Ada dua tren yang telah menghilangkan keluhan ini. Pertama, mikroprosesor yang lebih cepat telah mengurangi dampak overhead IPC. Kedua, pengembang mikrokernel komersial sudah mahir dalam mengoptimalkan IPC. Misalnya, Jochen Liedtke, penulis asli mikrokernel L4 open source, menerbitkan penelitian yang menunjukkan bahwa IPC mikrokernel dapat ditingkatkan secara dramatis melalui desain dibandingkan upaya generasi pertama sejak tahun 1980an.[2] Namun, bukti yang paling meyakinkan adalah bahwa mikrokernel telah menjadi praktis adalah penerapan tertanam yang sukses secara luas menggunakan mikrokernel sejak tahun 2000. Mikrokernel menjalankan hampir semua jenis produk tertanam, termasuk ponsel pintar, avionik, peralatan jaringan, sistem kontrol proses, dan perangkat medis. Contoh sistem operasi tertanam berbasis mikrokernel termasuk INTEGRITY, LynxSecure, Neutrino/QNX, OKL4, PikeOS, dan Symbian.

Selama bertahun-tahun, perancang sistem operasi telah menyatakan perbedaan pendapat, terkadang dengan lantang, mengenai manfaat relatif dari dua pendekatan utama dalam desain sistem operasi ini. O'Reilly menerbitkan bagian dari perdebatan terkenal antara Linus Torvalds, pendiri Linux, dan Andrew Tanenbaum, seorang profesor dan peneliti sistem operasi, yang dimulai pada tahun 1992.[3] Sayangnya, sebagian besar perdebatan ini gagal untuk fokus pada inti arsitektur. dampak desain sistem operasi pada keamanan. Sebaliknya, perdebatan telah meluas ke isu-isu perizinan open source, patching protokol, dan, terkadang, penghinaan pribadi. Meskipun Linux telah dan akan terus diadopsi secara besar-besaran karena lisensi open source dan model pengembangannya, jelas bagi semua pembaca bahwa mikrokernel berbasis komponen lebih unggul dalam hal keamanan dan ketahanan. Bahkan Torvalds mengakui hal tersebut dalam salah satu emailnya kepada Tanenbaum: “Benar, Linux

[2] Liedtke J. Toward Real Microkernels. Communications of the ACM September 1996;39(9):70e7.
[3] DiBona C, Stone S, Ockman M, editors. Appendix A, Open Sources: Voices from the Open Source Revolution. Sebastopol, CA: O’Reilly; 1999.

itu monolitik, dan saya setuju bahwa mikrokernel lebih bagus. Dari sudut pandang teoretis (dan estetika), Linux kalah." Praktis setiap peneliti sistem operasi universitas terkemuka seperti Andrew Tanenbaum, Gernot Heiser, Jochen Liedtke, Jonathan Shapiro, Kang Shin, Ken Thompson, Rajkumar Buyya, dan banyak lainnya telah menegaskan keunggulan arsitektur dari pendekatan mikrokernel.

Lebih jauh lagi, seperti yang akan kita bahas nanti di bab ini, keputusan untuk menggunakan sistem operasi mikrokernel atau monolitik tidak selalu eksklusif; virtualisasi memberikan kemampuan untuk menggabungkan implementasi terbaik dari kedua arsitektur dalam satu sistem.

Studi Kasus: Virus Duqu

Pada bulan November 2011, peneliti keamanan menemukan virus komputer berbahaya yang diberi nama Duqu. Mirip dengan serangan Stuxnet yang dijelaskan di Bab 1, Duqu diyakini telah ditulis oleh penyerang canggih dan memanfaatkan kerentanan Windows zero-day untuk mendapatkan akses ke infrastruktur kontrol kritis yang tertanam. Seperti yang dijelaskan dalam penasihat keamanan Microsoft: "Penyerang yang berhasil mengeksploitasi kerentanan ini dapat menjalankan kode arbitrer dalam mode kernel. Penyerang kemudian dapat menginstal program; melihat, mengubah, atau menghapus data; atau buat akun baru dengan hak pengguna penuh. Kami mengetahui adanya serangan yang ditargetkan yang mencoba memanfaatkan kerentanan yang dilaporkan."[4] Basis Data Kerentanan Nasional telah menetapkan tingkat keparahan kerentanan sebesar 9,3 (tinggi),[5] terutama karena kemampuan eksploitasi jarak jauh dan dampak yang ditimbulkan oleh akses root yang berbahaya.

Menurut laporan online, dan dikuatkan oleh ikhtisar entri Database Kerentanan Nasional yang disebutkan di atas, sistem operasi Windows yang umum (XP, Vista, Windows 7, Windows Server) semuanya menjalankan mesin pengurai font di kernel. Faktanya, penguraian font terjadi di win32k.sys, modul driver perangkat sistem Windows yang sangat besar yang telah menjadi penyebab banyak "layar biru kematian" (BSOD) selama bertahun-tahun.

Penulis Duqu membuat dokumen Word yang akan mengeksploitasi kerentanan penguraian font, memungkinkan eksekusi kode jarak jauh dengan hak pengguna super. Seperti yang telah kita pelajari dari banyak kerentanan seperti ini (Stuxnet hanyalah salah satu contoh kasus lainnya), begitu penyerang memiliki kendaraan malware yang efektif seperti ini, maka akan mudah untuk merekayasa target serangan secara sosial agar dapat membuka file tersebut. Penyerang hanya memerlukan satu intrusi titik akhir yang berhasil; PC yang terinfeksi kemudian digunakan sebagai titik peluncuran terhadap perangkat elektronik dan komputer yang dapat diakses oleh PC secara lokal dan melalui jaringan.

Duqu memberikan contoh bagus tentang dampak buruk dari desain sistem operasi monolitik. Dalam beberapa kasus, win32k.sys terdiri dari jutaan byte kode driver perangkat yang dijalankan dalam mode kernel/superuser.

---

[4] Microsoft Security Advisory (2639658). Vulnerability in TrueType Font Parsing Could Allow Elevation of Privilege, http://technet.microsoft.com/en-us/security/advisory/2639658; November 3, 2011.

[5] US-CERT/NIST National Vulnerability Database. Vulnerability Summary for CVE-2011-3402, http://web.nvd.nist.gov/view/vuln/detail?vulnId¼CVE-2011-3402.

2.4 PERSYARATAN KEAMANAN SISTEM OPERASI INTI TERTANAM

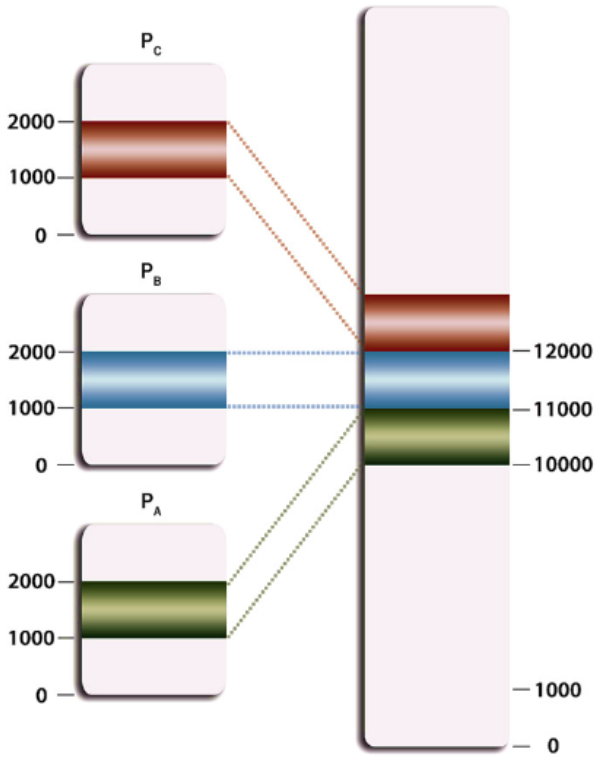
Penggunaan arsitektur mikrokernel, tentu saja, hanyalah salah satu bagian penting dari kisah keamanan sistem operasi tertanam. Pada bagian berikut, kami membahas fitur teknis utama sistem operasi tertanam yang berdampak pada keamanan.

Perlindungan memori merupakan persyaratan mendasar untuk sistem tertanam yang kuat.

Sistem operasi menggunakan perangkat keras perlindungan memori mikroprosesor untuk mengisolasi fungsi yang tidak terkait. Komponen yang dilindungi sering disebut proses. Aplikasi perangkat lunak dapat dipetakan ke satu atau lebih proses sistem operasi. Dengan perlindungan memori, kode berbahaya tidak dapat membuat aplikasi atau sistem operasi crash dengan merusak memorinya. Demikian pula, penunjuk yang salah yang disebabkan oleh kesalahan pemrograman dalam satu aplikasi tidak dapat mempengaruhi aplikasi lain yang dijalankan dalam proses terpisah yang dilindungi memori.

2.4.1 Memori Virtual

Sebagian besar sistem operasi modern mendukung memori virtual yang alamat kode proses dan datanya bersifat logis, bukan fisik. Kernel menggunakan perangkat keras manajemen memori untuk memetakan memori logis ke memori fisik. Sebagai contoh, mari kita pertimbangkan tiga proses: A, B, dan C, yang masing-masing memerlukan penyimpanan sebesar empat kilobyte. Dalam sistem memori virtual, A, B, dan C mungkin menempati rentang alamat logis yang sama, katakanlah 0x1000 hingga 0x2000, dalam ruang alamat virtual terpisah. Sistem operasi memetakan blok empat kilobyte ini ke lokasi fisik, misalnya 0x10000, 0x11000, dan 0x12000 yang tersembunyi dari proses itu sendiri (lihat Gambar 2.4).



Gambar 2.4 Tiga proses yang dilindungi memori virtual.

Beberapa sistem operasi tertanam, seperti OSE dan VxWorks, menggunakan perlindungan memori tanpa memori virtual. Dalam contoh A, B, dan C, sistem operasi model memori datar dapat menempatkan proses ini di alamat fisik 0x10000, 0x11000, dan 0x12000. Alamat-alamat ini dapat dilihat oleh program; kode dan objek data dihubungkan dan dimuat dalam rentang fisik ini.

Memori virtual menyediakan fitur keamanan tambahan, termasuk halaman penjaga dan pengaburan lokasi, selain perlindungan memori dasar.

Halaman Penjaga

Salah satu keuntungan besar dari memori virtual adalah kemampuan untuk memetakan dan menghapus peta halaman secara selektif ke dalam ruang alamat virtual. Tentu saja, halaman memori fisik dipetakan untuk menyimpan kode dan data aplikasi. Selain itu, halaman dipetakan untuk menampung tumpukan runtime proses aplikasi.

Sistem operasi harus menyediakan kemampuan untuk membiarkan alamat virtual senilai satu halaman (atau lebih) di bawah setiap area tumpukan tidak dipetakan sehingga jika suatu proses melebihi tumpukannya, kesalahan perlindungan memori perangkat keras akan dipicu. Kernel akan menangguhkan proses alih-alih membiarkannya merusak area memori penting lainnya dalam ruang alamat. Dalam model memori datar, setiap halaman memori fisik mewakili RAM (atau ROM) sebenarnya, jadi membuka peta halaman secara efektif menyebabkan potongan RAM tersebut, yang biasanya terbatas pada sistem tertanam, hilang. Perlindungan memori, termasuk deteksi stack overflow semacam ini, juga sangat berguna selama pengembangan aplikasi. Kesalahan pemrograman sering kali menghasilkan pengecualian yang langsung terdeteksi dan mudah dilacak ke kode sumber. Tanpa perlindungan memori, bug dapat menyebabkan kerusakan halus yang sangat sulit dilacak. Faktanya, pada beberapa mikroprosesor, RAM sering kali terletak di alamat fisik 0, sehingga dereferensi penunjuk NULL pun bisa tidak terdeteksi!

Kebingungan Lokasi

Manfaat keamanan lain dari memori virtual adalah menyembunyikan lokasi fisik kode dan data, sehingga mempersulit penyerang untuk membedakan operasi atau keadaan internal. Misalnya, penyerang mampu mengintip bus alamat fisik. Dengan memperhatikan permintaan alamat, penyerang dapat menentukan proses mana yang sedang dijalankan dan mendapatkan wawasan tentang perilaku proses tersebut. Dalam sistem memori virtual, referensi ke alamat fisik tidak dapat dipetakan kembali ke suatu proses tanpa terlebih dahulu membahayakan struktur data internal sistem operasi.

2.4.2 Pemulihan Kesalahan

Faktanya adalah ada bug dalam perangkat lunak, dan kegagalan akan terjadi. Kebanyakan programmer mungkin pernah melihat statistik yang merinci jumlah rata-rata bug per baris kode sumber produksi. Dan ketika aplikasi menjadi lebih kompleks, melakukan lebih banyak fungsi di dunia yang haus akan teknologi, jumlah bug aktual dalam sistem yang diterapkan akan terus meningkat.

Perancang sistem harus merencanakan kegagalan dan menerapkan teknik pemulihan kesalahan.

Tentu saja, pemulihan kesalahan bergantung pada aplikasi: respons terhadap program pengumpulan statistik yang menghasilkan data yang salah akan berbeda dengan respons terhadap koneksi jaringan yang gagal. Namun sistem operasi harus menyediakan beberapa fitur mendasar yang memungkinkan perancang sistem membangun toleransi kesalahan dan ketersediaan tinggi.

Ketika suatu proses mengalami kesalahan (misalnya, karena akses memori yang tidak valid), kernel harus menyediakan beberapa mekanisme dimana pemberitahuan dapat dikirim ke agen yang bertugas melakukan beberapa jenis tindakan pemulihan kesalahan. Proses supervisor ini harus dapat berjalan di ruang alamatnya sendiri karena data di ruang alamat yang berisi proses yang salah mungkin rusak. Kernel harus menyediakan mekanisme yang memungkinkan proses supervisor untuk menutup proses yang salah. Kernel harus menyediakan mekanisme untuk memulai ulang aplikasi. Kernel harus menyediakan mekanisme pencatatan peristiwa sehingga penyebab kegagalan dapat ditentukan dengan memeriksa segala sesuatu yang terjadi di sistem, seperti panggilan layanan kernel, peralihan konteks proses, dan interupsi, sebelum kesalahan terjadi. Kernel harus menyediakan kemampuan pengawas perangkat lunak dimana proses pengawas dapat diberitahu ketika proses periodik tidak menjalankan urutan kode yang diharapkan; ini penting karena beberapa kegagalan mungkin tidak secara langsung menyebabkan pengecualian perangkat keras.

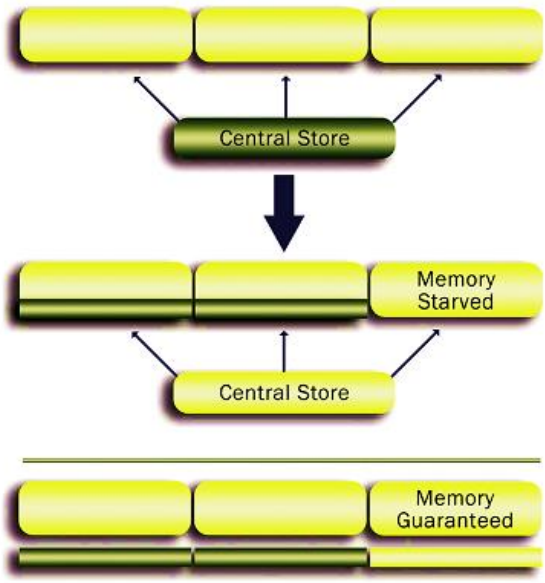
Kernel juga harus melindungi dirinya terhadap panggilan layanan yang tidak tepat. Banyak kernel yang meneruskan pointer sebenarnya ke objek kernel yang baru dibuat, seperti semaphore, kembali ke aplikasi sebagai pegangan dan kemudian melakukan dereferensi pointer ini ketika diteruskan ke panggilan layanan kernel berikutnya yang dibuat oleh aplikasi. Aplikasi dapat meneruskan pointer yang tidak valid dengan hasil yang buruk. Panggilan layanan kernel tidak boleh diizinkan untuk menghapus kernel. Sistem operasi harus menggunakan deskriptor buram untuk referensi aplikasi ke objek kernel. Kernel harus memvalidasi semua parameter panggilan layanan.

2.4.3 Sumber Daya Terjamin

Meskipun ada perlindungan memori dan memori virtual, kode berbahaya masih dapat menghapus aplikasi penting dengan menghabiskan sumber dayanya.

Sebagian besar sistem operasi menggunakan penyimpanan pusat untuk sumber daya memori. Ketika proses yang dilindungi meminta sumber daya baru (misalnya memori heap, thread), sistem operasi mengalokasikan sumber daya ini dari penyimpanan pusat. Aplikasi yang salah atau berbahaya dapat meminta terlalu banyak sumber daya, menyebabkan penyimpanan pusat habis dan aplikasi penting gagal saat mencoba mendapatkan sumber daya yang diperlukan. Serangan penolakan layanan serupa dapat terjadi seiring dengan waktu CPU. Proses berbahaya dapat memunculkan beberapa proses “konfederasi” yang menghabiskan waktu CPU, sehingga menghambat aplikasi penting untuk menyelesaikan tugasnya.

Dimungkinkan untuk merancang sistem operasi sedemikian rupa sehingga sumber daya penting ini dipartisi sesuai dengan kebutuhan aplikasi individual. Proses yang terinfeksi dapat menghabiskan kuota sumber dayanya sendiri, namun tidak mungkin mempengaruhi kuota sumber daya yang dimiliki oleh aplikasi penting. Perbedaan antara pendekatan penyimpanan pusat dan kuota terhadap alokasi memori digambarkan pada Gambar 2.5.



Gambar 2.5 Penyimpanan pusat versus kuota untuk memori.

Di bagian atas diagram, dua aplikasi membuat permintaan alokasi memori hingga seluruh penyimpanan pusat habis, sehingga membuat aplikasi ketiga kelaparan. Di bagian bawah diagram, setiap aplikasi memiliki kuota yang tidak dapat dipengaruhi oleh aplikasi lain.

Penting untuk dipahami bahwa jaminan partisi sumber daya tidak praktis untuk dipasang pada sistem operasi yang tidak dirancang untuk itu. Misalnya, relatif mudah untuk melakukan retrofit sistem kuota dasar untuk memori, file, dan sumber daya lainnya. Faktanya, UNIX dan Linux menyediakan sistem seperti itu, yang disebut rlimit, yang pertama kali ditambahkan ke Berkeley Software Distribution (BSD) UNIX pada versi 4.2, sekitar tahun 1983. Masalah dengan pendekatan bolt-on seperti rlimit adalah gagal memperhitungkan sumber daya yang dialokasikan secara internal oleh kernel atas nama prosesnya. Misalnya, kernel harus menggunakan memori fisik untuk penyimpanan internal blok kontrol objek, daftar penjadwalan, tabel halaman memori virtual, buffer driver perangkat, dan sumber daya lainnya. Sumber daya ini tersebar di seluruh kernel, dan UNIX rlimit gagal mengatribusikan alokasinya ke proses tertentu. Dengan demikian, masing-masing sumber daya bersama ini berpotensi menjadi kerentanan penolakan layanan.

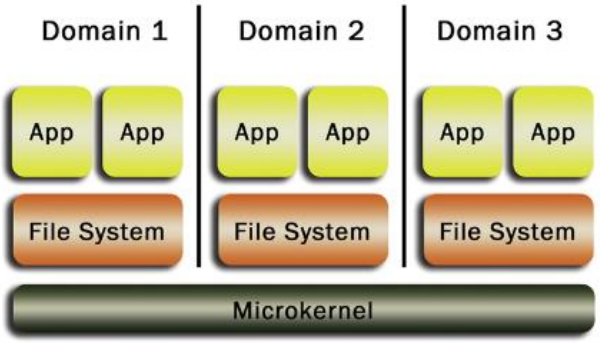
Eksploitasi penolakan layanan yang sederhana dari sistem operasi UNIX yang dilengkapi rlimit adalah fork bomb: menelurkan sejumlah besar proses untuk menghabiskan tabel proses internal kernel. Bahkan jika sistem menerapkan batas jumlah proses per pengguna, malware yang menyusup ke akun beberapa pengguna masih dapat membuat sistem tidak berfungsi, sehingga mencegah peluncuran aplikasi lain. Sebagian besar pengguna sistem UNIX atau Linux pernah mengalami pesan frustasi “Tidak ada proses lagi.”

Selain itu, tabel proses hanyalah salah satu sumber daya bersama yang terpengaruh oleh fork bomb. Operasi fork mahal dalam hal CPU dan bandwidth memori, dan tindakan mengeksekusi sejumlah besar fork secara bersamaan sudah cukup untuk mencegah kemajuan aplikasi penting.

Sistem operasi partisi yang aman tidak pernah mengalokasikan sumber daya secara dinamis dari kumpulan bersama. Misalnya, suatu proses memuat pustaka tautan dinamis (DLL). DLL dipetakan ke dalam ruang alamat virtual proses, memerlukan pemetaan halaman logis ke fisik baru yang harus dikelola oleh kernel. Manajemen ini mungkin memerlukan penyimpanan untuk akuntansi internal; memori tersebut harus diatribusikan dan dikurangi dari kuota proses yang memulai permintaan layanan. Partisi yang aman adalah landasan dari kernel pemisahan yang sesuai dengan MILS.

Penting juga untuk dicatat bahwa menggunakan mikrokernel partisi yang aman tidak selalu berarti bahwa proses host mikro mana pun yang menyediakan layanannya sendiri, seperti komunikasi jaringan atau manajemen file, juga akan memberikan jaminan sumber daya kepada kliennya. Perancang sistem tertanam kemungkinan akan dihadapkan pada pilihan server proses, beberapa di antaranya menyediakan partisi aman dan beberapa tidak. Misalnya, mikrokernel mungkin meng-host sistem file yang kompatibel dengan UNIX yang tidak menerapkan partisi aman pada struktur data internal atau media sistem file fisik. Sistem file yang kompatibel dengan UNIX mungkin berguna untuk portabilitas dan kompatibilitas dengan sistem komputer lain, namun bisa menjadi sumber masalah penolakan layanan jika layanan sistem file dapat diakses oleh proses kritis dan berbahaya.

Sekali lagi, mikrokernel yang aman dapat membantu: beberapa contoh sistem file yang kompatibel dengan UNIX dapat dimasukkan dalam arsitektur sistem secara keseluruhan; setiap instance melayani domain keamanan yang terisolasi (lihat Gambar 2.6). Aplikasi penting dalam satu domain dilindungi dari serangan penolakan layanan dari perangkat lunak berbahaya di domain terpisah (misalnya, domain yang terhubung ke Internet). Server multi-instance tidak perlu mahal dalam hal sumber daya memori; mikrokernel yang aman harus menawarkan kemampuan untuk berbagi bagian server file yang statis dan hanya bisa dibaca (termasuk semua kode yang dapat dieksekusi) di seluruh domain.



Gambar 2.6 Arsitektur sistem file multi-domain.

Potensi kelemahan lain dari sistem kuota adalah sifatnya yang statis. Beberapa sistem yang kompleks mungkin memiliki kebutuhan sumber daya yang berbeda-beda selama masa pakainya dan mungkin tidak memiliki cukup memori untuk menggabungkan semua kuota maksimum. Untuk mengatasi hal ini, sistem operasi partisi harus memberikan kemampuan bagi perancang sistem untuk menciptakan manajer sumber daya yang dapat dipercaya yang dapat mengekspor sumber daya ke, dan mengambil kembali sumber daya dari, aplikasi sesuai

permintaan. Perancang sistem harus dapat menentukan aplikasi penting mana yang dapat terhubung ke manajer sumber daya.

Perangkat lunak berbahaya yang diimpor ke proses baru tidak diberi akses ke pengelola sumber daya dan diatur oleh kuota.

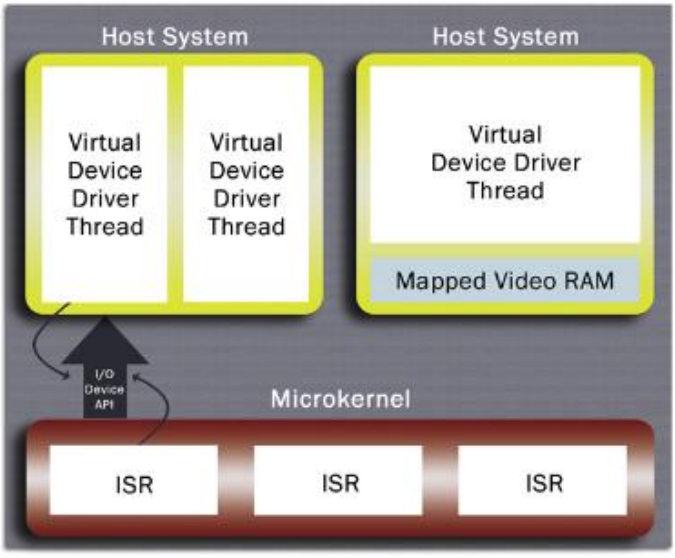
Alternatifnya, perancang sistem dapat menggunakan beberapa pengelola sumber daya, satu untuk setiap domain keamanan. Aplikasi dalam domain keamanan bebas berbagi sumber daya spesifik domainnya yang tidak dapat dibajak oleh aplikasi apa pun yang berjalan di domain keamanan berbeda.

Partisi, ditambah dengan kemampuan untuk menentukan pengelola sumber daya dan beberapa server contoh, memungkinkan perancang sistem memperoleh fleksibilitas jika diperlukan dan keamanan jika diperlukan.

2.4.4 Driver Perangkat Virtual

Driver perangkat sering kali menjadi penyebab masalah keandalan sistem dan menjadi sasaran peretas; dengan demikian, driver perangkat adalah beberapa komponen terpenting dari sistem operasi yang harus diisolasi dan dilindungi.

Sistem operasi monolitik biasanya memungkinkan pengguna dan proses menginstal driver perangkat secara dinamis ke dalam kernel. Driver perangkat yang salah dapat mengalami serangan buffer overflow di mana malware menimpa tumpukan runtime untuk menginstal kode ke dalam kernel. Driver perangkat virtual mencegah jenis serangan ini karena driver perangkat yang disusupi hanya dapat membahayakan proses yang berisi driver tersebut, bukan kernel itu sendiri. Untuk memfasilitasi pengembangan driver perangkat virtual, sistem operasi perlu menyediakan mekanisme yang fleksibel untuk kontrol I/O pada proses driver virtual. Namun, driver virtual harus diberikan akses hanya ke sumber daya perangkat tertentu yang dibutuhkan driver untuk mencapai fungsi yang diinginkan. Gambar 2.7 menunjukkan beberapa contoh driver perangkat virtual, driver komunikasi berbasis interupsi yang terdiri dari dua thread, dan driver perangkat grafis yang hanya menulis ke RAM video yang dipetakan tanpa memerlukan interupsi apa pun.



Gambar 2.7 Driver perangkat virtual.

2.4.5 Dampak Determinisme

Sistem operasi melakukan berbagai layanan atas nama proses kliennya dan sistem tertanam secara keseluruhan. Sistem operasi menjadwalkan beban kerja pada satu atau lebih inti CPU, mengalokasikan memori, mengacak pesan antar proses, mengontrol akses ke I/O, mengelola konsumsi daya, memantau kesehatan sistem, dan sebagainya. Ada banyak kelonggaran dalam hal bagaimana sistem operasi dirancang untuk memprioritaskan semua aktivitas ini. Sebagian besar sistem operasi tujuan umum dirancang untuk memberikan respons yang baik atau adil kepada semua kliennya. Meskipun proses dapat diberi prioritas numerik, proses tersebut ditafsirkan sebagai pedoman yang tidak mengganggu kemampuan sistem lainnya untuk membuat kemajuan.

Sistem tertanam real-time adalah sistem yang ketidakmampuannya merespons dalam jangka waktu maksimum tertentu merupakan kegagalan. Misalnya, komputer kendali fly-by-wire pesawat harus membaca dan memproses sejumlah besar sensor secara berkala. Jika aplikasi kontrol tidak dapat membaca dan memproses pembacaan sensor secara tepat waktu, keselamatan penerbangan terancam. Pembom B-2 adalah contoh yang bagus: struktur sayap terbangnya pada dasarnya tidak stabil; sistem fly-by-wire mengatasi ketidakstabilan ini dengan penyesuaian kontrol sepersekian detik secara terus menerus, terlepas dari pilot, berdasarkan masukan yang diterima dari sensor. Apa yang disebut soft real time hanya mengacu pada ketidakmampuan untuk memberikan jaminan mutlak (dengan kata lain, bukan real time).

Sistem operasi real-time (RTOS) mampu menjamin waktu eksekusi kasus terburuk absolut (WCET) untuk layanan, seperti penanganan interupsi periferal. Oleh karena itu, sistem tertanam waktu nyata harus dikontrol oleh RTOS. Untuk menjadi RTOS sejati, operasi kernel harus sepenuhnya deterministik: WCET untuk setiap layanan kernel harus dibatasi dan dapat dihitung.

Sulit untuk membangun sistem operasi yang sepenuhnya deterministik. Sebagai contoh, mari kita lihat bagaimana kernel sistem operasi menangani interupsi yang diakibatkan oleh kedatangan pesan I/O yang tidak sinkron, berakhirnya waktu, dan kejadian lainnya. Interupsi dapat terjadi kapan saja, termasuk ketika kernel sedang melakukan beberapa operasi kritis. Ketika suatu proses menjalankan panggilan sistem, dan kernel memodifikasi struktur data internal untuk menyelesaikan permintaan, preemption interupsi dinonaktifkan oleh kernel untuk memastikan bahwa proses lain tidak dapat dialihkan konteksnya untuk menjalankan panggilan sistem yang dapat mengakses struktur data internal ini. sementara mereka berada dalam kondisi yang tidak konsisten. Urutan kode yang tidak dapat diinterupsi ini disebut bagian kritis. Kernel dapat mengklaim determinisme hanya dengan mampu menjamin WCET yang dapat dihitung untuk setiap bagian kritis.

Bahkan sistem operasi real-time sederhana yang dijelaskan sendiri dapat menonaktifkan pemrosesan interupsi di ratusan lokasi. Dalam sebuah studi tentang latensi interupsi sistem operasi real-time,[6] peneliti memperkirakan bahwa kernel berisi sekitar 1.200 bagian penting, yang mana para peneliti mampu menghitung WCET secara statis sekitar setengahnya, menyisakan sekitar 600 rangkaian penonaktifan yang tidak diketahui secara

[6] Carlsson M. Worst Case Execution Time Analysis, Case Study on Interrupt Latency, for the OSE Real-Time

statis. WCET yang dihitung. Dari rangkaian yang dapat dihitung, para peneliti menemukan loop bersarang rangkap tiga dengan perkiraan WCET lebih dari 25.000 siklus.

RTOS yang dianalisis dalam penelitian ini jauh lebih rumit dibandingkan sistem operasi tujuan umum seperti Windows atau Linux. Meskipun terdapat upaya yang signifikan untuk mengurangi latensi umum pada banyak operasi, kemampuan kernel monolitik ini untuk menjamin determinisme masih belum dapat dilakukan.

Mengapa semua ini penting bagi keamanan? Dampak paling jelas dari determinisme adalah kemampuan sistem tertanam real-time untuk menjamin integritas dan ketersediaan fungsinya dengan menghindari tenggat waktu yang terlewat yang dapat menyebabkan kegagalan sistem. Namun, hanya sebagian kecil dari semua sistem tertanam yang mempunyai tenggat waktu real-time yang sulit.

Determinisme diperlukan untuk menegakkan partisi waktu yang aman.

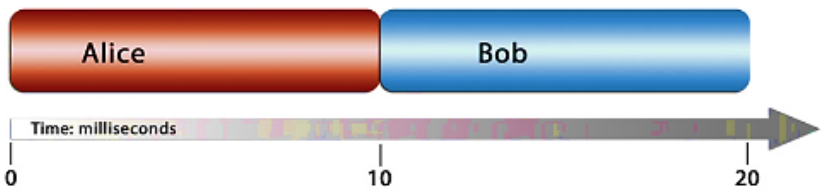
Aplikasi yang kritis terhadap keamanan tidak boleh kekurangan waktu eksekusi oleh aplikasi yang berbahaya atau salah. Jika aplikasi jahat menemukan kelemahan di mana panggilan layanan sistem operasi tertentu membutuhkan waktu lama untuk dijalankan dalam keadaan tertentu, maka panggilan layanan ini dapat disalahgunakan oleh aplikasi jahat untuk menunda aplikasi penting.

Selain itu, variasi waktu eksekusi panggilan layanan dapat digunakan sebagai saluran pengaturan waktu terselubung atau sampingan, yang merupakan contoh serangan keamanan yang canggih.

Saluran samping adalah mekanisme yang digunakan entitas jahat untuk menyimpulkan informasi sensitif tentang proses penting hanya dengan mengamati variasi fisik yang disebabkan oleh proses tersebut.

Variasi fisik yang dapat diamati meliputi waktu (saluran waktu), konsumsi daya, pancaran elektromagnetik, dan suhu. Kelemahan, ancaman, dan tindakan penanggulangan saluran samping sering kali dianalisis dalam sistem kripto, yang akan kita bahas lebih lanjut di Bab 4.

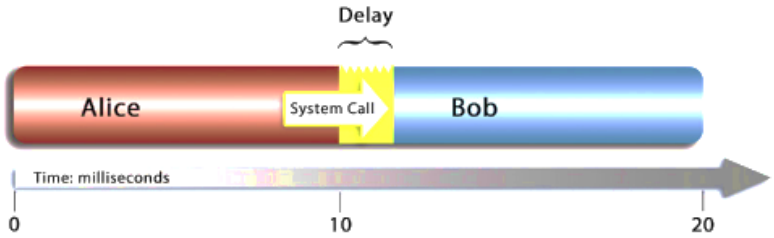
Saluran rahasia juga mengeksploitasi variasi fisik tetapi mengasumsikan dua atau lebih pihak yang berkolusi menggunakan efek ini untuk menciptakan saluran komunikasi tidak sah untuk diri mereka sendiri. Saluran waktu terselubung menggunakan variasi waktu sebagai sarana komunikasi. Misalkan dua aplikasi yang terisolasi namun berkolusi, Alice dan Bob, masing-masing diberi waktu eksekusi 10 milidetik dalam jadwal berulang 20 milidetik (lihat Gambar 2.8). Tugas sistem operasi adalah memastikan Alice dan Bob mendapatkan waktu eksekusi tepat 10 milidetik. Kernel akan menyetel pengatur waktu untuk kedaluwarsa setiap 10 milidetik dan mengalihkan konteks Alice dan Bob ketika interupsi pengatur waktu diaktifkan.



Gambar 2.8 Jadwal partisi sederhana.

Jika Alice dapat meningkatkan waktu eksekusinya untuk sementara di atas 10 milidetik dengan menunda saklar konteks, maka Bob akan mengetahui hilangnya waktu eksekusinya. Dengan demikian, Alice dapat mengirimkan sedikit informasi kepada Bob untuk setiap transit melalui jadwal: satu untuk jendela eksekusi 10 milidetik dan nol untuk jendela sub-10 milidetik. Ini adalah saluran rahasia dengan bandwidth tinggi: kunci enkripsi pribadi 128-bit terungkap dalam waktu kurang dari tiga detik.

Alice dapat menggunakan panggilan layanan sistem operasi sebagai sarana untuk menunda peralihan konteks. Panggilan layanan bertindak sebagai bagian penting yang menunda penjadwal beralih ke Bob. Dengan demikian, Alice dapat mengeksekusi panggilan sistem di dekat akhir jendela eksekusinya untuk mencapai penundaan ini (lihat Gambar 2.9).



Gambar 2.9 Penggunaan panggilan layanan sistem operasi yang berbahaya untuk melakukan serangan waktu antar partisi.

Penanggulangan yang jelas terhadap serangan waktu ini adalah dengan menempatkan buffer waktu antara Alice dan Bob sama dengan panjang panggilan layanan terlama yang mungkin dilakukan. Jadi, apapun panggilan sistem yang dicoba, Alice tidak dapat menunda dimulainya partisi waktu Bob.

Inilah sebabnya mengapa determinisme relevan dengan keamanan. Jika Alice dapat menemukan panggilan sistem yang, dalam keadaan tertentu, dijalankan lebih lama dari yang diharapkan, maka Alice dapat menggunakan panggilan sistem tersebut sebagai saluran waktu rahasia. Tidak semua sistem tertanam perlu khawatir tentang ancaman saluran waktu, namun mereka yang perlu khawatir harus mendapatkan analisis WCET dari vendor sistem operasi. Selain itu, beberapa evaluasi dan sertifikasi keamanan memerlukan analisis yang cermat terhadap saluran rahasia dari semua sumber fisik yang berlaku dalam sistem tertanam. Dengan mendeteksi dan mengukur bandwidth saluran rahasia, perancang dapat menentukan kebutuhan dan/atau kelayakan tindakan penanggulangan tertentu.

2.4.6 Penjadwalan Aman

Pengembang sistem tertanam yang akrab dengan sistem operasi waktu nyata akan memperhatikan bahwa pendekatan penjadwalan yang dijelaskan di bagian sebelumnya tidak seperti penjadwal preemptive berbasis prioritas yang menggunakan prioritas yang ditetapkan pengembang untuk Alice dan Bob. Misalkan Alice diberi prioritas 100, lebih tinggi dari prioritas Bob yaitu 50. Selama Alice terus mengeksekusi, Alice berjalan di depan Bob (dengan asumsi satu inti pemrosesan). Penjadwalan berbasis prioritas gagal menyediakan partisi waktu yang aman: Alice dapat dengan mudah membuat Bob kehabisan waktu eksekusi dengan memutar, dan Alice serta Bob dapat memodulasi penggunaan CPU mereka untuk membentuk saluran waktu rahasia dengan bandwidth tinggi.

Pendekatan jendela eksekusi yang dijelaskan di bagian sebelumnya dikenal sebagai penjadwalan partisi. Tergantung pada implementasinya, partisi waktu dapat mencakup satu proses atau kumpulan proses terkait yang dapat berbagi sebagian waktu CPU dengan aman. Sistem operasi dapat mengizinkan perancang sistem untuk menggunakan prioritas numerik atau parameter penjadwalan lainnya untuk mengontrol bagaimana jendela waktu tersebar di seluruh proses atau thread penyusun partisi.

Dimungkinkan juga untuk menetapkan proses tepercaya tertentu ke lebih dari satu jendela eksekusi. Ada berbagai bentuk penjadwalan partisi; dari perspektif keamanan, pertimbangan penting adalah bahwa proses yang tidak dipercaya tidak dapat mengubah jadwal yang ditentukan sistem. Partisi waktu yang aman adalah fitur penting dari kernel pemisahan; tidak semua mikrokernel menyediakan partisi waktu yang aman.

Masalah yang melekat pada penjadwal thread RTOS adalah bahwa mereka tidak mengetahui proses atau ruang alamat di mana thread berada. Misalkan Alice mengeksekusi dalam proses pengumpulan statistik sementara thread kritis yang dijalankan Bob dalam proses pemrosesan panggilan. Kedua aplikasi tersebut dipartisi dan dilindungi dalam domain ruang, tetapi tidak dalam domain waktu. Perancang sistem yang aman memerlukan kemampuan untuk menjamin bahwa karakteristik runtime dari aplikasi pengumpulan statistik tidak mungkin mempengaruhi karakteristik runtime dari sistem pemrosesan panggilan. Penjadwal thread tidak bisa memberikan jaminan ini. Mari kita pertimbangkan situasi di mana Bob biasanya mendapatkan semua runtime yang dibutuhkan dengan menjadikannya prioritas lebih tinggi daripada Alice atau thread lain dalam aplikasi pengumpulan statistik. Karena bug atau desain yang buruk atau pengujian yang tidak tepat, Bob dapat menurunkan prioritasnya sendiri (kemampuan untuk melakukan hal ini tersedia pada hampir semua penjadwal thread), menyebabkan thread dalam aplikasi pengumpulan statistik mendapatkan kendali atas waktu proses prosesor. Demikian pula, Alice dapat menaikkan prioritasnya di atas prioritas Bob dengan efek yang sama. Satu-satunya cara untuk menjamin bahwa thread dalam ruang alamat kritis yang berbeda tidak dapat mempengaruhi satu sama lain adalah dengan menyediakan penjadwal tingkat ruang alamat, atau partisi.

Perancang perangkat lunak penting keselamatan telah mengetahui persyaratan ini sejak lama. Konsep penjadwalan partisi adalah bagian utama dari Spesifikasi ARINC 653, sebuah Antarmuka Standar Perangkat Lunak Aplikasi Avionik. Penjadwal partisi ARINC 653 mengeksekusi partisi waktu sesuai dengan garis waktu yang ditetapkan oleh perancang sistem. Setiap ruang alamat disediakan satu atau lebih jendela eksekusi dalam timeline berulang. Selama setiap jendela, semua thread di partisi lain tidak dapat dijalankan; hanya thread dalam partisi saat ini yang dapat dijalankan (dan biasanya dijadwalkan sesuai dengan aturan penjadwalan thread standar). Ketika jendela aplikasi pemrosesan panggilan aktif, sumber daya pemrosesannya dijamin; aplikasi pengumpul statistik tidak dapat berjalan dan menyita waktu pemrosesan dari aplikasi kritis. Meskipun tidak ditentukan dalam ARINC 653, tambahan yang bijaksana pada implementasinya adalah dengan menyediakan konsep partisi latar belakang. Ketika tidak ada thread yang dapat dijalankan dalam partisi aktif, penjadwal partisi harus dapat menjalankan thread latar belakang, jika ada, di partisi latar belakang alih-alih dalam keadaan diam. Contoh thread latar belakang mungkin merupakan agen diagnostik berprioritas rendah yang berjalan sesekali namun tidak memiliki persyaratan real-time yang

sulit. Upaya telah dilakukan untuk menambahkan penjadwalan partisi di atas sistem operasi komersial yang tersedia dengan secara selektif menghentikan semua thread di partisi aktif dan kemudian menjalankan semua thread di partisi berikutnya. Jadi, waktu peralihan partisi linier dengan jumlah thread di partisi: implementasi yang sangat buruk. Kernel harus memastikan waktu yang konstan, peralihan partisi latensi minimal.

2.5 KONTROL AKSES DAN KEMAMPUAN

Semua sistem operasi menyediakan beberapa bentuk kontrol akses untuk proses. Sistem tertanam mengandung banyak sumber daya: komunikasi, file, proses, berbagai perangkat, memori, dan banyak lagi. Tujuan penting dari desain keamanan adalah untuk memastikan bahwa aplikasi dapat mengakses sumber daya yang diperlukan dan tidak diperbolehkan mengakses sumber daya yang tidak diperlukan.

Banyak masalah keamanan yang disebabkan oleh arsitektur dan/atau implementasi kontrol akses yang buruk dalam sistem operasi atau penggunaan fasilitas kontrol akses yang tidak tepat oleh perancang sistem tertanam.

Pada tingkat kasar, kebijakan kontrol akses dapat dibagi menjadi dua kelas: kontrol akses diskresioner (DAC) dan kontrol akses wajib (MAC). Keduanya memiliki tempat masing-masing dalam desain sistem tertanam yang aman.

Contoh DAC adalah file UNIX: suatu proses atau thread dapat, atas kebijakannya sendiri, mengubah izin pada file yang dimilikinya, sehingga mengizinkan akses ke file tersebut oleh proses lain dalam sistem. DAC berguna untuk beberapa jenis objek di beberapa jenis sistem. Namun sistem operasi harus melangkah lebih jauh demi keamanan dan menyediakan MAC objek sistem yang penting. MAC dikelola oleh sistem dan tidak dapat dimodifikasi oleh pengguna atau proses. Sebagai contoh, mari kita pertimbangkan perangkat komunikasi yang dikelola oleh aplikasi pemrosesan panggilan. Perancang sistem harus dapat mengkonfigurasi sistem sedemikian rupa sehingga aplikasi pemrosesan panggilan, dan hanya aplikasi pemrosesan panggilan, yang memiliki akses ke perangkat ini. Aplikasi lain dalam sistem tidak dapat meminta dan mendapatkan akses secara dinamis ke perangkat ini. Dan program pemrosesan panggilan tidak dapat secara dinamis memberikan akses ke perangkat ke aplikasi lain di sistem. Kontrol akses diberlakukan oleh kernel, tidak dapat dilewati oleh kode aplikasi, dan karenanya bersifat wajib. Kontrol akses wajib memberikan jaminan. Kontrol akses diskresi hanya efektif jika aplikasi yang menggunakannya.

Kombinasi kontrol akses wajib dan diskresi seringkali diperlukan untuk mengimplementasikan sistem tertanam yang canggih. Misalnya, kontrol akses wajib dapat digunakan untuk membagi sumber daya sistem di beberapa domain keamanan sehingga tidak ada aplikasi dalam domain keamanan yang dapat mengakses sumber daya tidak sah. Namun, dalam domain keamanan, kontrol akses diskresi dapat digunakan untuk memberikan pembagian sumber daya yang lebih fleksibel.

DAC dan MAC bisa hadir dalam berbagai bentuk; keputusan kontrol akses mungkin didasarkan pada kombinasi atribut, peran, dan status subjek (pengguna/proses) dan objek (perangkat) yang hampir tak terbatas.

Studi Kasus: Browser Web Aman

Kontrol akses menerapkan prinsip otoritas terkecil atas proses dan sumber daya dalam sistem tertanam. Ketika kerentanan keamanan tingkat tinggi yang disebabkan oleh kelemahan perangkat lunak pada sistem operasi umum seperti Windows dipublikasikan, kita sering kali membuat asumsi yang salah bahwa kerentanan perangkat lunak tersebut patut disalahkan. Sebenarnya, kurangnya pembatasan otoritas akses pada sistemlah yang sering kali menjadi penyebabnya. Cacat perangkat lunak, jika diatasi dengan benar, mungkin tidak memiliki dampak keamanan apa pun. Sayangnya, kerentanan dalam suatu aplikasi sering kali dieksploitasi oleh penyerang untuk mendapatkan akses ke sumber daya yang seharusnya tidak dimiliki oleh aplikasi tersebut.

Mari kita lihat contoh browser web pada umumnya, seperti Firefox atau Chrome. Browser adalah aplikasi interaktif pengguna dan oleh karena itu memerlukan akses ke tampilan grafis serta input keyboard dan mouse (keduanya dapat disediakan melalui akses ke pengelola jendela sistem). Browser memerlukan akses ke sistem file untuk menyimpan file yang diunduh dari jaringan atau untuk memuat file untuk penelusuran. Browser memerlukan akses yang dapat dieksekusi ke kode dan data aplikasi pribadinya. Browser harus dapat meluncurkan dan/atau berkomunikasi dengan berbagai macam plug-in, seperti penyaji HTML5 dan JPEG. Plug-in browser biasanya diimplementasikan menggunakan perpustakaan tautan dinamis (DLL). DLL memungkinkan aplikasi mendapatkan akses dinamis dan terprogram ke fungsionalitas eksternal dengan memuat fungsionalitas tersebut langsung ke ruang alamat aplikasi pemanggil. DLL mudah dikonfigurasi, diluncurkan, dan didistribusikan.

Sayangnya, DLL, setidaknya cara penerapannya di sebagian besar sistem operasi, merupakan contoh klasik dari manajemen hak kontrol akses yang buruk. DLL dijalankan dalam lingkungan aplikasi pemanggil dan oleh karena itu biasanya memiliki izin kontrol akses yang sama seperti yang dimiliki pengguna. Dengan demikian, plugin yang tidak dipercaya oleh pengguna dapat meluncurkan program lain, mengakses file pengguna mana pun, mengakses memori browser dan sumber daya lainnya, mengakses jaringan, dan sebagainya. Selain itu, banyak plugin browser yang berukuran besar, program perangkat lunak canggih, dan penuh dengan bug.

Kerentanan buffer overflow di plug-in memberikan potensi eksekusi kode penyerang di lingkungan pengguna. Serangan seperti ini sering terjadi di alam liar. Aplikasi populer yang telah dieksploitasi antara lain Adobe Flash, Adobe Acrobat, dan penyaji JPEG. Vektor serangan yang umum adalah membuat file Flash, PDF, atau JPEG berbahaya yang, ketika dijelajahi melalui web, menyebabkan buffer overflow selama proses rendering dan sebagai hasilnya kode berbahaya dimasukkan.

Membangun browser aman yang kebal terhadap masalah seperti itu memerlukan lebih dari sekedar penggunaan kontrol akses, namun mari kita mulai dari sana (di Bab 3 kita akan membahas lebih lanjut tentang cara membuat perangkat lunak yang aman). Mengapa penyaji JPEG harus memiliki akses ke sistem file, jaringan, atau sumber daya aplikasi lain apa pun selain yang diperlukan untuk memecahkan kode JPEG?

Mari kita asumsikan untuk saat ini bahwa penyaji JPEG dapat diimplementasikan sebagai sebuah proses yang aksesnya ke sumber daya komputer dapat diatur secara independen dari pengguna atau aplikasi browser pengguna. Masukan ke proses JPEG adalah aliran bit yang mewakili file masukan terkompresi JPEG. Output dari proses JPEG adalah aliran bit yang mewakili gambar yang didekompresi. Satu-satunya sumber daya yang diperlukan oleh proses JPEG adalah sedikit RAM, beberapa waktu pemrosesan, dan mekanisme komunikasi yang digunakan browser untuk menyediakan masukan terkompresi dan menerima keluaran terdekompresi.

Proses JPEG harus melarang akses ke file lain, jaringan, atau bahkan kemampuan untuk meluncurkan program lain. Sekarang mari kita asumsikan bahwa proses JPEG penuh dengan bug, termasuk jenis kelemahan buffer overflow yang merupakan kutukan bagi pengguna web di seluruh dunia. Cacat pada aplikasi JPEG dapat menyebabkan gambar JPEG perusak yang berbahaya meluap ke tumpukan waktu proses proses JPEG, sehingga memasukkan kode berbahaya. Kode ini memungkinkan penyerang untuk merusak, menonaktifkan, atau bahkan sepenuhnya menundukkan proses JPEG sesuai keinginannya. Namun, penyerang tidak dapat mengakses file pengguna; meluncurkan program lain yang dapat menyebabkan kerusakan; atau mengakses sumber daya lain apa pun di luar cakupan proses JPEG itu sendiri yang terbatas dan terbatas. Rekayasa kontrol akses secara dramatis membatasi kerusakan akibat buffer overflow.

Alasan terbesar mengapa buffer overflows begitu merusak adalah karena sistem operasi yang tidak aman dan teknik pengembangan perangkat lunak mendorong penggunaan otoritas sekitar.

Seperti terlihat pada contoh browser web yang tidak aman, otoritas plugin bersifat ambien karena diwarisi dari lingkungan sekitar pengguna. Demi kenyamanan, pengembang merancang browser sehingga plugin dapat dengan mudah menjalankan fungsi dan mengakses sumber daya atas nama pengguna. Otoritas yang berlebihan inilah yang mengakibatkan kerusakan luas akibat kelemahan pemrograman yang seharusnya tidak berbahaya.

2.5.1 Perincian versus Kesederhanaan Kontrol Akses

Salah satu tantangan terbesar yang dihadapi perancang sistem tertanam sehubungan dengan kebijakan keamanan kontrol akses adalah menemukan keseimbangan yang tepat antara rincian kebijakan dan kemampuan pemeliharaan serta jaminan kebijakan.

Kebijakan yang terlalu sederhana mungkin memberikan hak istimewa yang berlebihan pada proses di seluruh sistem. Ironisnya, kebijakan yang terlalu rumit dapat menimbulkan kerentanan keamanan. Spesifikasi kebijakan adalah bagian dari TCB, dan seperti kode sumber dalam TCB, spesifikasi kebijakan harus dibuat dengan hati-hati.

Kerangka kebijakan kontrol akses Linux modern disebut SELinux (Linux yang Ditingkatkan Keamanan). Sebuah makalah penelitian USENIX melaporkan bahwa sampel kebijakan Linux 2.4.19 SELinux terdiri dari lebih dari 50.000 pernyataan kebijakan dan 100.000 penetapan izin pada 700 jenis subjek.[7] Spesifikasi besar semacam ini tidak praktis untuk dijamin dan dipelihara. Para peneliti menyatakan, “Kami percaya bahwa ukuran dan kompleksitas kebijakan contoh SELinux membuat tidak praktis untuk mengharapkan bahwa administrator pada umumnya dapat menyesuaikannya untuk memastikan perlindungan Basis

[7] Jaeger T, Sailer R, Zhang X. Analyzing Integrity Protection in the SELinux Example Policy. Washington, DC: Proceedings of the 12th USENIX Security Symposium; August 4e8, 2003.

Komputasi Tepercaya (TCB) mereka dan untuk memenuhi sasaran keamanan situs mereka di TCB ini." Kompleksitas kebijakan SELinux yang berlebihan tidak boleh ditafsirkan sebagai kritik terhadap kerangka kontrol akses yang mendasari kebijakan tersebut; melainkan merupakan hasil kompleksitas dari TCB Linux itu sendiri. Meskipun demikian, perancang yang tertanam harus menyadari perlunya menerapkan kebijakan yang dapat dikelola dan dapat dibuktikan.
Untuk mengeksplorasi lebih jauh topik penting ini, mari kita lihat kontrol akses untuk sistem file. Kebijakan kontrol akses paling sederhana untuk sistem file adalah dengan mengizinkan akses sistem file tanpa batas ke proses apa pun yang berhubungan dengan file. Proses apa pun yang tidak memerlukan layanan file tidak diperbolehkan mengakses sistem file. Misalkan sistem tertanam memiliki proses Alice dan Bob, dan Alice menggunakan sistem file untuk pencatatan peristiwa sistem dan Bob adalah proses waktu nyata yang tidak memerlukan file. Kebijakan kontrol aksesnya sederhana:

Proses	File System Access
Alice	Ya
Bob	Tidak

Sekarang anggaplah proses ketiga, Charlie, memerlukan akses sistem file, namun tidak pernah ke file yang sama yang digunakan oleh Alice. Jika kebijakan keamanan sistem file mencakup perincian tingkat direktori, maka perancang dapat menentukan direktori sandbox terisolasi, A dan C, di mana Alice dan Charlie dapat dengan bebas membuat, membaca, dan menulis file pribadi mereka:

Proses	File System Access	A	C
Alice	Ya	Ya	Tidak
Bob	Tidak	Tidak	Tidak
Charlie	Ya	Tidak	Ya

Sekarang anggaplah proses keempat, David, adalah monitor kesehatan yang secara berkala membaca file status yang ditulis oleh Alice. Hal ini menyiratkan bahwa kebijakan keamanan harus menyediakan tidak hanya perincian tingkat file, namun juga perincian izin tipe akses individual untuk file:

Proses	File System Access	A	C	A/Status Read	A/Status Write
Alice	Ya	Ya	Tidak	Tidak	Ya
Bob	Tidak	Tidak	Tidak	Tidak	Tidak
Charlie	Ya	Tidak	Ya	Tidak	Tidak
David	Ya	Ya	Tidak	Ya	Tidak

Sekarang anggaplah Alice dipercaya untuk menambahkan pesan status ke file Status, namun perancang sistem tidak ingin memberikan Alice hak untuk menghapus atau memotong file Status. Ini bukanlah hal yang aneh; pencatatan audit sering kali dianggap sebagai operasi sensitif di mana akses terhadap log audit dikontrol secara ketat untuk mengurangi

kemungkinan korupsi atau gangguan. Parameter kontrol akses dapat diperluas untuk mencakup jenis operasi spesifik apa pun yang dapat dilakukan pada file:

Proses	File System Access	A	C	A/Status Read	A/Status Write	A/Status Lampiran	A/Status Create	A/Status Trunc	A/Status Search	A/Status Delete
Alice	Ya	Ya	Tidak	Tidak	Ya	Ya	Tidak	Tidak	Tidak	Tidak
Bob	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
Charlie	Ya	Tidak	Ya	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
David	Ya	Ya	Tidak	Ya	Tidak	Tidak	Tidak	Tidak	Ya	Tidak

Tabel ini menyiratkan kebijakan datar yang dikodekan dengan satu bit (ya/tidak) untuk setiap hak akses. Sebaliknya, sistem kontrol akses dapat diimplementasikan secara hierarki sehingga akses ke sistem file diperlukan sebelum permintaan sistem file dapat dibuat. Kemudian, akses ke direktori tertentu diperiksa sebelum permintaan file apa pun dalam direktori tersebut dapat dibuat. Dan seterusnya. Demikian pula, alih-alih konfigurasi biner, struktur kontrol akses dapat diimplementasikan sebagai daftar, daftar tupel, dan seterusnya; ketika akses ke file diminta, sistem dapat memeriksa daftar tupel proses dan operasi yang valid.

Tabel ini hanya mencakup akses sistem file. Dalam sistem tertanam, jenis subjek dan objek yang memerlukan kebijakan kontrol akses mencakup perangkat komunikasi, saluran komunikasi ke proses lain, jenis memori tertentu, semafor, pengatur waktu, dan sebagainya. Sangat mudah untuk melihat bagaimana kebijakan kontrol akses dapat dengan cepat menjadi terlalu rumit untuk dikelola secara efektif.

Pada akhirnya, perancang sistemlah yang diberi tanggung jawab untuk menemukan media yang menyenangkan untuk kecanggihan kebijakan keamanan. Tujuannya adalah untuk menemukan kebijakan paling sederhana yang memberikan batasan hak istimewa yang diperlukan di seluruh sistem. Dalam praktiknya, sebagian besar sumber daya akan memiliki satu pemilik selama masa pakai sumber daya tersebut. Idealnya, kebijakan keamanan untuk sumber daya tersebut harus dikodekan dengan satu entri, baik itu bit dalam tabel, elemen XML, atau representasi apa pun yang digunakan sistem operasi dan/atau manajer sumber daya.

2.5.2 Daftar Putih versus Daftar Hitam

Kesederhanaan kebijakan pengendalian akses sangat penting untuk mendapatkan jaminan bahwa kebijakan tersebut lengkap dan benar. Diberikan suatu objek, O, dan N proses dalam sistem dP1, P2, P3, ., PN, dimana M < N memerlukan akses ke Od, ada dua cara untuk menentukan daftar kontrol akses dari M proses yang diberi wewenang untuk mengakses O. Yang pertama metode, daftar hitam, dikaitkan dengan O daftar proses NeM yang tidak diizinkan untuk mengakses O. Sebaliknya, daftar putih dikaitkan dengan O daftar proses M yang diizinkan untuk mengakses O. Ketika akses ke O diminta oleh proses P, agen penegakan keamanan (bisa berupa kernel atau proses yang dipercaya untuk mengelola akses ke O) memeriksa daftar kontrol akses untuk melihat apakah P diotorisasi untuk O. Alternatifnya, daftar kontrol akses dapat dikaitkan dengan subjek, bukan objek. Dengan kata lain, daftar putih untuk proses P mencakup objek-objek yang diotorisasi oleh P.

Untuk sebagian besar objek dan sumber daya, daftar putih lebih disukai daripada daftar hitam karena kecenderungannya terhadap berkurangnya implementasi dan desain hak istimewa.

Daftar hitam dapat menimbulkan kerentanan karena secara tidak sengaja mengizinkan otoritas yang berlebihan. Misalkan suatu kejadian sistem menyebabkan proses baru, Q, dibuat untuk menangani kejadian tersebut. Jika akses objek O dikendalikan oleh daftar hitam, maka Q secara otomatis diberikan akses ke O baik Q memerlukannya atau tidak. Sebaliknya, jika akses ke O dikontrol oleh daftar putih, maka Q tidak secara otomatis diotorisasi untuk O. Perancang sistem harus mengambil tindakan eksplisit untuk menambahkan Q ke daftar putih untuk O jika Q memerlukannya.

Paket anti-malware pada dasarnya adalah suatu bentuk daftar hitam. Saat suatu program dimasukkan ke dalam sistem, proses anti-malware membandingkan program tersebut dengan daftar program jahat yang diketahui atau mencari pola kode malware yang diketahui. Masalahnya, tentu saja, malware yang belum ditemukan (termasuk yang baru dibuat) lolos uji. Semakin banyak desainer yang memikirkan keamanan yang mencari daftar putih untuk mengatasi masalah ini. Sistem tertanam dapat dikirimkan dengan daftar program bagus yang diketahui dapat dijalankan. Program apa pun yang gagal mencocokkan sesuatu dalam daftar putih akan ditolak. Perbandingan tersebut mungkin mencakup pemeriksaan hash untuk memverifikasi integritas program yang masuk daftar putih.

Alternatifnya, sistem tertanam dapat menerapkan kebijakan yang hanya memuat program tepercaya, di mana kepercayaan dibangun melalui tanda tangan digital. Pemilik sistem harus secara kriptografis menandatangani program yang dapat dimuat secara apriori, dan pemuat yang tertanam memverifikasi tanda tangan tersebut sebelum menerima adopsi program ke dalam sistem. Tanda tangan tersebut mewakili pernyataan pemilik bahwa kumpulan bit khusus ini aman untuk diterapkan. Tentu saja, mekanisme ini menyiratkan bahwa pemilik sistem (atau otoritas yang didelegasikan untuk mengambil keputusan tersebut) tidak menandatangani program palsu atau jahat. Teknologi tanda tangan digital untuk otentikasi dibahas pada Bab 4.

Daftar hitam sebaiknya digunakan hanya pada situasi dimana tidak ada alternatif lain yang praktis. Contoh penting adalah penggunaan daftar pencabutan untuk titik akhir komunikasi pada jaringan besar yang berkembang pesat (misalnya Internet). Ketika klien ingin memverifikasi validitas server, tidak praktis untuk memeriksa daftar putih server yang valid. Ketika sertifikat server dianggap tidak valid sebelum tanggal kedaluwarsa alaminya (kejadian yang relatif jarang terjadi), sertifikat tersebut ditempatkan pada daftar pencabutan yang diperiksa selama pembentukan sesi komunikasi.

2.5.3 Deputi Masalah yang Bingung

Sebelumnya pada bab ini, kita telah mempelajari konsep browser web yang dibuat lebih aman menggunakan rekayasa kontrol akses.

Tujuannya adalah untuk membatasi otoritas plug-in browser relatif terhadap otoritas ambient default pengguna yang secara otomatis didelegasikan ke plug-in saat diluncurkan. Hak istimewa yang berlebihan melalui delegasi implisit adalah penyebab masalah kontrol

akses umum lainnya, yang disebut deputi yang bingung, seperti yang dikemukakan oleh Norman Hardy.[8]

Deputi yang bingung adalah aplikasi yang melayani beberapa aplikasi klien. Server memiliki banyak otoritas, yang masing-masing digunakan untuk mengakses sumber daya untuk tujuan berbeda. Misalnya, server memiliki wewenang untuk mengakses log audit (dengan nama path yang telah ditentukan dalam sistem file) yang mencatat tindakan server. Selain itu, server mempunyai wewenang untuk menulis file keluaran yang namanya ditentukan oleh klien dalam permintaannya ke server. Server menjadi deputi yang bingung ketika klien menentukan file output yang nama jalurnya cocok dengan log audit server. Karena memiliki wewenang untuk menulis log auditnya sendiri, server gagal mendeteksi bahwa otoritas tersebut disalahgunakan, dan log audit rusak. Server tidak pernah dirancang untuk menangani kemungkinan upaya yang salah (atau berbahaya) oleh klien untuk menentukan file output yang cocok dengan nama log audit.

Deputi yang bingung adalah contoh peningkatan hak istimewa: meskipun klien seharusnya hanya memiliki hak untuk memanggil server untuk memproses file output klien, deputi yang bingung memungkinkan klien mendapatkan otoritas terlarang untuk menimpa log audit server.

Meskipun ada beberapa pendekatan yang mungkin untuk menyelesaikan kasus uji khusus ini, solusi umum yang efektif adalah dengan mengikat suatu objek dengan jenis hak istimewa tertentu dan memaksa subjek (klien dan server) untuk hanya menangani tupel hak istimewa objek ini. Dalam contoh sebelumnya, server akan dikonfigurasi dengan otoritas "log audit" yang menyediakan akses ke log audit, dan klien akan dikonfigurasi dengan otoritas khusus pengguna yang menyediakan akses ke file output pengguna. Klien meneruskan referensi otoritas ini ke server setelah pemanggilan layanan. Server menggunakan otoritas ini untuk menulis file keluaran pengguna. Karena sistem operasi memaksa semua referensi objek dibuat melalui tupel hak istimewa ini, tidak ada mekanisme yang ditentukan bagi klien untuk merujuk ke otoritas log audit. Klien yang dibatasi dengan baik tidak akan pernah mendapatkan hak istimewa ini. Keunggulan dari pendekatan ini adalah server tidak perlu dibebani dengan kode tambahan untuk melacak izin akses yang berbeda untuk setiap akses file yang diperlukan.

2.5.4 Kemampuan versus Daftar Kontrol Akses

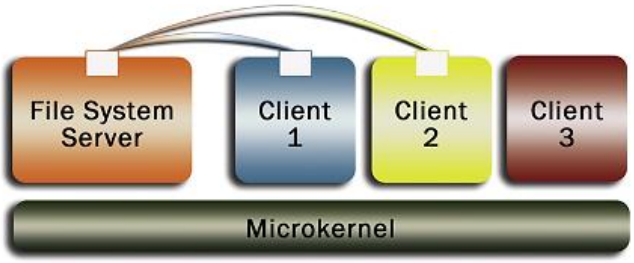
Konsep referensi objek yang menyediakan akses implisit ke objek disebut kemampuan objek. Dengan daftar kontrol akses tradisional, suatu objek direferensikan dengan deskriptor (misalnya, deskriptor file UNIX), dan metadata terpisah harus dipertahankan untuk melacak hak akses objek tersebut.

Kemampuan bertindak baik sebagai mekanisme akses maupun hak akses: tidak ada cara bagi subjek untuk mengakses suatu objek jika subjek tidak memiliki kemampuan objek tersebut.

Kemampuannya sederhana dan efisien serta mendukung desain aman dengan hak istimewa paling rendah. Dalam sistem operasi berbasis mikrokernel yang menggunakan kemampuan, titik akhir IPC seringkali merupakan bentuk kemampuan yang paling penting karena IPC digunakan oleh klien untuk meminta layanan, baik dari kernel atau dari penyedia

[8] Hardy N. "The Confused Deputy," ACM SIGOPS Operating Systems Review October 1988;22(4).

layanan mode pengguna seperti sistem file dan tumpukan jaringan. Jika suatu proses memiliki koneksi IPC ke sistem file, maka koneksi ini memberikan hak istimewa untuk meminta layanan dari sistem file. Jika suatu proses tidak memiliki koneksi IPC ke sistem file, maka proses tersebut tidak dapat meminta layanan sistem file apa pun (lihat Gambar 2.11).



Gambar 2.11: Klien 1 dan 2 masing-masing memiliki kemampuan untuk layanan sistem file; klien 3 tidak dapat meminta layanan sistem file apa pun.

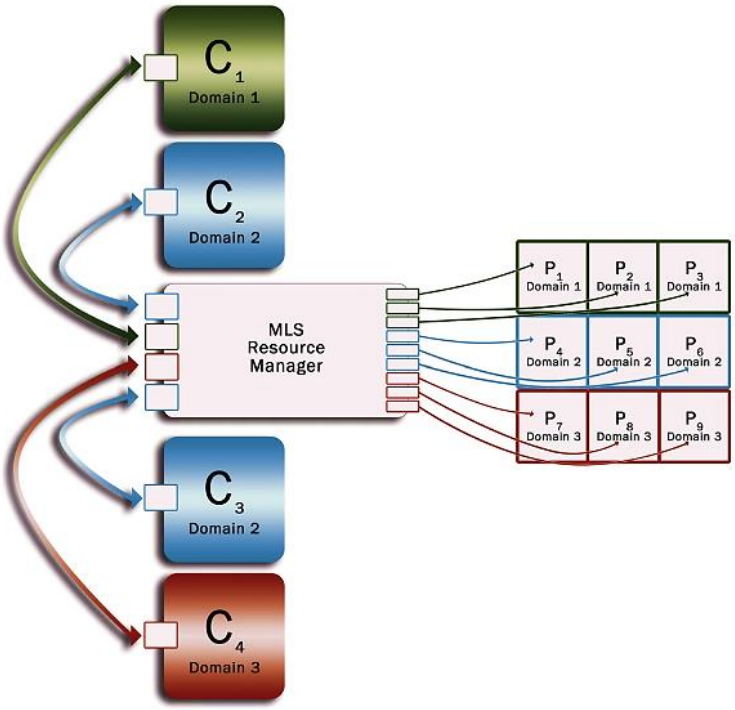
Kemampuan IPC juga sepenuhnya terukur dalam hal granularitas akses yang dapat dikontrolnya. Dalam contoh sebelumnya, granularitas kapabilitas sistem file bersifat kasar: suatu proses memiliki akses penuh atau tidak. Alternatifnya adalah dengan menggunakan kemampuan terpisah untuk setiap volume sistem file. Klien dengan kemampuan volume diberikan akses hanya ke volume tertentu serta direktori dan file penyusunnya, dan server file dapat menggunakan kemampuan ini untuk mempartisi sumber daya media di seluruh klien. Untuk kontrol yang lebih detail, sistem file dapat menggunakan kemampuan untuk setiap direktori dan file individual atau bahkan untuk izin akses individual (baca, tulis, eksekusi, dll.) untuk setiap file. Meskipun peningkatan granularitas menyebabkan berkurangnya hak istimewa, pilihan skema manajemen kapabilitas sering kali mempunyai dampak besar pada kompleksitas sistem. Dan seperti yang telah kita bahas, kompleksitas yang berlebihan dapat berdampak buruk pada keamanan sistem.

Dalam kebanyakan kasus, penyedia sistem operasi akan memberikan skema kemampuan default untuk server middleware apa pun, seperti sistem file, yang disediakan bersama dengan sistem operasi. Untuk server aplikasi yang dibuat pengguna, kemampuan IPC dapat digunakan sesuai keinginan perancang. Penyedia sistem operasi sering kali menawarkan layanan konsultasi untuk membantu memastikan model kemampuan terbaik dengan hak istimewa paling sedikit dibandingkan dengan kompleksitas.

2.5.5 Studi Kasus: Manajer Sumber Daya MLS

Bukan hal yang aneh jika sistem operasi berbasis kemampuan mendukung daftar kontrol akses selain kemampuan atau dikombinasikan dengan kemampuan untuk beberapa layanan. Misalnya, mari kita pertimbangkan aplikasi manajemen sumber daya multi-level secure (MLS) yang mengelola sekumpulan halaman memori pada berbagai tingkat keamanan. Proses klien diperbolehkan akses baca ke halaman jika dan hanya jika tingkat keamanan klien mendominasi (lebih besar atau sama dengan) tingkat keamanan halaman. Klien diperbolehkan akses tulis ke suatu halaman jika dan hanya jika tingkat keamanan klien didominasi oleh (kurang dari atau sama dengan) tingkat keamanan halaman.

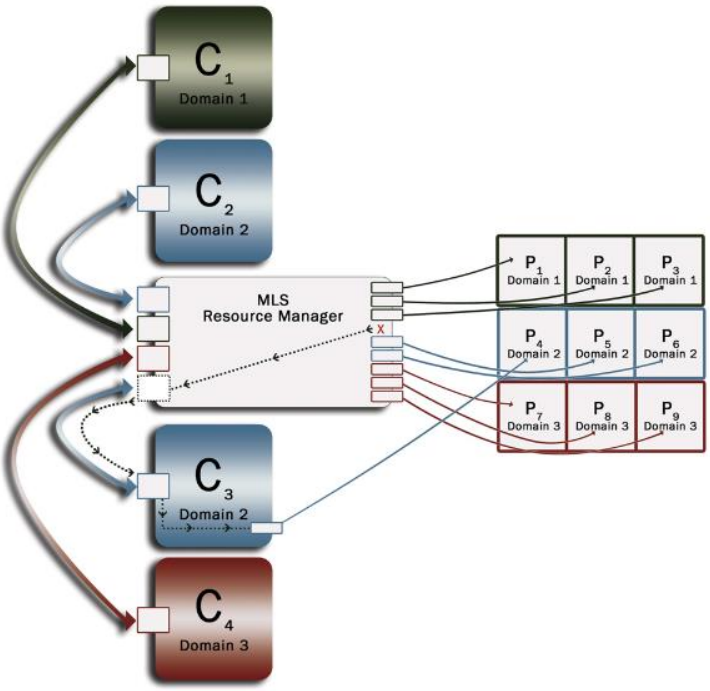
Kami juga membuat asumsi bahwa sistem dikonfigurasi sedemikian rupa sehingga klien hanya dapat berkomunikasi dengan klien lain pada tingkat keamanan yang sama atau dengan manajer sumber daya MLS; oleh karena itu klien tidak dapat mentransfer kemampuan ke klien pada tingkat keamanan yang lebih tinggi atau lebih rendah. Untuk memulai, setiap klien yang diizinkan untuk membuat permintaan dari manajer sumber daya MLS diberikan kemampuan komunikasi (lihat Gambar 2.12).



Gambar 2.12 Klien dengan kemampuan manajer sumber daya MLS untuk meminta kemampuan halaman memori.

Ketika klien meminta akses baca ke suatu halaman (misalnya, berdasarkan alamat atau nama, bergantung pada bagaimana sistem diterapkan), manajer sumber daya berkonsultasi dengan daftar kontrol akses internal halaman yang dikelola. Dalam contoh ini, daftar kontrol akses terdiri dari tupel kemampuan halaman dan label tingkat keamanan terkait. Kebijakan lain, seperti kontrol akses berbasis peran, memerlukan atribut tambahan dalam tabel kontrol akses.

Jika tingkat keamanan klien mendominasi tingkat keamanan halaman, maka manajer sumber daya menyetujui permintaan tersebut dan menggunakan kemampuan komunikasinya kepada klien untuk mentransfer kemampuan baru yang menyediakan akses baca ke halaman memori yang diminta (lihat Gambar 2.13).



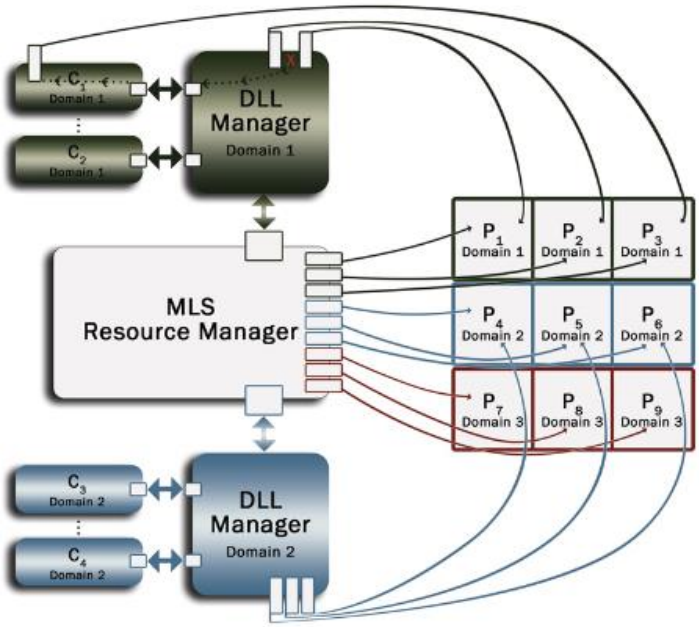
Gambar 2.13: Manajer sumber daya mentransfer kemampuan halaman memori ke klien.

Klien sekarang dapat menggunakan kemampuan baru yang diterima untuk melakukan pembacaan langsung berbagai konten halaman sebanyak yang diperlukan. Kemampuan ini memberikan efisiensi yang unggul dibandingkan dengan penerapan kontrol akses murni di mana setiap akses individu ke konten halaman perlu divalidasi terhadap daftar kontrol akses oleh manajer sumber daya.

Contoh ini menunjukkan bahwa secara praktis semua kebijakan kontrol akses dapat diterapkan berdasarkan kemampuan. Lebih jauh lagi, contoh ini mengilustrasikan bagaimana kemampuan dapat digunakan untuk menyediakan kontrol akses wajib dan diskresi. Model kapabilitas, pada dasarnya, merupakan sistem kontrol akses wajib, karena semua kepemilikan objek dan hubungan akses diberlakukan oleh mikrokernel. Perancang sistem dapat menciptakan hubungan kepemilikan yang sepenuhnya statis antara proses dan objeknya sendiri. Dalam desain seperti itu, tidak ada kontrol akses diskresi. Namun, jika diperlukan lebih banyak fleksibilitas, server tepercaya dapat dirancang dan dilengkapi dengan kemampuan yang dapat mengelola kebijakan kontrol akses kliennya sendiri (bersama dengan kemampuan untuk berkomunikasi dengan klien tersebut). Dengan demikian, model kemampuan dapat mendukung sistem kontrol akses diskresi sewenang-wenang di atas, atau sebagai tambahan, kontrol wajib yang ditentukan sistem.

Sekarang mari kita ambil studi kasus ini selangkah lebih jauh dan pertimbangkan skenario di mana pengelola sumber daya halaman memori ditugaskan untuk mendistribusikan halamannya ke server lain yang akan mengelola subset halaman pengelola sumber daya utama. Misalnya, sistem mungkin menerapkan layanan manajemen perpustakaan tautan dinamis. Ketika klien ingin memuat DLL, manajer sumber daya utama menyediakan akses baca/tulis manajer DLL ke sekumpulan halaman yang sesuai dengan konten DLL. Manajer DLL kemudian akan memberikan kemampuan read-only halaman ini kepada klien.

Tentu saja, manajer DLL juga dapat menerapkan kebijakan keamanan independen untuk akses ke DLL tertentu. Manajer DLL terpisah bahkan dapat dipakai untuk mengelola DLL individual atau kumpulan DLL (lihat Gambar 2.14). Misalnya, sistem dapat menggunakan contoh manajer DLL yang berbeda untuk setiap domain keamanan. Setiap pengelola DLL dipercaya untuk mendistribusikan halamannya dengan benar kepada klien, namun hak keseluruhan pengelola DLL dibatasi hanya pada kumpulan halaman yang disediakan oleh pengelola sumber daya utama. Dengan cara ini, kemampuan dapat mengalir di berbagai perantara, memberikan fleksibilitas yang sangat baik dalam desain sistem, sekaligus memberikan minimalisasi otoritas untuk meningkatkan keamanan. Kombinasi distribusi kemampuan yang fleksibel dan terbatas hak istimewa ini sering terjadi dalam sistem yang canggih. Contoh lainnya adalah akses ke buffer jaringan bersama saat buffer tersebut mengalir melintasi berbagai lapisan dalam tumpukan komunikasi: dari driver antarmuka jaringan, ke lapisan TCP/IP, hingga klien TCP/IP seperti proses individual yang membuat panggilan API soket, klien/server HTTP , dan klien/server FTP.



Gambar 2.14: Contoh domain kemampuan terpisah: setiap manajer DLL mengelola kemampuan halaman memori untuk klien dalam domain keamanan atas nama manajer halaman memori aman multi-level tingkat yang lebih tinggi.

2.5.6 Pembatasan dan Pencabutan Kemampuan

Salah satu keuntungan potensial yang dimiliki daftar kontrol akses dibandingkan kemampuannya adalah kolokasi kebijakan dengan penunjukan otoritas yang terkandung dalam kebijakan tersebut. Permintaan kontrol akses masuk ke server, dan server juga berisi daftar kontrol akses untuk membuat keputusan kontrol akses. Daftar kontrol akses adalah gudang tunggal informasi kebijakan yang berkaitan dengan layanan tersebut. Jika hak akses klien terhadap sumber daya dalam server harus dicabut, maka perancang sistem mengetahui bahwa menghapus klien dari daftar akses server sudah cukup.

Sebaliknya, kemampuan tidak terletak pada satu tempat saja. Memang benar, kemampuan untuk mendistribusikan kemampuan merupakan keunggulan fleksibilitas yang penting dibandingkan daftar kontrol akses. Namun bagaimana jika server tidak menginginkan kemampuan untuk melampaui klien yang diberi akses? Bagaimana server atau perancang sistem dapat yakin bahwa kemampuan ini tidak pernah dialihkan ke klien lain? Dan jika otoritas klien harus dicabut, bagaimana hal ini dapat dicapai ketika kemampuan telah didistribusikan melampaui server atau bahkan melampaui klien asli?

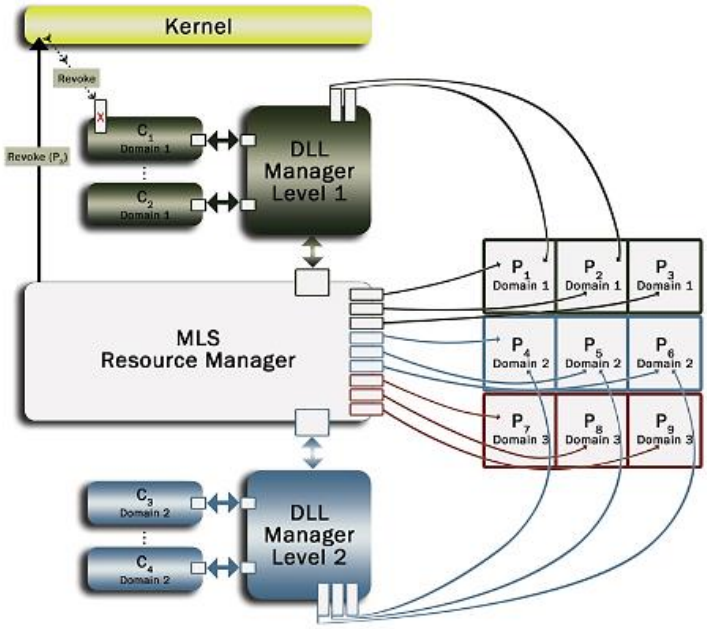
Dalam contoh wakil kami yang membingungkan, server dipercayakan untuk mengelola kemampuan file keluaran pengguna dengan benar. Jika, pada titik tertentu, administrator keamanan sistem memutuskan bahwa server tidak lagi dapat dipercaya, maka pengguna kurang beruntung; kemampuan yang didelegasikan ada di sana untuk disalahgunakan oleh server. Selain itu, karena server dapat berkomunikasi dengan klien lain, server dapat dengan jahat mentransfer kemampuan file keluaran klien ke klien lain.

Sistem kemampuan terdistribusi menunjukkan kebutuhan sistem operasi untuk menyediakan sarana dimana kemampuan dapat dibatasi dalam domain hak istimewa serta sarana untuk pencabutan.

Dalam sistem UNIX pada umumnya, proses dapat dengan mudah berkomunikasi dengan proses lain menggunakan pipa, file, soket, dan mekanisme lainnya. Deskriptor file dapat dibagikan semudah data mentah. Karena kebijakan defaultnya adalah mengizinkan komunikasi, sistem kemampuan yang ditambahkan ke UNIX tidak akan memiliki properti pengekangan yang diinginkan. Sebaliknya, pengekangan secara implisit disediakan oleh sistem mikrokernel yang melindungi proses dan kepemilikan kemampuan dalam setiap proses. Proses yang baru dibuat tidak memiliki kemampuan apa pun. Hal ini memaksa perancang sistem untuk mempertimbangkan domain kurungan dengan cermat. Jika Alice tidak memiliki kemampuan komunikasi dengan proses lainnya, maka kemampuan Alice terbatas pada Alice. Jika Alice dan Bob dapat mengkomunikasikan kemampuan satu sama lain, namun tidak ada yang memiliki koneksi ke proses lain, maka Alice dan Bob membentuk domain kurungan untuk penyatuan kemampuan mereka. Selain itu, mikrokernel dapat menyediakan mekanisme dimana proses dapat berkomunikasi (misalnya penyampaian pesan, sinkronisasi primitif) tetapi tidak dapat mentransfer kemampuan.

Untuk mengilustrasikan perlunya pencabutan, mari kita pertimbangkan kembali studi kasus sebelumnya tentang pengelola halaman memori dan pengelola DLL yang patuh. Misalkan manajer sumber daya utama memutuskan bahwa ia harus mengambil kembali halaman memori yang ditetapkan ke salah satu manajer DLL. Misalkan juga manajer DLL telah mendistribusikan kemampuan baca untuk halaman-halaman ini ke sejumlah besar kliennya agar dapat menyediakan layanan perpustakaan bersama. Pencabutan sekarang menyiratkan persyaratan untuk menghapus kemampuan baca dari semua proses klien yang terpengaruh serta kemampuan baca/tulis dari manajer DLL itu sendiri. Selain itu, manajer sumber daya utama tidak mengetahui klien mana yang diberi akses baca. Dengan demikian, skema yang mencoba untuk mencatat semua distribusi kemampuan untuk pencabutan di masa depan sangatlah tidak praktis.

Sistem kapabilitas mikrokernel harus menyediakan mekanisme sederhana bagi pemilik asli kapabilitas (dalam hal ini, manajer sumber daya utama) untuk mencabut kapabilitas tersebut, terlepas dari bagaimana kapabilitas tersebut telah disebarkan ke seluruh proses klien. Salah satu solusi untuk masalah ini mengingatkan kita pada bagaimana fungsi C++ meneruskan akses baca ke objek yang ditentukan ke fungsi lain. Alih-alih meneruskan objek asli, fungsi meneruskan referensi ke objek tersebut. Demikian pula, suatu proses dapat mendistribusikan referensi ke kemampuan. Referensi ini digunakan sebagai deskriptor akses dengan cara yang sama seperti objek aslinya. Namun, ketika proses pemilikan mengeluarkan operasi pencabutan pada objek asli, semua referensi yang beredar secara otomatis dinonaktifkan (lihat Gambar 2.15). Keterkaitan antara referensi dan objek asli disimpan oleh mikrokernel.



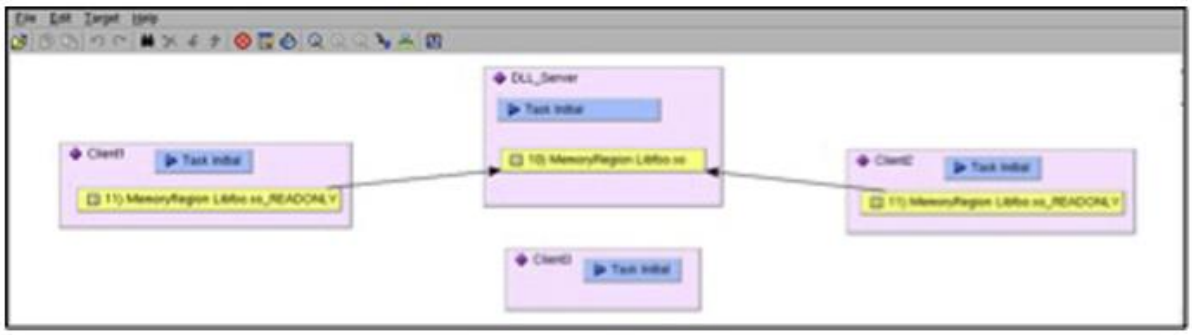
Gambar 2.15: Contoh pencabutan kemampuan: manajer sumber daya tingkat tinggi secara efisien mencabut/menonaktifkan kemampuan halaman memori yang telah didistribusikan ke server tingkat kedua dan kemudian ke kliennya.

Pengembang hanya perlu mengetahui cara menggunakan referensi yang dapat dibatalkan alih-alih mentransfer objek aslinya.

2.5.7 Desain Aman Menggunakan Kemampuan

Kemampuan memberikan cara yang sangat sederhana untuk mendorong kebiasaan desain yang aman. Karena kemampuan mewujudkan penunjukan sumber daya serta otoritas aksesnya, diagram arsitektur perangkat lunak secara bersamaan menggambarkan interaksi fungsional serta hak akses antara subjek dan objek. Sebaliknya, desain arsitektur sistem yang hanya berbasis daftar kontrol akses yang menunjukkan konektivitas antara subjek dan objek akan gagal memberikan gambaran lengkap tentang kebijakan keamanan sehubungan dengan hak akses objek. Daftar kontrol akses setiap manajer sumber daya dalam sistem perlu ditambahkan ke gambar.

Sebagai ilustrasi, mari kita lihat kembali contoh manajer DLL sebelumnya yang memiliki kemampuan pada wilayah memori yang dapat ditulisi yang berisi kode perpustakaan dan klien manajer DLL yang diberikan akses baca-saja ke wilayah memori ini jika dan hanya jika mereka diizinkan oleh perancang sistem. Vendor sistem operasi berbasis kemampuan sering kali menyediakan alat grafis untuk mendefinisikan interaksi antara proses dan sumber daya yang memerlukan akses.



Gambar 2.16: Desain aman dengan kemampuan.

Gambar 2.16 menunjukkan salah satu alat tersebut, Integrate, yang digunakan dengan sistem operasi INTEGRITY dari Green Hills Software. Dalam tangkapan layar ini, server DLL memiliki kemampuan baca/tulis ke wilayah memori yang sesuai dengan perpustakaan, Libfoo.so. Kotak “MemoryRegion Libfoo.so” di dalam kotak proses DLL_Server mencerminkan kepemilikan ini. Dua proses lain dalam sistem, Client1 dan Client2, didefinisikan memiliki kemampuan read-only yang mereferensikan wilayah memori. Panah yang menghubungkan MemoryRegions Klien resmi kembali ke objek asli mencerminkan hubungan ini. Perhatikan bahwa proses Client3 tidak memiliki kemampuan seperti itu dan karenanya tidak memiliki otoritas atau mekanisme untuk mengakses perpustakaan bersama. Dengan demikian, diagram arsitektur sederhana menyampaikan fungsionalitas dan wewenang klien untuk mengakses perpustakaan bersama yang disediakan oleh server.

Alat Integrasi menyimpan informasi ini dalam file berformat ASCII (lihat Gambar 2.17) sehingga perancang sistem yang lebih suka dapat memeriksa atau melakukan pasca-proses hubungan keamanan dalam editor atau utilitas favorit mereka. File ini dibaca langsung oleh sistem build INTEGRITY dan dimasukkan ke dalam biner gambar akhir itu sendiri.

```
AddressSpace                                DLL_Server
      Object          10
                MemoryRegion
                Name                        Libfoo.so
      EndObject
EndAddressSpace

AddressSpace                                Client1
      Object          11
                MemoryRegion
                MapTo                   Libfoo.so
                Name                        Libfoo.so_READONLY
                Write                   false
      EndObject
EndAddressSpace


AddressSpace                                Client2
      Object          11
                MemoryRegion
                Name                        Libfoo.so_READONLY
                MapTo                   Libfoo.so
                Write                   false
      EndObject
EndAddressSpace

AddressSpace                                Client3
EndAddressSpace
```

Gambar 2.17 Representasi desain sistem format ASCII.

Kemampuan objek mewakili kecanggihan dalam desain kontrol akses sistem operasi. Kemampuannya kuat dan efisien. Mereka mempromosikan arsitektur perangkat lunak berbasis komponen dengan hak istimewa paling rendah. Dan kemampuannya mudah dipahami oleh pengembang dan administrator keamanan. Kemampuan dapat digunakan untuk mengelola sejumlah kecil sumber daya dengan pemilik tunggal dan eksklusif hingga skema logika kebijakan yang paling canggih.

2.6 HYPERVISOR DAN VIRTUALISASI SISTEM

Sistem operasi telah lama memainkan peran penting dalam sistem tertanam. Tujuan historis utama dari sistem operasi adalah untuk menyederhanakan kehidupan pengembang produk elektronik, membebaskan mereka untuk fokus pada diferensiasi. Sistem operasi memenuhi misi ini dengan mengabstraksi sumber daya perangkat keras (dRAM dan penyimpanan), periferal konektivitas seperti USB dan Ethernet, dan perangkat antarmuka manusia seperti layar sentuh.

Abstraksi disajikan kepada pengembang dalam bentuk API dan mekanisme yang nyaman untuk berinteraksi dengan perangkat keras dan mengelola beban kerja aplikasi. Sistem operasi dipadukan dengan lingkungan pengembangan, kompiler, debugger, editor, penganalisis kinerja, dan sebagainya, yang membantu para insinyur membangun aplikasi canggih mereka dengan cepat dan memanfaatkan sistem operasi secara maksimal.

Tentu saja, lingkungan ini telah berkembang secara dramatis. Daripada hanya menyediakan tumpukan TCP/IP, sistem operasi tertanam terkadang harus menyediakan rangkaian lengkap protokol dan layanan tingkat aplikasi seperti FTP dan server web. Daripada perpustakaan grafis sederhana, sistem operasi mungkin perlu menyediakan audio multimedia canggih dan kerangka grafis 3D. Selain itu, seiring dengan semakin mumpuninya Sistem-on-Chip (SoC) tertanam, aplikasi dan beban kerja real-time dikonsolidasikan. Contoh yang dibahas secara singkat di Bab 1 tentang sistem infotainmen otomotif yang mengintegrasikan kemampuan kamera pandangan belakang merupakan representasi dari tren ini: sistem operasi harus menyediakan lingkungan aplikasi yang kuat sekaligus merespons kejadian real-time secara instan dan melindungi antarmuka komunikasi sensitif dari kerusakan.

Karena perannya yang sentral, sistem operasi terkadang menjadi medan pertempuran antara produsen elektronik yang tidak hanya bertujuan untuk membedakan tetapi juga melindungi keunikan dan investasi dalam inovasi mereka. Pasar ponsel pintar adalah contoh nyata: vendor silikon, produsen elektronik konsumen, dan penyedia layanan di semua tingkatan ingin mengendalikan antarmuka manusia-mesin, yang bertindak sebagai portal pendapatan, loyalitas, dan pengenalan merek.

Tren menuju konsolidasi telah menimbulkan tantangan yang signifikan bagi pemasok sistem operasi tertanam yang harus menavigasi berbagai tumpukan, antarmuka, standar, dan paket perangkat lunak. Kompleksitas ini juga memicu tren menuju model open source untuk mendapatkan keuntungan dari basis pengembang yang besar dan terdistribusi. Linux adalah kisah sukses utama. Namun, meskipun Linux telah berhasil memperoleh pangsa pasar yang besar, Linux juga mengalami fragmentasi yang sangat besar. Apa yang disadari oleh pengembang sistem tertanam adalah bahwa mereka harus menyesuaikan dan memperluas Linux agar dapat memperoleh diferensiasi dan kontrol platform. Intinya, upaya berbasis Linux ini telah menjadi sistem operasi mandiri yang baru bagi para pemangku kepentingan.

Masalah utamanya adalah abstraksi sistem operasi pada file, perangkat, komunikasi jaringan, grafik, dan thread sudah mulai mencapai batas kegunaannya. Pengembang aplikasi dan pemasok elektronik yang menjadi terlalu bergantung pada satu lingkungan abstraksi sistem operasi dapat mengalami kesulitan jika sistem operasi tersebut gagal memenuhi persyaratan yang muncul, mengalami hambatan dalam perizinan atau hak IP, atau dikalahkan oleh sistem operasi lain di pasar.

Jelasnya, pengembang sistem tertanam memerlukan platform untuk inovasi dan diferensiasi yang cukup fleksibel untuk mengakomodasi evolusi cepat yang tak terhindarkan dalam teknologi perangkat keras dan perangkat lunak. Bayangkan sebuah platform yang dapat menjalankan dua sistem operasi yang sangat berbeda secara bersamaan pada mikroprosesor yang sama. Bayangkan sebuah platform yang dapat menjalankan sistem operasi multimedia konsumen tercanggih seperti Android sambil tetap menjalankan tumpukan komunikasi hard real-time dan fungsi-fungsi penting keamanan yang sepenuhnya terlindungi di luar jangkauan Androiddall pada SoC yang sama. Bayangkan sebuah platform yang dapat memenuhi persyaratan konsolidasi otomotif yang disebutkan di atas: sistem infotainment otomotif multimedia yang juga mencapai waktu boot dingin dalam milidetik untuk mendukung kamera tampak belakang yang aktif secara instan (di layar yang sama dengan sistem operasi infotainment utama) dan komunikasi melalui jaringan otomotif real-time (misalnya, Controller

Area NetworkdCAN) menggunakan SoC yang sama untuk menghemat ukuran, berat, daya, dan biaya.

Tingkat abstraksi baru yang diperlukan untuk mengatasi sistem elektronik terkonsolidasi yang semakin canggih adalah sistem operasi itu sendiri, bukan hanya sumber daya perangkat keras komputer.

Pengembang memerlukan fleksibilitas untuk menjalankan sistem operasi apa pun, bukan sembarang aplikasi, dengan mudah, pada perangkat keras yang sama. Jawaban terhadap dilema sistem operasi adalah hypervisor tertanam, yang mengimplementasikan virtualisasi sistem. Virtualisasi sistem memungkinkan hosting beberapa sistem komputer virtual pada satu sistem komputer fisik. Sistem komputer virtual ini sering disebut mesin virtual. Kemampuan untuk memvirtualisasikan seluruh sistem komputer memungkinkan beberapa sistem operasi independen untuk dijalankan secara bersamaan pada platform perangkat keras bersama. Sistem operasi ini sering disebut sebagai sistem operasi "tamu" karena mereka diizinkan untuk berbagi perangkat keras fisik sebagai tamu dari lapisan perangkat lunak yang memilikinya. Lapisan ini disebut hypervisor. Tidak seperti hypervisor perusahaan, hypervisor tertanam dirancang khusus untuk elektronik tertanam dan seluler.

Motivasi teknologi virtualisasi sistem di pusat data sudah diketahui, termasuk optimalisasi sumber daya dan peningkatan ketersediaan layanan. Namun teknologi virtualisasi memiliki penerapan yang lebih luas di seluruh dunia tertanam, termasuk perangkat seluler yang mendukung keamanan, peralatan keamanan virtual tertanam, transaksi keuangan tepercaya, konsolidasi beban kerja, dan banyak lagi. Visi ini terwujud, sebagian karena teknologi virtualisasi berbantuan perangkat keras yang kini diperluas ke perangkat tertanam dan seluler. Bagian selanjutnya dari bab ini memberikan gambaran umum tentang evolusi arsitektur hypervisor tertanam, termasuk tren perangkat lunak dan perangkat keras, dan pengaruhnya terhadap keamanan virtualisasi sistem. Kami juga mendiskusikan serangkaian aplikasi menarik untuk virtualisasi aman di seluruh komunitas yang berkepentingan.

2.6.1 Pengenalan Virtualisasi Sistem

Virtualisasi sistem komputer pertama kali diperkenalkan di mainframe pada tahun 1960an dan 70an. Meskipun virtualisasi masih merupakan fasilitas yang belum dimanfaatkan selama tahun 80an dan 90an, para ilmuwan komputer telah lama memahami banyak penerapan virtualisasi sistem, termasuk kemampuan untuk menjalankan sistem operasi yang berbeda dan lama pada satu platform perangkat keras.

Pada awal milenium, VMware membuktikan kepraktisan virtualisasi sistem penuh, menghosting sistem operasi "tamu" yang tidak dimodifikasi dan bertujuan umum seperti Windows, pada platform perangkat keras umum berbasis Intel Architecture (IA).

Pada tahun 2005, Intel meluncurkan Teknologi Virtualisasi (Intel VT), yang menyederhanakan dan mempercepat virtualisasi. Akibatnya, sejumlah produk perangkat lunak virtualisasi telah muncul, atau disebut monitor mesin virtual atau hypervisor, dengan karakteristik dan tujuan yang berbeda-beda. Bantuan perangkat keras serupa untuk virtualisasi sistem telah muncul di arsitektur CPU tertanam populer lainnya, termasuk ARM dan Power.

Meskipun virtualisasi mungkin paling dikenal karena penerapannya dalam konsolidasi dan penyediaan server pusat data, teknologi ini telah berkembang pesat di sistem kelas desktop dan laptop, dan baru-baru ini diterapkan pada lingkungan seluler dan tertanam.

Ketersediaan teknologi virtualisasi sistem di berbagai platform komputasi memberi pengembang dan ahli teknologi platform terbuka terbaik: kemampuan untuk menjalankan sistem operasi apa pun dalam kombinasi apa pun, menciptakan fleksibilitas yang belum pernah terjadi sebelumnya dalam penerapan dan penggunaan.

Namun sistem tertanam seringkali memiliki kendala sumber daya dan keamanan yang sangat berbeda dibandingkan dengan komputasi server. Kami juga fokus pada dampak arsitektur hypervisor terhadap kendala ini.

2.6.2 Penerapan Virtualisasi Sistem

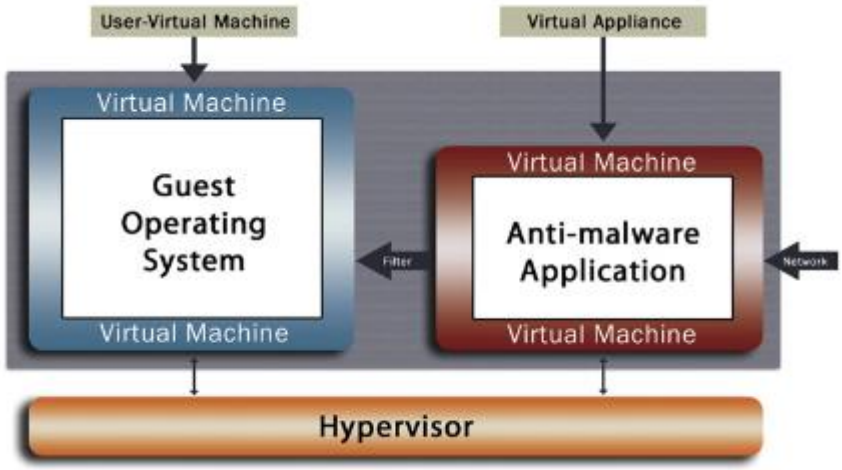
Virtualisasi mainframe didorong oleh beberapa aplikasi yang sama yang ditemukan dalam sistem perusahaan saat ini. Awalnya, virtualisasi digunakan untuk berbagi waktu, serupa dengan peningkatan pemanfaatan perangkat keras yang mendorong konsolidasi server pusat data modern. Penggunaan penting lainnya melibatkan pengujian dan eksplorasi arsitektur sistem operasi baru. Virtualisasi juga digunakan untuk menjaga kompatibilitas versi sistem operasi yang lama.

2.6.3 Lingkungan Sandboxing

Tersirat dalam konsep konsolidasi adalah premis bahwa mesin virtual independen disimpan terpisah satu sama lain dengan aman (juga disebut sebagai isolasi mesin virtual). Kemampuan untuk menjamin pemisahan sangat bergantung pada ketahanan perangkat lunak hypervisor yang mendasarinya. Seperti yang telah kita diskusikan sebelumnya dan akan segera diperluas, para peneliti telah menemukan kelemahan pada hypervisor komersial yang melanggar asumsi pemisahan ini. Namun demikian, penerapan teoretis yang penting dari kompartementalisasi mesin virtual adalah untuk mengisolasi perangkat lunak yang tidak tepercaya. Misalnya, browser web yang terhubung ke Internet dapat dimasukkan ke dalam kotak pasir (sandbox) di mesin virtual sehingga malware atau kerentanan browser yang ditularkan melalui Internet tidak dapat menyusup atau berdampak buruk pada lingkungan sistem operasi utama pengguna.

2.6.4 Peralatan Keamanan Virtual

Contoh lainnya, alat keamanan virtual, melakukan hal yang sebaliknya: sandbox, atau perangkat lunak tepercaya yang terpisah dari lingkungan sistem operasi utama sistem tertanam. Mari pertimbangkan perangkat lunak anti-virus yang berjalan di perangkat seluler. Beberapa tahun yang lalu, Trojan “Metal Gear” mampu menyebarkan dirinya ke seluruh ponsel berbasis sistem operasi Symbian dengan menonaktifkan perangkat lunak anti-malware mereka. Virtualisasi dapat mengatasi masalah ini dengan menempatkan perangkat lunak anti-malware ke dalam mesin virtual terpisah (lihat Gambar 2.18).



Gambar 2.18 Peningkatan keamanan menggunakan peralatan virtual yang terisolasi.

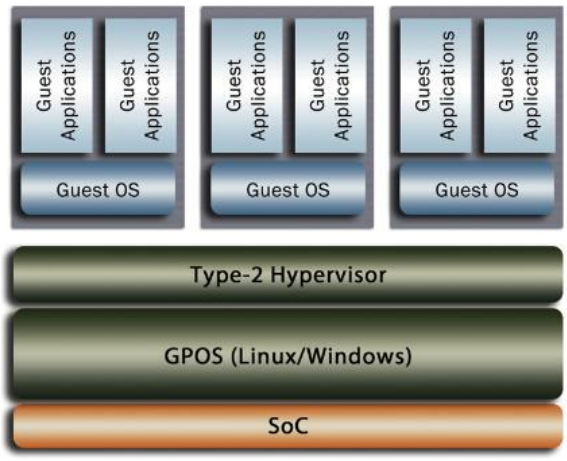
Peralatan virtual dapat menganalisis data yang masuk dan keluar dari lingkungan aplikasi utama atau terhubung ke sistem operasi platform untuk pemrosesan berdasarkan permintaan.

2.6.5 Arsitektur Hypervisor

Hypervisor ditemukan dalam berbagai rasa. Beberapa di antaranya bersifat open source; yang lainnya merupakan hak milik. Beberapa menggunakan hypervisor tipis yang ditambah dengan sistem operasi tamu khusus. Lainnya menggunakan hypervisor monolitik yang sepenuhnya mandiri. Di bagian ini, kami membandingkan dan membedakan teknologi yang tersedia, dengan penekanan pada dampak keamanan. Perhatikan bahwa bagian ini membandingkan dan membedakan apa yang disebut hypervisor Tipe-1 yang dijalankan pada bare metal. Hypervisor tipe-2 berjalan di atas sistem operasi tujuan umum, seperti Windows atau Linux, yang menyediakan I/O dan layanan lain atas nama hypervisor (lihat Gambar 2.19).

Karena mereka tidak lebih aman daripada sistem operasi host tujuan umum yang mendasarinya (yang diketahui rentan), hypervisor Tipe-2 tidak cocok untuk penerapan misi-kritis dan secara historis dihindari dalam lingkungan seperti itu.

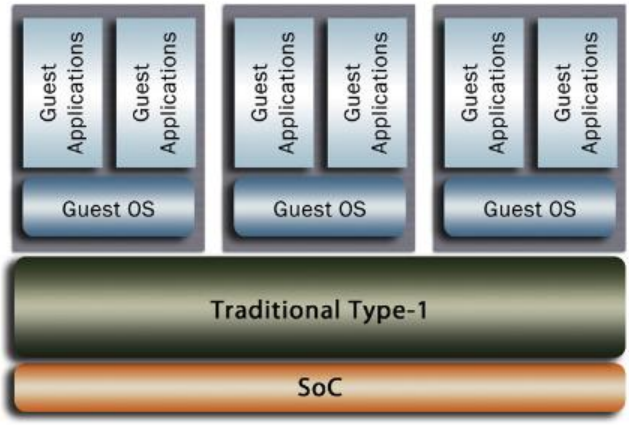
Dengan demikian, teknologi Tipe-2 dihilangkan dari pembahasan berikut.



Gambar 2.19: Arsitektur hypervisor tipe-2.

Hypervisor Monolitik

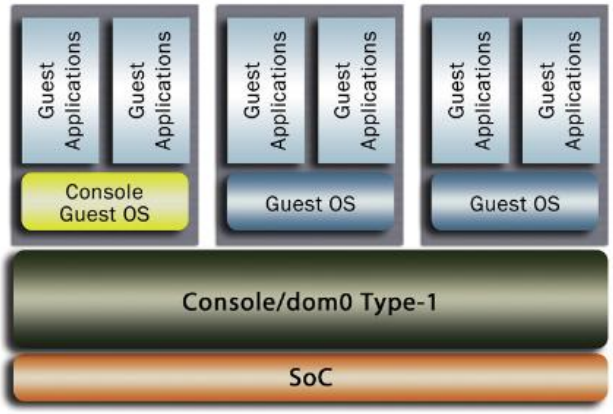
Arsitektur hypervisor paling sering menggunakan arsitektur monolitik, seperti yang ditunjukkan pada Gambar 2.20. Mirip dengan sistem operasi monolitik, hypervisor monolitik memerlukan sejumlah besar perangkat lunak operasi, termasuk driver perangkat dan middleware, untuk mendukung eksekusi satu atau lebih lingkungan tamu. Selain itu, arsitektur monolitik sering kali menggunakan satu komponen virtualisasi untuk mendukung banyak lingkungan tamu. Dengan demikian, satu kelemahan pada hypervisor dapat mengakibatkan kompromi pemisahan lingkungan tamu mendasar yang dimaksudkan oleh virtualisasi.



Gambar 2.20 Arsitektur hypervisor tipe-1 monolitik tradisional.

Hypervisor Tamu Konsol

Pendekatan alternatif menggunakan hypervisor yang dipangkas yang berjalan dalam mode mikroprosesor yang paling istimewa namun menggunakan partisi sistem operasi tamu khusus untuk menangani kontrol I/O dan layanan untuk sistem operasi tamu lainnya (lihat Gambar 2.21). Contoh arsitektur ini termasuk Xen dan Microsoft Hyper-V. Xen memelopori pendekatan konsol tamu di perusahaan; dalam Xen, tamu konsol disebut Domain 0, atau disingkat Dom0. Oleh karena itu, arsitektur tamu konsol terkadang disebut sebagai arsitektur Dom0. Dengan pendekatan konsol tamu, sistem operasi tujuan umum tetap harus diandalkan untuk keamanan sistem. Tamu konsol pada umumnya seperti Linux dapat menambahkan lebih banyak kode ke lapisan virtualisasi daripada yang ditemukan di hypervisor monolitik.

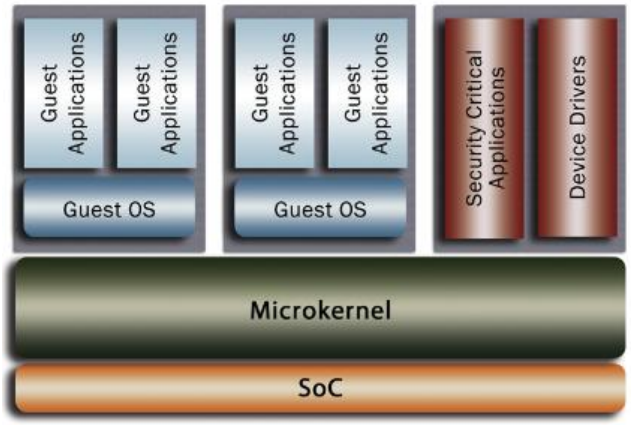


Gambar 2.21 Tamu konsol atau arsitektur hypervisor Dom0.

Hypervisor berbasis mikrokernel

Hypervisor berbasis mikrokernel, arsitektur Tipe-1, dirancang khusus untuk memberikan pemisahan yang kuat antara lingkungan tamu.

Gambar 2.22 menunjukkan arsitektur hypervisor berbasis mikrokernel. Karena mikrokernel adalah lapisan logam tipis, hypervisor berbasis mikrokernel dianggap sebagai arsitektur Tipe-1.



Gambar 2.22 Arsitektur hypervisor Tipe-1 berbasis mikrokernel.

Arsitektur ini menambahkan virtualisasi komputer sebagai layanan di atas mikrokernel tepercaya. Dalam beberapa kasus, komponen virtualisasi terpisah digunakan untuk setiap lingkungan tamu. Dengan demikian, lapisan virtualisasi hanya perlu memenuhi tingkat ketahanan yang setara (dan, biasanya, relatif rendah) dari tamu itu sendiri. Dalam arsitektur mikrokernel, hanya mikrokernel tepercaya yang berjalan dalam mode hak istimewa tertinggi. Contoh hypervisor tertanam yang menggunakan pendekatan mikrokernel termasuk INTEGRITY Multivisor dari Green Hills Software dan beberapa varian mikrokernel L4 standar terbuka.

2.6.6 Paravirtualisasi

Virtualisasi sistem dapat diimplementasikan dengan virtualisasi penuh atau paravirtualisasi, istilah yang pertama kali diciptakan pada proyek Denali tahun 2001.[9] Dengan virtualisasi penuh, sistem operasi tamu yang tidak dimodifikasi dapat didukung. Dengan paravirtualisasi, sistem operasi tamu dimodifikasi untuk meningkatkan kemampuan hypervisor yang mendasarinya untuk mencapai fungsi yang diinginkan.

Paravirtualisasi terkadang mampu meningkatkan kinerja. Misalnya, driver perangkat di sistem operasi tamu dapat dimodifikasi untuk memanfaatkan langsung perangkat keras I/O alih-alih memerlukan akses I/O untuk dijebak dan ditiru oleh hypervisor. Paravirtualisasi mungkin diperlukan pada arsitektur CPU yang tidak memiliki fitur akselerasi virtualisasi perangkat keras.

Keuntungan utama virtualisasi penuh dibandingkan paravirtualisasi adalah kemampuan untuk menggunakan versi sistem operasi tamu yang tidak dimodifikasi yang memiliki

[9] Whitaker A, et al. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Boston, MA: USENIX Annual Technical Conference; June 10e15, 2002.

silsilah yang terbukti dan tidak memerlukan pemeliharaan yang terkait dengan modifikasi khusus.

Penghematan pemeliharaan ini sangat penting terutama di perusahaan yang menggunakan berbagai sistem operasi dan/atau secara teratur melakukan upgrade ke versi sistem operasi baru dan rilis patch.

2.6.7 Memanfaatkan Bantuan Perangkat Keras untuk Virtualisasi

Penambahan bantuan perangkat keras CPU untuk virtualisasi sistem telah menjadi kunci penerapan praktis hypervisor dalam sistem tertanam.

Intel VT, pertama kali dirilis pada tahun 2005, telah menjadi faktor kunci dalam meningkatnya adopsi virtualisasi penuh di dunia komputasi perusahaan. Teknologi Virtualisasi untuk x86 (VT-x) menyediakan sejumlah kemampuan bantuan hypervisor, termasuk mode hypervisor perangkat keras sebenarnya yang memungkinkan sistem operasi tamu yang tidak dimodifikasi untuk mengeksekusi dengan hak istimewa yang dikurangi. Misalnya, VT-x akan mencegah sistem operasi tamu mereferensikan memori fisik melebihi apa yang telah dialokasikan ke mesin virtual tamu. Selain itu, VT-x memungkinkan injeksi pengecualian selektif sehingga kelas pengecualian yang ditentukan hypervisor dapat ditangani langsung oleh sistem operasi tamu tanpa menimbulkan biaya tambahan dari perangkat lunak hypervisor yang dimasukkan. Meskipun teknologi VT menjadi populer di chipset Intel kelas server, teknologi VT-x yang sama kini juga tersedia di prosesor tertanam dan seluler Intel Atom.

Pada tahun 2009, badan tata kelola Arsitektur Daya, Power.org, menambahkan virtualisasi ke spesifikasi tertanam dalam Arsitektur Daya versi 2.06 Arsitektur Set Instruksi (ISA). Pada saat penulisan ini, Freescale Semiconductor adalah satu-satunya vendor mikroprosesor tertanam yang telah merilis produk, termasuk prosesor jaringan multicore QorIQ P4080 dan P5020, yang mendukung spesifikasi virtualisasi tertanam ini.

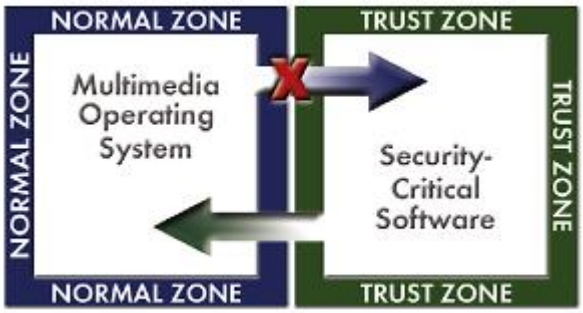
Pada tahun 2010, ARM Ltd. mengumumkan penambahan ekstensi virtualisasi perangkat keras ke arsitektur ARM serta inti ARM pertama, Cortex A15, yang mengimplementasikannya. Penerima lisensi yang diumumkan secara publik berencana untuk membuat SoC tertanam berdasarkan Cortex A15 termasuk Texas Instruments, Nvidia, Samsung, dan ST-Ericsson.

Sebelum munculnya ekstensi virtualisasi perangkat keras ini, virtualisasi penuh hanya mungkin dilakukan dengan menggunakan terjemahan biner dinamis dan teknik penulisan ulang instruksi yang sangat kompleks dan tidak mampu bekerja cukup dekat dengan kecepatan asli agar praktis dalam sistem tertanam. Misalnya, desktop berbasis x86 era 2005 yang menjalankan teknologi virtualisasi pra-VT Green Hills Software mampu mendukung tidak lebih dari dua klip audio/video gerak penuh secara bersamaan (masing-masing dalam mesin virtual terpisah) tanpa menghilangkan frame. Dengan implementasi berbasis VT-x dari Green Hills Software pada desktop kelas serupa, hanya total RAM yang tersedia untuk menampung beberapa mesin virtual yang umumnya membatasi jumlah klip secara bersamaan. Tolok ukur virtualisasi x86 secara umum menunjukkan perkiraan kinerja dua kali lipat menggunakan VT-x dibandingkan dengan platform sebelum VT. Selain itu, lapisan perangkat lunak virtualisasi disederhanakan karena kemampuan VT-x.

Zona Kepercayaan ARM

Kemampuan virtualisasi yang sering diabaikan dan diremehkan dalam mikroprosesor ARM modern adalah ARM TrustZone.

TrustZone memungkinkan bentuk virtualisasi sistem khusus berbasis perangkat keras. TrustZone menyediakan dua zona: zona "normal" dan zona "kepercayaan" atau "aman". Dengan TrustZone, sistem operasi multimedia (misalnya, apa yang biasanya dilihat pengguna di ponsel cerdas) berjalan di zona normal sementara perangkat lunak penting keamanan berjalan di zona aman. Meskipun perangkat lunak mode pengawas zona aman dapat mengakses memori zona normal, hal sebaliknya tidak dapat dilakukan (lihat Gambar 2.23). Dengan demikian, zona normal bertindak sebagai mesin virtual yang dikendalikan oleh hypervisor yang berjalan di zona kepercayaan. Namun, tidak seperti teknologi virtualisasi perangkat keras lainnya seperti Intel VT, sistem operasi tamu zona normal tidak menimbulkan overhead eksekusi dibandingkan berjalan tanpa TrustZone. Dengan demikian, TrustZone menghilangkan hambatan kinerja (dan bisa dibilang hambatan terbesar) dalam penerapan virtualisasi sistem pada perangkat tertanam dengan sumber daya terbatas.



Gambar 2.23 Zona Kepercayaan ARM.

TrustZone adalah kemampuan yang melekat pada inti prosesor aplikasi ARM modern, termasuk ARM1176, Cortex A5, Cortex A8, Cortex A9, dan Cortex A15. Namun, penting untuk dicatat bahwa tidak semua SoC yang menggunakan inti ini sepenuhnya mengaktifkan TrustZone. Pabrikan chip harus mengizinkan partisi zona aman pada memori dan interupsi periferal I/O di seluruh kompleks SoC. Selain itu, penyedia chip harus membuka zona aman untuk sistem operasi dan aplikasi pihak ketiga yang tepercaya. Contoh SoC seluler yang mendukung TrustZone adalah Freescale i.MX53 (Cortex A8) dan Texas Instruments OMAP 4430 (Cortex A9).

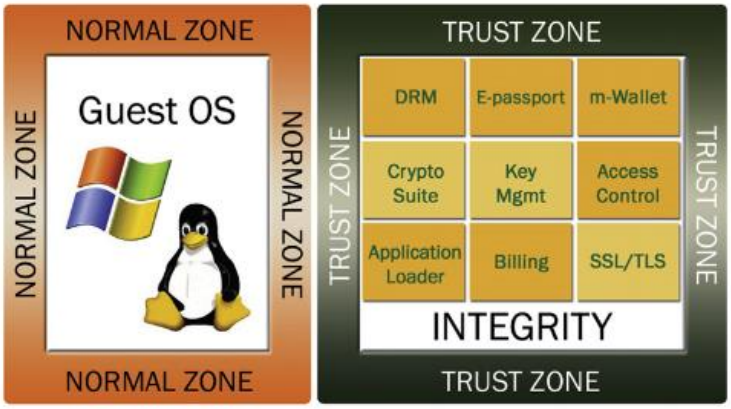
Perangkat lunak tepercaya mungkin mencakup algoritma kriptografi, protokol keamanan jaringan (seperti SSL/TLS) dan materi kunci; perangkat lunak manajemen hak digital (DRM); keypad virtual untuk input kredensial jalur tepercaya; subsistem pembayaran seluler; data identitas elektronik; dan hal lain yang dianggap layak dilindungi oleh penyedia layanan, produsen perangkat seluler, dan/atau pemasok SoC seluler dari lingkungan pengguna.

Selain meningkatkan keamanan, TrustZone dapat mengurangi biaya dan waktu pemasaran perangkat seluler yang memerlukan sertifikasi untuk digunakan di perbankan dan industri penting lainnya. Dengan TrustZone, bank (atau otoritas sertifikasi) dapat membatasi

ruang lingkup sertifikasi pada zona aman dan menghindari kerumitan (jika bukan ketidaklayakan) dalam mensertifikasi lingkungan sistem operasi multimedia.

Sistem operasi zona aman dapat mengurangi biaya dan waktu sertifikasi, karena dua alasan utama. Pertama, karena sistem operasi yang disertifikasi sudah dipercaya, dengan desain dan artefak pengujiannya tersedia bagi otoritas sertifikasi, biaya dan waktu untuk mensertifikasi lingkungan pengoperasian zona aman dapat dihindari.

Kedua, karena zona aman adalah inti ARM logis yang lengkap, sistem operasi aman dapat menggunakan kemampuan partisi unit manajemen memori (MMU) untuk membagi lebih lanjut zona aman menjadi zona meta (lihat Gambar 2.24). Misalnya, bank mungkin memerlukan sertifikasi meta-zona kriptografi yang digunakan untuk mengautentikasi dan mengenkripsi pesan transaksi perbankan, namun bank tidak akan peduli dengan sertifikasi meta-zona DRM multimedia, yang meskipun penting untuk perangkat secara keseluruhan, namun tidak digunakan. dalam transaksi perbankan dan dijamin oleh sistem operasi yang aman tidak mengganggu.



Gambar 2.24 Implementasi virtualisasi TrustZone dengan zona meta di dalam TrustZone.

SoC yang mendukung TrustZone mampu mempartisi periferal dan menginterupsi antara kondisi aman dan normal. Sistem operasi tujuan umum zona normal seperti Android tidak dapat mengakses periferal yang dialokasikan ke zona aman dan tidak akan pernah melihat interupsi perangkat keras yang terkait dengan periferal tersebut. Selain itu, periferal apa pun yang dialokasikan ke zona normal tidak dapat mengakses memori di zona normal.

2.6.8 Keamanan Hypervisor

Beberapa orang menyebut virtualisasi sebagai teknik “pertahanan berlapis” untuk keamanan sistem. Teori ini mendalilkan bahwa karena hanya sistem operasi tamu yang terkena ancaman eksternal, penyerang yang menembus tamu tersebut tidak akan mampu menumbangkan sistem lainnya. Intinya, perangkat lunak virtualisasi menyediakan fungsi isolasi yang serupa dengan model proses yang disediakan oleh sebagian besar sistem operasi modern.

Namun, produk virtualisasi perusahaan pada umumnya belum memenuhi persyaratan keamanan tingkat tinggi dan tidak pernah dirancang atau dimaksudkan untuk memenuhi tingkat ini. Oleh karena itu, tidak mengherankan jika teori keamanan melalui virtualisasi tidak memiliki bukti keberadaan dalam implementasi perusahaan pada umumnya. Sebaliknya,

sejumlah penelitian tentang keamanan virtualisasi dan subversi hypervisor yang berhasil telah dipublikasikan.

SubVirt

Pada tahun 2006, proyek SubVirt Samuel King mendemonstrasikan rootkit hypervisor yang menumbangkan VMware dan Microsoft VirtualPC.[10]

Pil Biru

Proyek Blue Pill membawa eksploitasi hypervisor selangkah lebih maju dengan mendemonstrasikan muatan malware yang merupakan hypervisor yang dapat diinstal dengan cepat, di bawah sistem operasi Windows yang berjalan secara native. Boot aman dan pengesahan platform (dibahas nanti dalam bab ini) diperlukan untuk mencegah hypervisor ditumbangkan dengan cara ini.

Ormandia

Tavis Ormandy melakukan studi empiris terhadap kerentanan hypervisor. Tim peneliti Ormandy menghasilkan aktivitas I/O acak ke dalam hypervisor, mencoba memicu error atau perilaku anomali lainnya. Proyek ini menemukan kerentanan di QEMU, VMware Workstation and Server, Bochs, dan sepasang produk hypervisor berpemilik yang tidak disebutkan namanya.[11]

Trilogi Pemilik Xen

Pada konferensi Black Hat tahun 2008, peneliti keamanan Joanna Rutkowska dan timnya mempresentasikan temuan mereka dari proyek penelitian singkat untuk menemukan kerentanan di Xen. Salah satu hipotesisnya adalah bahwa Xen cenderung tidak memiliki kerentanan yang serius, dibandingkan dengan VMware dan Microsoft Hyper-V, karena fakta bahwa Xen adalah teknologi open source dan oleh karena itu mendapat manfaat dari paparan basis kode yang “banyak mata”.

Tim Rutkowska menemukan tiga kerentanan berbeda yang dapat dieksploitasi sepenuhnya, yang digunakan para peneliti untuk menyita komputer melalui hypervisor.[12] Ironisnya, salah satu serangan ini memanfaatkan cacat buffer overflow pada lapisan Flask Xen. Flask adalah kerangka keamanan, sama dengan yang digunakan di SELinux, yang ditambahkan ke Xen untuk meningkatkan keamanan. Hal ini lebih lanjut menggarisbawahi sebuah prinsip penting: perangkat lunak yang belum dirancang dan dievaluasi dengan jaminan tingkat tinggi harus dianggap dapat ditumbangkan oleh entitas yang memiliki tekad dan sumber daya yang baik.

Sertifikasi Keamanan VMware dan Postingan Kerentanan Selanjutnya

Seiring dengan berkembangnya penerapan virtualisasi VMware di pusat data, para pakar keamanan telah menyuarakan keprihatinan mengenai implikasi “VM sprawl” dan kemampuan teknologi virtualisasi untuk menjamin keamanan. Pada tanggal 2 Juni 2008, VMware berusaha menghilangkan kekhawatiran ini dengan mengumumkan bahwa produk hypervisornya telah mencapai sertifikasi keamanan Common Criteria EAL 4+. Siaran pers

---

[10] King S, Peter C, Yi-Min W, Chad V, Helen JW, Jacob RL. SubVirt: Implementing Malware with Virtual Machines. Berkeley, CA: IEEE Symposium on Security and Privacy; May 21e24, 2006.

[11] Ormandy T. An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments, http://taviso.decsystem.org/virtsec.pdf; 2006.

[12] Rutkowska J, Tereshkin A, Wojtczuk R. “Detecting and Preventing the Xen Hypervisor Subversions,” “Bluepilling the Xen Hypervisor,” “Subverting the Xen Hypervisor,”. Las Vegas, NV: Black Hat USA; August 7, 2008..

VMware mengklaim bahwa produk virtualisasinya kini dapat digunakan "untuk lingkungan pemerintahan yang sensitif dan menuntut keamanan paling ketat."

Pada tanggal 5 Juni, hanya tiga hari kemudian, kerentanan parah pada hypervisor VMware bersertifikat telah diposting ke Database Kerentanan Nasional. Di antara kendala lainnya, kerentanan "memungkinkan pengguna sistem operasi tamu untuk mengeksekusi kode arbitrer."[13]

Produk virtualisasi VMware terus memiliki kerentanan yang parah, misalnya CVE-2009-3732, yang memungkinkan penyerang jarak jauh mengeksekusi kode arbitrer. Jelas, risiko "melarikan diri" dari lapisan mesin virtual, yang mengekspos semua tamu, sangatlah nyata. Hal ini terutama berlaku untuk hypervisor yang bercirikan basis kode monolitik.

Penting bagi pengembang untuk memahami bahwa penggunaan hypervisor tidak berarti isolasi yang sangat terjamin antara mesin virtual, tidak lebih dari penggunaan sistem operasi dengan perlindungan memori berarti isolasi proses yang terjamin dan keamanan sistem secara keseluruhan.

Bab 3 membahas standar evaluasi keamanan Kriteria Umum dan Tingkat Jaminan yang Dievaluasi, serta implikasinya dalam kaitannya dengan jaminan keamanan.

2.7 VIRTUALISASI I/O

Salah satu dampak terbesar terhadap keamanan dan efisiensi dalam virtualisasi sistem adalah pendekatan pengelolaan I/O di seluruh mesin virtual.

Bagian ini membahas beberapa kelemahan keamanan dan tren yang muncul dalam virtualisasi I/O tertanam.

2.7.1 Kebutuhan I/O Bersama

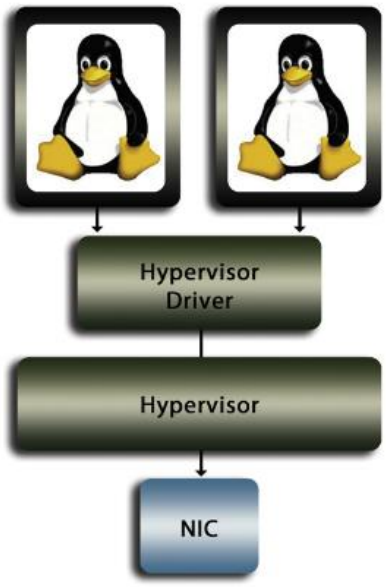
Dalam sistem tertanam apa pun, selalu ada kebutuhan untuk berbagi perangkat I/O fisik terbatas di seluruh beban kerja. Sistem operasi tertanam menyediakan abstraksi, seperti soket lapisan dua, tiga, dan empat, untuk tujuan ini. Soket pada dasarnya menyediakan antarmuka virtual untuk setiap proses yang memerlukan penggunaan perangkat antarmuka jaringan bersama. Demikian pula, dalam sistem tervirtualisasi, hypervisor harus mengambil peran menyediakan antarmuka virtual yang aman untuk mengakses perangkat I/O fisik bersama. Tantangan tersulit dalam virtualisasi tertanam adalah tugas mengalokasikan, melindungi, berbagi, dan memastikan efisiensi I/O di seluruh mesin dan aplikasi virtual.

2.7.2 Emulasi

Metode tradisional virtualisasi I/O adalah emulasi: semua akses sistem operasi tamu ke sumber daya I/O perangkat disadap, divalidasi, dan diterjemahkan ke dalam operasi yang dimulai oleh hypervisor (lihat Gambar 2.25). Metode ini memaksimalkan keandalan, keamanan, dan kemudahan berbagi. Sistem operasi tamu tidak akan pernah merusak sistem melalui perangkat I/O karena semua akses I/O dilindungi melalui driver perangkat hypervisor tepercaya. Satu perangkat dapat dengan mudah dimultipleks di beberapa mesin virtual, dan jika satu mesin virtual gagal, mesin virtual lainnya dapat terus menggunakan perangkat I/O fisik yang sama, sehingga memaksimalkan ketersediaan sistem. Kelemahan terbesarnya adalah efisiensi; lapisan emulasi menyebabkan overhead yang signifikan pada semua operasi

[13] CVE-2008-2100. National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/detail?vulnId¼CVE- 2008-2100.

I/O. Selain itu, vendor hypervisor harus mengembangkan dan memelihara driver perangkat secara independen dari driver sistem operasi tamu.

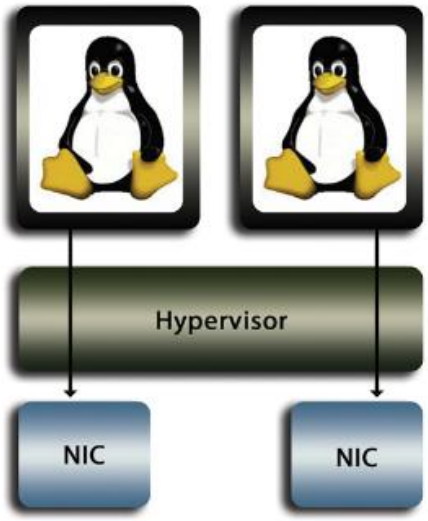


Gambar 2.25 Virtualisasi I/O menggunakan emulasi.

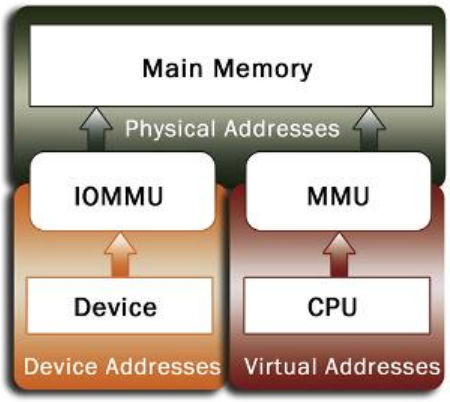
2.7.3 Model Pass-through

Sebaliknya, model pass-through (lihat Gambar 2.26) memberikan sistem operasi tamu akses langsung ke perangkat I/O fisik. Tergantung pada CPU, driver tamu dapat digunakan tanpa modifikasi atau dengan paravirtualisasi minimal. Satu perangkat dapat dibagikan kepada beberapa tamu dengan menyediakan antarmuka I/O virtual antara tamu yang memiliki perangkat fisik dan tamu lain yang memerlukan akses ke perangkat tersebut. Untuk perangkat jaringan, antarmuka virtual ini sering disebut saklar virtual (lapisan 2) dan merupakan fitur umum dari sebagian besar hypervisor. Model pass-through memberikan peningkatan efisiensi namun mengorbankan ketahanan: akses yang tidak tepat oleh tamu dapat melumpuhkan tamu lain, atau aplikasi, atau keseluruhan sistem. Model ini melanggar kebijakan keamanan utama virtualisasi sistem: isolasi lingkungan virtual untuk hidup berdampingan secara aman dari beberapa sistem operasi pada satu komputer.

Jika ada, IOMMU memungkinkan model virtualisasi I/O pass-through tanpa risiko akses memori langsung di luar memori yang dialokasikan mesin virtual. Karena MMU memungkinkan hypervisor untuk membatasi akses memori mesin virtual, IOMMU membatasi akses memori I/O (terutama DMA), baik yang berasal dari perangkat lunak yang berjalan di mesin virtual atau periferal eksternal itu sendiri (lihat Gambar 2.27).



Gambar 2.26 Virtualisasi I/O menggunakan pass-through.



Gambar 2.27 IOMMU digunakan untuk sandbox akses memori terkait perangkat.

IOMMU menjadi semakin umum pada mikroprosesor tertanam, seperti Intel Core, Freescale QorIQ, dan ARM Cortex A15. Dalam prosesor Intel, IOMMU disebut sebagai Intel Virtualization Technology for Directed I/O (Intel VT-d). Pada prosesor QorIQ yang mendukung virtualisasi Freescale seperti P4080, IOMMU disebut sebagai Peripheral Access Management Unit (PAMU). Pada Cortex A15 (dan inti ARM lainnya yang mendukung Ekstensi Virtualisasi ARM[14]), IOMMU bukan bagian dari spesifikasi virtualisasi dasar. Sebaliknya, ARM Ltd. memiliki penawaran kekayaan intelektual terpisah, yang disebut Sistem MMU (SMMU), yang secara opsional dapat dilisensikan oleh produsen semikonduktor ARM. Selain itu, pabrikan dapat menggunakan implementasi IOMMU khusus alih-alih MMU Sistem ARM. Selain itu, ARM TrustZone menyediakan bentuk IOMMU antara zona normal dan aman dari prosesor ARM; akses zona normal yang dilakukan oleh CPU atau periferal yang dialokasikan ke zona normal dilindungi terhadap akses memori di zona aman.

Model IOMMU memungkinkan efisiensi kinerja yang sangat baik dengan peningkatan ketahanan dibandingkan model pass-through tanpa IOMMU. Namun, IOMMU merupakan konsep yang relatif baru.

[14]Mijat R, Nightingale A. Virtualization Is Coming to a Platform Near You. White Paper. ARM Ltd; 2010.
.

Sejumlah kerentanan (cara untuk menghindari perlindungan) telah ditemukan di IOMMU dan harus diatasi secara hati-hati dengan bantuan pemasok perangkat lunak/hipervisor sistem.

Dalam sebagian besar kasus kerentanan, tamu yang salah atau jahat dapat membahayakan keamanan melalui operasi tingkat perangkat, bus, atau chipset selain akses memori langsung. Para peneliti di Green Hills Software, misalnya, telah menemukan cara bagi sistem operasi tamu untuk mengakses memori di luar mesin virtualnya, menolak layanan eksekusi ke mesin virtual lain, memasang hypervisor jahat di bawah hypervisor sistem default (serangan yang dijelaskan nanti dalam bab ini), dan mematikan seluruh komputer melalui perangkat I/O yang dilindungi IOMMU. Untuk aplikasi dengan keandalan tinggi dan/atau keamanan penting, IOMMU harus diterapkan dengan cara yang berbeda dari pendekatan pass-through tradisional di mana sistem operasi tamu memiliki akses tak terbatas ke perangkat I/O. Banyaknya trade-off dalam penggunaan IOMMU berada di luar cakupan buku ini; konsultasikan dengan vendor hypervisor untuk memahami opsi penggunaan IOMMU dan virtualisasi I/O secara umum.

Kelemahan utama dari pendekatan pass-through (dengan atau tanpa IOMMU) adalah pendekatan ini mencegah pembagian yang kuat dari satu perangkat I/O di beberapa mesin virtual. Mesin virtual yang ditetapkan kepemilikan perangkat pass-through memiliki akses eksklusif, dan mesin virtual lainnya harus bergantung pada mesin virtual pemilik untuk meneruskan I/O. Jika mesin virtual pemilik disusupi, semua mesin virtual akan ditolak untuk diservis untuk perangkat tersebut.

2.7.4 IOMMU Bersama

Kekurangan ini menyebabkan munculnya teknologi yang menyediakan kemampuan untuk berbagi satu perangkat I/O di beberapa sistem operasi tamu menggunakan IOMMU dan mekanisme partisi perangkat keras yang dibangun ke dalam kompleks I/O perangkat (misalnya, chipset ditambah perangkat itu sendiri). Salah satu contoh perangkat pass-through yang dapat dibagikan dan berkemampuan IOMMU adalah prosesor Intel yang dilengkapi dengan Teknologi Virtualisasi Intel untuk Konektivitas (Intel VT-c) ditambah dengan kartu Ethernet PCI-express yang mengimplementasikan Virtualisasi I/O Single-Root (SR-IOV), standar PCI-SIG. Dengan sistem seperti itu, perangkat keras akan menyediakan sumber daya I/O independen, seperti beberapa cincin buffer paket, dan beberapa bentuk layanan eksekusi berkualitas di antara mesin virtual. Mekanisme ini cocok untuk perangkat jaringan seperti Ethernet, Rapid I/O, dan Fibre Channel; namun, pendekatan lain diperlukan untuk berbagi periferal yang aman dan independen seperti kartu grafis, keyboard, dan port serial. Namun demikian, kemungkinan besar teknologi perangkat jaringan yang mendukung perangkat keras, dilindungi IOMMU, dan dapat dibagikan akan semakin populer di seluruh prosesor tertanam.

2.7.5 IOMMU dan Driver Perangkat Virtual

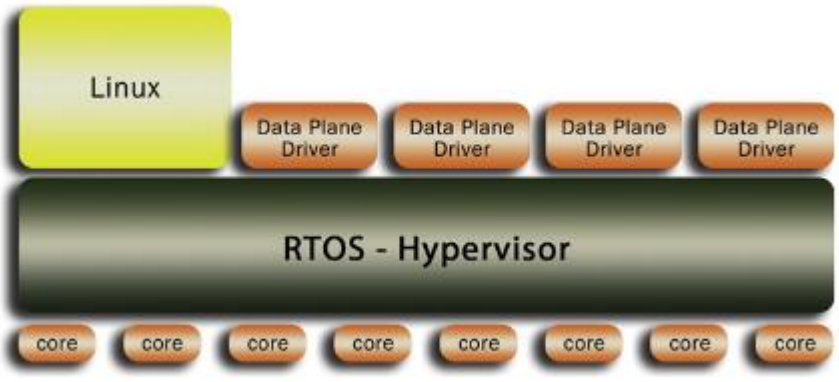
Sebelumnya di bab ini, kita telah membahas pentingnya driver perangkat virtual dalam membatasi hak istimewa dan mendorong desain sistem yang kuat. Driver perangkat virtual yang ideal memerlukan sangat sedikit kode khusus perangkat untuk berada dalam kernel mode supervisor: rutin layanan interupsi (ISR) dan, dalam kasus perangkat jaringan, akses ke register pemrograman akses memori langsung (DMA).

ISR harus ada di kernel (vektor interupsi dijalankan oleh perangkat keras dalam mode supervisor). Pemrograman DMA sering kali disimpan di dalam kernel karena pengoperasiannya harus dipercaya: akses ke register DMA memungkinkan driver untuk menimpa lokasi memori fisik mana pun, bahkan di kernel itu sendiri.

Sayangnya, pendekatan driver virtual masih menyisakan sedikit kode khusus perangkat di kernel. Untuk meningkatkan pemeliharaan dan arsitektur yang lebih bersih, akan lebih baik jika pemrograman DMA dapat berada dalam mode pengguna tanpa meningkatkan hak istimewa pengemudi. IOMMU memungkinkan pemrograman DMA berada di driver perangkat virtual. Mungkin yang paling penting, dengan mengaktifkan akses langsung ke register perangkat yang dipetakan memori dan pemrograman DMA, IOMMU mempromosikan bentuk arsitektur driver perangkat virtual yang lebih murni tanpa mengorbankan efisiensi kinerja.

2.7.6 Virtualisasi I/O yang Aman dalam Mikrokernel

Seperti dibahas sebelumnya, driver perangkat virtual biasanya digunakan oleh sistem operasi bergaya mikrokernel. Hypervisor berbasis mikrokernel juga cocok untuk mengamankan virtualisasi I/O: alih-alih pendekatan monolitik yang khas dengan menempatkan driver perangkat ke dalam hypervisor itu sendiri atau ke dalam sistem operasi tamu Linux tujuan khusus (metode Dom0 yang dijelaskan sebelumnya), mikrokernel-hypervisor berbasis menggunakan proses asli yang kecil dan memiliki hak istimewa yang lebih rendah untuk driver perangkat, multipleksor I/O, manajer kesehatan, manajer daya, dan fungsi pengawasan lainnya yang diperlukan dalam lingkungan virtual. Masing-masing aplikasi ini hanya disediakan sumber daya minimum yang diperlukan untuk mencapai fungsi yang dimaksudkan, sehingga mendorong desain sistem tertanam yang aman. Gambar 2.28 menunjukkan arsitektur tingkat sistem hypervisor berbasis mikrokernel yang digunakan dalam aplikasi jaringan multicore yang harus mengelola fungsionalitas bidang kendali Linux dengan aman bersama dengan pemrosesan paket bidang data dengan throughput tinggi dan latensi rendah dalam driver perangkat virtual.



Gambar 2.28 Driver perangkat virtual dalam arsitektur virtualisasi sistem berbasis mikrokernel.

Tanpa virtualisasi, aplikasi sebelumnya dapat diimplementasikan dengan konfigurasi ganda Linux/RTOS di mana sistem operasi bidang kontrol dan data terikat secara statis ke sekumpulan inti independen. Ini disebut pendekatan Asymmetric Multiprocessing (AMP). Salah satu keuntungan virtualisasi dibandingkan pembagian kerja AMP adalah fleksibilitas

dalam mengubah alokasi beban kerja bidang kontrol dan data ke inti. Misalnya, dalam mode operasi normal, arsitek mungkin ingin menggunakan hanya satu inti untuk kontrol dan semua inti lainnya untuk pemrosesan data. Namun, sistem dapat ditempatkan ke mode manajemen di mana Linux memerlukan empat inti (SMP) sementara pemrosesan data untuk sementara dibatasi. Lapisan virtualisasi dapat menangani realokasi inti dengan lancar, sesuatu yang tidak dapat didukung oleh sistem AMP statis.

Keamanan juga dapat ditingkatkan dengan menambahkan aplikasi, atau bahkan mesin virtual terpisah (konsep peralatan virtual yang dijelaskan sebelumnya dalam bab ini), yang menjalankan fungsi keamanan khusus seperti anti-malware atau firewall.

Peningkatan kompleksitas perangkat lunak dan sistem serta konektivitas mendorong evolusi dalam cara sistem tertanam mengelola I/O dan dalam arsitektur sistem operasi dan hypervisor yang bertanggung jawab untuk memastikan keamanannya. Kombinasi hak istimewa yang lebih rendah, desain berbasis komponen serta virtualisasi I/O yang cerdas untuk memungkinkan konsolidasi yang aman tanpa mengorbankan efisiensi akan tetap menjadi fokus pemasok perangkat lunak sistem dalam memenuhi tuntutan fleksibilitas, skalabilitas, dan ketahanan dari perangkat tertanam generasi berikutnya. sistem.

2.8 MANAJEMEN JARAK JAUH

Ketika sistem tertanam gagal di lapangan, pengembang (dan terkadang tim forensik pemerintah) ditugaskan untuk menentukan penyebab kegagalan tersebut. Perekam penerbangan adalah sistem diagnostik lapangan yang terkenal: produk akhir (pesawat) dikirimkan dengan kemampuan diagnostik bawaan (kotak hitam). Namun kelas perangkat tertanam yang sedang berkembang memerlukan kemampuan diagnostik dan manajemen lapangan. Berbeda dengan kotak hitam, yang merupakan alat forensik murni, koneksi jaringan aktif diperlukan di banyak sistem. Koneksi ini memungkinkan teknisi untuk memeriksa sistem yang ada untuk menemukan sumber perilaku anomali seperti hilangnya fungsi atau penurunan kinerja, menginstal patch atau peningkatan perangkat lunak lainnya, melakukan audit otomatis, mengubah konfigurasi, atau melaksanakan sejumlah tugas manajemen lainnya. Selain itu, dengan meningkatnya ketersediaan layanan jaringan yang dibangun ke dalam sistem tertanam, manajemen perangkat dapat dengan mudah dilakukan melalui Internet. Kabel atau kotak satelit di rumah memiliki koneksi jaringan yang kemungkinan besar telah digunakan untuk melakukan diagnostik jarak jauh dan peningkatan firmware. Fungsi manajemen perangkat telah mengalami transformasi, meningkatkan masa pakai produk, keandalan, kemudahan servis, dan kepuasan pelanggan sekaligus mengurangi biaya pemeliharaan dan total biaya kepemilikan.

Contoh bagus dari kekuatan manajemen jarak jauh adalah Mars Pathfinder: manajemen jarak jauh menyelamatkan misi tahun 1997 dari bencana ketika malfungsi didiagnosis hingga cacat perangkat lunak yang diperbaiki dengan patch yang dipasang melalui tautan radio dari Bumi.

2.8.1 Implikasi Keamanan

Ambisi peretas: menemukan kerentanan yang, jika dimanipulasi dengan benar, memberikan akses ke sistem komputer untuk tujuan jahat. Seiring waktu, eksploitasi jaringan jarak jauh menjadi semakin canggih. Pada bulan April 2010, peneliti keamanan IBM Mark

Dowd mendapat pujian dengan laporan setebal 25 halaman yang merinci serangkaian langkah berbelit-belit yang dapat diambil untuk mengeksploitasi kerentanan akses web, yang sebelumnya diyakini tidak berbahaya, dalam program Flash Adobe yang ada di mana-mana.

Manajemen perangkat jarak jauh adalah jawaban atas impian terliar para peretas: sistem tertanam tidak hanya dilengkapi dengan akses Internet, tetapi juga sarana untuk memodifikasi dan menambal perangkat lunak dari jarak jauh.

Tidak diperlukan vektor serangan Bizantium; melewati kontrol dasar sistem operasi, dan perangkat yang tertanam akan menjadi arena kejahatan.

Pada Bab 1, kami menjelaskan secara singkat kerentanan manajemen jarak jauh VxWorks di mana port diagnostik sistem operasi biasanya dibiarkan terbuka untuk diakses oleh peretas pemula untuk dieksploitasi. Kerentanan ini begitu luas sehingga kecil kemungkinan Internet akan sepenuhnya menghapus perangkat yang mengandung kelemahan ini. Dalam kasus kelemahan VxWorks, koneksi diagnostik jarak jauh memungkinkan peretas untuk menginstal malware, bahkan melakukan rooting atau mengganti sistem operasi itu sendiri (lihat Gambar 2.29). Pertahanan dasarnya adalah menjaga port manajemen jarak jauh dengan otentikasi yang kuat, menggunakan protokol keamanan jaringan standar seperti TLS/SSL atau IKE/IPsec (keduanya dijelaskan di Bab 5). Pertama, sistem tertanam jarak jauh harus mengautentikasi komputer yang digunakan oleh administrator jarak jauh. Hal ini memastikan bahwa hanya komputer administratif yang dikenal dan tepercaya yang digunakan untuk mengakses sistem tertanam untuk tujuan manajemen. Kedua, jika perintah manajemen dijalankan oleh operator manusia, maka operator tersebut harus diautentikasi secara kuat ke komputer manajemen secara lokal sebelum membuat sambungan jarak jauh. Setelah operator diautentikasi, SSL atau IPsec akan menggunakan enkripsi yang diautentikasi untuk melindungi integritas dan kerahasiaan perintah dan data manajemen jarak jauh.

Gambar 2.29 Penyisipan malware ke dalam sistem tertanam melalui port manajemen jarak jauh.

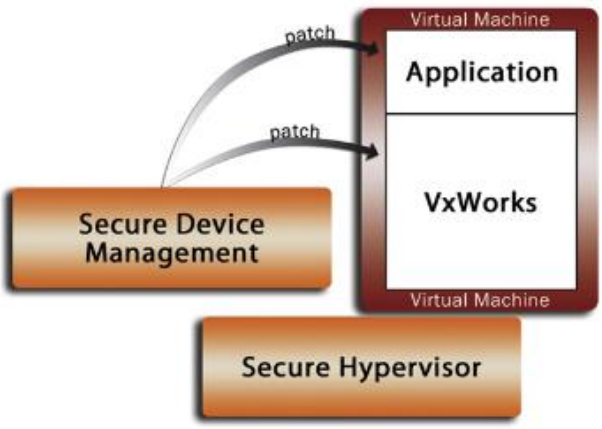
Tentu saja, meretas komputer manajemen dapat membuat perlindungan keamanan jaringan yang disebutkan di atas tidak berguna. Misalnya, malware dalam sistem operasi komputer host dapat mendukung koneksi terenkripsi untuk menyusup ke sistem tertanam. Kehati-hatian yang ekstrim harus diberikan untuk melindungi komputer manajemen dari Internet atau jaringan terbuka lainnya yang dapat diakses oleh peretas. Idealnya, stasiun kerja manajemen didedikasikan untuk tujuannya dan tidak pernah terhubung ke Internet. Menggunakan stasiun kerja operator yang tidak aman untuk mengelola sistem tertanam jarak jauh sama dengan memasang gembok pada brankas yang terbuat dari karton.

Salah satu contohnya ditemukan pada sistem manajemen jarak jauh paling terkenal di dunia: Pembaruan Windows. Pembaruan Windows dirancang untuk secara diam-diam dan jarak jauh memberi komputer pribadi patch keamanan terbaru yang tervalidasi. Namun peretas telah menyita fasilitas ini untuk mengunggah perangkat lunak yang tidak sah. Koneksi SSL hanyalah puncak gunung es dalam memastikan bahwa sistem tertanam dikembangkan dengan benar untuk memungkinkan manajemen jarak jauh yang aman. Ancaman orang dalam dan kelemahan pengembangan dapat menyebabkan sistem tertanam terkena kerentanan yang mengubah saluran manajemen jarak jauh menjadi metode akses peretas yang kuat. Proses pengembangan yang aman untuk mengatasi ancaman ini adalah fokus dari Bab 3. Tanpa jaminan yang tepat yang mencakup pengujian pada tingkat biner, pengiriman yang aman, dan kontrol lainnya, pengembang dapat memasukkan pintu belakang menggunakan berbagai teknik yang telah terbukti. Solusi manajemen perangkat yang aman mencegah penyisipan kode berbahaya dengan menggunakan otentikasi dan tanda tangan digital dengan jaminan tinggi: administrator TI, teknisi, petugas kebersihan, dan pengguna yang tidak berwenang tidak dapat menghindari kontrol akses wajib yang diberlakukan oleh sistem.

Meskipun pengembang dapat menerapkan proses pengembangan dengan jaminan tinggi untuk perangkat lunak mereka sendiri, bagaimana pengembang dapat melindungi terhadap kerentanan dalam sistem operasi pihak ketiga, yang banyak di antaranya dikirimkan dalam bentuk biner saja dan tidak memiliki sumber keamanan, silsilah, atau ganti rugi? Karena sistem operasi sering kali menyediakan kemampuan keamanan jaringan yang dijelaskan sebelumnya, hal ini jelas merupakan masalah kritis.

Virtualisasi sistem dapat memberikan solusi efektif terhadap masalah penggabungan atau retrofit manajemen jarak jauh yang aman ke sistem tertanam: lingkungan operasi lama diangkat ke dalam mesin virtual, diisolasi secara aman dari fungsi manajemen jarak jauh, seperti otentikasi koneksi dan manajemen konfigurasi, asalkan oleh hypervisor yang dapat dipercaya.

Faktanya, perangkat lunak manajemen perangkat dapat digunakan untuk memantau, mengkonfigurasi, dan melakukan patch pada kernel sistem operasi lama itu sendiri (lihat Gambar 2.30).



Gambar 2.30 Arsitektur manajemen perangkat untuk sistem lama.

Dalam banyak kasus, solusi manajemen perangkat yang aman melibatkan layanan konsultasi untuk memastikan bahwa rangkaian komponen keamanan yang sesuai terintegrasi dan diterapkan dengan cara yang kuat dan hemat biaya ke dalam perangkat akhir. Mengingat meningkatnya risiko finansial, keselamatan, dan keamanan yang terkait dengan akses jarak jauh, banyak pembuat perangkat tertanam dan seluler memikirkan kembali strategi pengelolaan perangkat mereka. Virtualisasi sistem adalah salah satu pendekatan yang berpotensi ampuh untuk mengatasi tantangan manajemen jarak jauh yang aman.

2.9 MENJAMIN INTEGRITAS TCB

2.9.1 Perangkat Keras dan Rantai Pasokan Tepercaya

Kini kita beralih ke ancaman keamanan tingkat sistem yang kritis yang dapat membuat perangkat lunak yang benar-benar aman sekalipun menjadi tidak berdaya. Keamanan sistem tertanam memerlukan integritas TCB (sistem operasi tepercaya serta semua aplikasi dan middleware tepercaya, termasuk protokol perlindungan data yang dibahas di Bab 5).

Misalnya, sistem operasi memori virtual bergantung pada fakta bahwa tidak ada entitas yang tidak sah yang dapat mengakses memori yang dialokasikan sistem operasi untuk proses tertentu menggunakan manajemen memori mikroprosesor dan perangkat keras perlindungan. Jika sistem operasi di-boot pada perangkat keras berbahaya yang mengekspos lokasi memori tertentu (misalnya, dengan mengirimkan konten memori tersebut melalui antarmuka Ethernet yang terpasang), maka jelas kebijakan keamanan sistem yang diterapkan oleh sistem operasi dapat diperdebatkan.

Serangan terhadap komponen perangkat keras, subsistem, dan periferal bukanlah sesuatu yang dibuat-buat. Banyak laporan tentang perangkat keras palsu dan dirusak telah terjadi selama bertahun-tahun. Pada tahun 2007, beberapa hard drive Maxtor/Seagate ditemukan memiliki virus yang telah diinstal sebelumnya (oleh produsen kontrak) yang akan mengirimkan data yang tersimpan di drive tersebut ke situs web berbahaya.[15] Sebuah “kill switch” tersembunyi dimasukkan ke dalam gerbang yang dapat diprogram di lapangan. array (FPGA) selama desainnya diduga digunakan untuk menonaktifkan sistem radar selama serangan militer.[16]

Pengembang sistem tertanam harus melakukan semua yang mereka bisa untuk memastikan kepercayaan rantai pasokan perangkat keras.

Rantai pasokan dan keamanan manufaktur merupakan topik yang rumit dan berada di luar cakupan buku ini. Kenyataannya adalah sebagian besar organisasi sistem tertanam melakukan yang terbaik yang mereka bisa dengan membeli dari pemasok yang dapat diandalkan dan mengambil risiko sehubungan dengan mempercayai perangkat keras.

2.9.2 Boot Aman

Selain perangkat keras tepercaya, kita harus memastikan bahwa firmware/perangkat lunak tepercaya tidak dapat ditumbangkan selama proses booting.

[15] Dowd M. Application-Specific Attacks: Leveraging the ActionScript Virtual Machine, http://documents.iss.net/whitepapers/IBM_X-Force_WP_final.pdf; April 2008.

[16] Maxtor Basics Personal Storage 3200 (PS 3200) virus [205131]. Seagate Knowledge Base, online URL, http:// seagate.custkb.com/seagate/crm/selfservice/search.jsp?DocId¼205131.

Serangan firmware platform jauh lebih mudah dan mahal untuk dilakukan dibandingkan serangan rantai pasokan dan merupakan ancaman penting yang harus dipertimbangkan oleh semua pengembang sistem tertanam. Tindakan menetapkan keadaan awal yang aman sering disebut sebagai boot aman.

Jika perangkat keras dan pemuat boot memiliki kemampuan untuk memuat firmware sistem (sistem operasi, hypervisor, seluruh TCB) dari perangkat alternatif, seperti USB, dan bukan perangkat tepercaya yang dimaksudkan (misalnya Flash), maka penyerang memiliki akses ke sistem dapat mem-boot sistem operasi jahat yang mungkin bertindak seperti sistem operasi tepercaya tetapi dengan perilaku jahat, seperti menonaktifkan layanan otentikasi jaringan atau menambahkan login pintu belakang. Alternatifnya, hypervisor jahat dapat di-boot, dan hypervisor kemudian dapat meluncurkan sistem operasi tepercaya dalam mesin virtual. Hypervisor jahat memiliki akses penuh ke RAM dan karenanya dapat secara diam-diam mengamati lingkungan tepercaya, mencuri kunci enkripsi, atau mengubah kebijakan keamanan sistem. King dkk., memberikan contoh bagus mengenai serangan ini dalam makalah yang menjelaskan SubVirt, sebuah hypervisor malware.10 Serangan terkenal lainnya, yang disebut Blue Pill, memperluas pendekatan SubVirt untuk membuat rootkit permanen yang dapat diluncurkan dengan mudah saat itu juga. menggunakan kelemahan pada sistem operasi Windows yang diinstal pabrik.[17]

2.9.3 Akar Kepercayaan Statis versus Dinamis

Di sebagian besar sistem tertanam, rangkaian firmware harus dijalankan untuk menetapkan keadaan awal yang aman di mana TCB aktif dan berjalan serta mengendalikan keamanan sistem. Umumnya, CPU pertama-tama mengeksekusi boot loader kecil, yang dibakar ke dalam ROM pada waktu produksi. Boot aman bergantung pada akar kepercayaan berbasis perangkat keras; dalam hal ini, kami bergantung pada fakta bahwa ROM tidak dapat dimodifikasi pasca produksi. ROM loader sering kali mem-boot boot loader tingkat kedua yang lebih fungsional yang berada di Flash internal. Misalnya, banyak sistem tertanam berbasis ARM menggunakan pemuat boot u-boot yang populer. Boot loader ini sering kali mem-boot sistem operasi utama atau hypervisor yang pada gilirannya mem-boot aplikasi tingkat yang lebih tinggi.

Metode boot aman yang umum adalah memverifikasi keaslian setiap komponen dalam rantai boot ini. Jika ada tautan dalam rantai yang putus, keadaan awal yang aman akan terganggu. ROM loader tahap pertama juga harus memiliki kunci kriptografi yang telah dibakar sebelumnya yang digunakan untuk memverifikasi tanda tangan digital dari boot loader tingkat berikutnya. Kunci ini dapat diintegrasikan ke dalam image ROM loader itu sendiri, dipasang menggunakan sekering yang dapat diprogram satu kali, atau disimpan dalam TPM lokal yang dapat memberikan perlindungan tamper yang lebih baik. Akar kepercayaan perangkat keras harus menyertakan kunci verifikasi awal ini. Bab 4 menjelaskan konsep tanda tangan digital, biasanya diimplementasikan dengan kriptografi kunci publik.

Kunci tanda tangan digunakan untuk memverifikasi keaslian komponen tahap kedua dalam rantai boot. Oleh karena itu, tanda tangan yang baik dan diketahui juga harus disimpan

[17]Adee S. The Hunt for the Kill Switch. IEEE Spectrum Magazine, http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch/; May 2008.

di area yang dilindungi perangkat keras. Verifikasi komponen tingkat kedua mencakup gambar yang dapat dieksekusi serta tanda tangan baik yang diketahui dan kunci verifikasi tanda tangan tahap ketiga, jika ada. Rantai verifikasi bisa sangat panjang. Bukan hal yang aneh jika beberapa sistem komputasi tertanam dan seluler yang canggih memiliki rantai panjang atau bahkan rangkaian komponen terverifikasi yang membentuk TCB. Gambar 2.31 menggambarkan contoh urutan boot aman tiga tingkat. Ketika rantai verifikasi dimulai saat pengaturan ulang sistem dan mencakup semua firmware yang dijalankan sebelum pembentukan kondisi stabil waktu proses, hal ini disebut sebagai akar kepercayaan statis.

Sebaliknya, akar kepercayaan dinamis memungkinkan sistem yang sudah berjalan (yang mungkin tidak berada dalam kondisi aman yang diketahui) untuk melakukan pengukuran rantai TCB dan kemudian mengatur ulang sebagian sumber daya komputer sehingga hanya rantai dinamis ini yang berkontribusi terhadap keamanan. keadaan awal. Akar kepercayaan yang dinamis memerlukan perangkat keras khusus, seperti Teknologi Eksekusi Tepercaya (TXT) dari Intel,[18] yang tersedia pada beberapa chipset tertanam berbasis Arsitektur Intel (pada saat penulisan artikel ini, yang lebih canggih). Dorongan utama di balik akar kepercayaan dinamis adalah menghapus komponen waktu booting yang besar, yang harus dijalankan untuk menginisialisasi komputer, dari TCB. Pada sistem berbasis Arsitektur Intel, BIOS sering kali merupakan perangkat lunak berukuran sangat besar yang digunakan untuk menginisialisasi sistem. Karena merupakan perangkat lunak monolitik yang besar, BIOS mungkin (dan dalam beberapa kasus telah terbukti) mengandung kerentanan yang dapat dieksploitasi. Dengan melakukan reset dinamis (terkadang juga disebut sebagai peluncuran terlambat) setelah BIOS menginisialisasi perangkat keras, menghapus semua hak istimewa dari lingkungan eksekusi BIOS, sistem secara teori telah mengurangi TCB-nya dan meningkatkan kemungkinan keadaan awal yang aman. Sayangnya, beberapa kelemahan,[19] baik dalam perangkat keras maupun perangkat lunak, yang menerapkan mekanisme peluncuran yang terlambat, telah ditemukan oleh para peneliti, sehingga menimbulkan pertanyaan mengenai kemampuan untuk mencapai tingkat kepercayaan yang tinggi dalam lingkungan boot yang rumit. Selain itu, meskipun Trusted Computing Group (TCG) memiliki antarmuka TPM yang terstandarisasi,[20] implementasinya belum melampaui lingkungan komputasi berbasis Arsitektur Intel.

Kabar baik untuk boot aman adalah bahwa sebagian besar sistem komputasi tertanam dan seluler mengandalkan pemuat boot sederhana yang cocok untuk pendekatan akar kepercayaan statis yang dapat diimplementasikan tanpa perangkat keras khusus.

Bahkan sistem berbasis PC pun dapat menggunakan BIOS khusus dan aman yang dikembangkan oleh pakar keamanan perangkat lunak tertanam, jika keamanan sistem tertanam sepadan dengan investasi tersebut. Pembaca harus berkonsultasi dengan vendor perangkat lunak sistem untuk memahami pilihan yang tersedia.

---

[18]Rutkowska J. Subverting Vista Kernel for Fun and Profit. Las Vegas, NV: Black Hat Briefings; August 3, 2006.

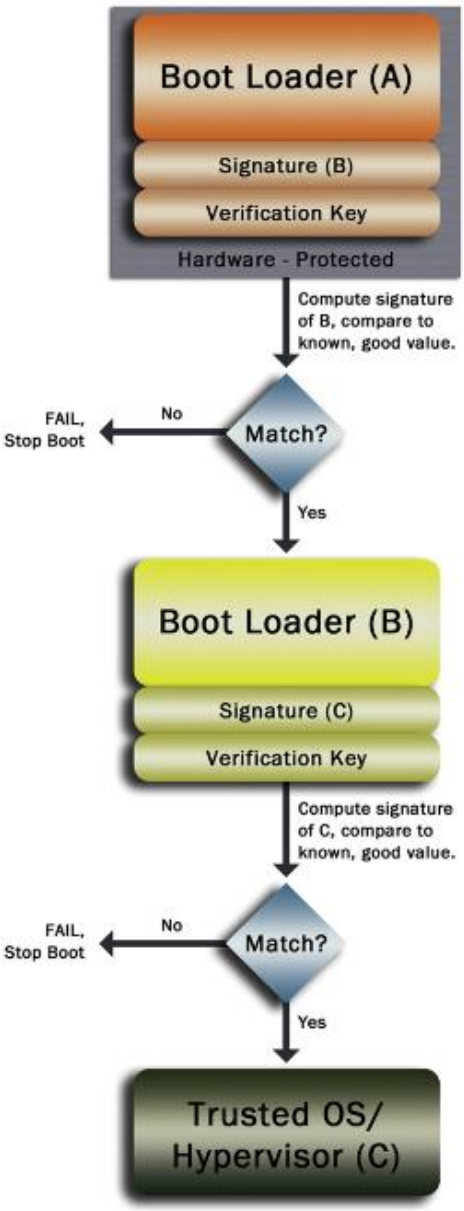
[19] Intel Trusted Execution Technology (Intel TXT). Software Development Guide/Measured Launched Environment Developer's Guide; March 2011.

[20]Wojtczuk R, Rutkowska J. Attacking Intel Trusted Execution Technology. Washington, DC: Black Hat DC; February 18e19, 2009.

Contoh bagus lainnya tentang pentingnya boot aman disajikan dalam pembahasan perlindungan data saat tidak digunakan di Bab 5.

2.9.4 Pengesahan Jarak Jauh

Boot aman memberikan keyakinan kepada pengembang sistem tertanam bahwa produk yang diterapkan tahan terhadap serangan firmware tingkat rendah saat boot. Namun demikian, risiko tetap ada di mana penyerang canggih dapat membahayakan proses boot aman. Selain itu, penyerang mungkin dapat mengganti produk yang dikerahkan secara grosir dengan peniruan identitas yang berbahaya. Misalnya, meteran pintar dapat diambil dari tiang telepon dan diganti dengan meteran pintar jahat yang terlihat sama tetapi secara diam-diam mengirimkan informasi penghitungan energi pribadi ke situs web jahat. Oleh karena itu, bahkan dengan boot aman, pengguna dan administrator mungkin memerlukan jaminan bahwa produk yang diterapkan secara aktif menjalankan TCB yang dikenal baik.



Gambar 2.31 Contoh urutan boot aman akar kepercayaan statis.

Ketika sistem tertanam terhubung ke jaringan manajemen, pengesahan jarak jauh dapat digunakan untuk menyediakan fungsi keamanan penting ini. Sekali lagi, TCG telah melakukan standarisasi mekanisme sistem yang sesuai dengan TCG untuk melakukan pengesahan jarak jauh menggunakan pengukuran berbasis TPM. Akses jaringan dapat ditolak ketika klien penghubung gagal memberikan pengesahan yang tepat. Dalam TCG, fungsi ini disebut Trusted Network Connect (TNC).[21] Namun, pendekatan sederhana dan tidak bergantung pada perangkat keras dapat digunakan untuk sistem tertanam apa pun.

Mari kita asumsikan bahwa sistem tertanam dapat berkomunikasi ke server pengesahan jarak jauh menggunakan saluran aman, seperti IKE/IPsec atau SSL (keduanya dibahas di Bab 5). Pembentukan sesi awal menggunakan kriptografi kunci publik. Secara khusus, kunci pribadi statis yang mewakili identitas sistem tertanam jarak jauh digunakan untuk menandatangani data yang kemudian diautentikasi oleh pengesahan. Selama kunci pribadi ini dan sisi klien dari perangkat lunak protokol koneksi aman disertakan dalam TCB yang divalidasi selama boot aman, pengesahan memiliki jaminan bahwa sistem tertanam menjalankan beberapa firmware yang dikenal baik. Oleh karena itu, tindakan pembuatan sesi IKE atau SSL yang berhasil dapat digunakan untuk pengesahan jarak jauh. Peningkatan pada pendekatan ini, yang memberikan jaminan bahwa sistem tertanam menjalankan serangkaian komponen firmware tepercaya tertentu, adalah meminta klien mengirimkan rangkaian lengkap tanda tangan digital yang sesuai dengan rantai TCB ke pengesahan yang menyimpan kumpulan tanda tangan yang diketahui baik. secara lokal. Hal ini lebih sulit diterapkan karena tanda tangan harus dihitung dan disimpan pada waktu produksi, sebelum produk yang disematkan diterapkan ke lapangan.

[21] Trusted Computing Group (TCP) Trusted Platform Module (TPM) Main Specification. Level 2, Version 1.2, Revision 116 (in three parts, “Design Principles,” “Structures of the TPM,” “Commands”); March 1, 2011.

BAB 3
PENGEMBANGAN PERANGKAT LUNAK TERTANAM YANG AMAN

3.1 PENGANTAR PHASEDPRINCIPLES REKAYASA PERANGKAT LUNAK

Pada Bab 1, kita membahas tren peningkatan kompleksitas perangkat lunak dan dampak buruknya terhadap keamanan dan keselamatan. Satu contoh lagi, sistem kontrol lalu lintas terpusat di Swiss National Railways baru-baru ini dilaporkan berisi empat juta baris kode.[1] Kompleksitas membebani teknik keandalan tradisional, seperti peninjauan kode, dan menyiratkan meningkatnya kebutuhan akan pendekatan komprehensif terhadap jaminan perangkat lunak.

> *Jaminan perangkat lunak mengacu pada tingkat keyakinan yang dimiliki oleh pengguna akhir perangkat lunak dan pemangku kepentingan terkait lainnya (misalnya pemberi sertifikasi) bahwa kebijakan dan fungsi keamanan yang diklaim oleh perangkat lunak tersebut benar-benar terpenuhi.*

Memenuhi persyaratan fungsional saja tidak akan mencapai jaminan yang diperlukan untuk sistem tertanam yang kritis terhadap keamanan. Tujuan bab ini adalah untuk mendiskusikan teknik yang telah terbukti meningkatkan jaminan perangkat lunak tertanam. Peningkatan jaminan mengurangi kerentanan, meningkatkan keamanan, keselamatan, dan keandalan.

Kami memulai diskusi tentang pengembangan perangkat lunak tertanam yang aman dengan pengenalan PHASE, sebuah metodologi yang dapat memutus lingkaran setan perangkat lunak yang tidak dapat diandalkan.

PHASEdPrinciples of High Assurance Software Engineeringd menetapkan serangkaian lima prinsip yang akan digunakan dalam pembuatan perangkat lunak dan sistem yang sangat andal:

1. Minimnya implementasi
2. Arsitektur komponen
3. Hak istimewa paling sedikit
4. Mengamankan proses pembangunan
5. Validasi ahli independen

Setiap prinsip dijelaskan, dengan contoh, di bagian berikut.

3.2 IMPLEMENTASI MINIMAL

Kebanyakan pengembang perangkat lunak tidak bekerja di lingkungan yang mengharuskan menghasilkan solusi seminimal mungkin terhadap suatu masalah. Kode spageti adalah sumber kerentanan yang merajalela dalam perangkat lunak dan memberikan jalan eksploitasi bagi peretas.

Jauh lebih sulit untuk menciptakan solusi yang sederhana dan elegan terhadap suatu masalah dibandingkan dengan solusi yang rumit dan berbelit-belit. Sebagai contoh, mari kita pertimbangkan server web yang mendukung HTML 1.1. Insinyur di Green Hills Software mengembangkan server web dengan jaminan tinggi (HAWS) yang menggunakan pemrosesan

[1] ILTISdRailway Traffic Control. Siemens Switzerland Transportation Systems. http://www.aonix.com/pdf/CaseStudy_ILTIS.pdf.

protokol berbasis negara alih-alih penguraian dan manipulasi string yang rawan kesalahan. Hasilnya: beberapa ratus baris kode sempurna, bukan puluhan ribu baris yang ditemukan di banyak server web komersial. Server web Green Hills berjalan pada sistem operasi INTEGRITAS dengan jaminan tinggi. Pada tahun 2008, sebuah situs web yang berjalan pada platform ini disebarkan di Internet, dan Netragard, organisasi peretas topi putih terkemuka, diundang untuk melakukan penilaian kerentanan terhadap situs web tersebut. CTO Netragard Adriel Desautels melaporkan bahwa situs web tersebut "tidak memiliki permukaan serangan apa pun." HAWS dibahas lebih mendalam sebagai studi kasus di akhir bab ini.

Sebagai contoh lain, mari kita pertimbangkan sistem file. Insinyur di Green Hills Software mengembangkan sistem file penjurnalan dengan jaminan tinggi, yang disebut PJFS, menggunakan beberapa ribu baris kode yang dibuat dengan cermat. Sistem file mencapai kinerja yang sangat baik, memberikan jaminan kuota penyimpanan media untuk klien (penting dalam konteks keselamatan kritis), dan menggunakan penjurnalan transaksional untuk menjamin integritas data dan metadata sistem file (dan waktu reboot instan) jika terjadi kehilangan daya secara tiba-tiba. Sebaliknya, sistem file penjurnalan komersial biasanya melebihi 100.000 baris sumber, dengan banyak kelemahan perangkat lunak.

3.3 ARSITEKTUR KOMPONEN

Sangat penting untuk menggunakan antarmuka antar komponen yang terdefinisi dengan baik dan terdokumentasi. Antarmuka ini berfungsi sebagai kontrak antara pemilik komponen dan harus dibuat dengan hati-hati untuk meminimalkan churn yang memaksa serangkaian perubahan implementasi, pengujian, dan integrasi. Jika modifikasi pada antarmuka diperlukan, pemilik komponen yang komponennya menggunakan antarmuka ini harus menyetujuinya, dengan melibatkan manajemen bersama untuk menyelesaikan perselisihan jika diperlukan.

Prinsip ketahanan perangkat lunak yang penting adalah menyusun sistem perangkat lunak besar dari komponen-komponen kecil, yang masing-masing mudah dipelihara, idealnya, oleh seorang insinyur yang memahami setiap baris kode.

Akibat wajar yang penting dari prinsip arsitektur komponen adalah bahwa fungsi keselamatan dan/atau penegakan keamanan harus ditempatkan ke dalam komponen terpisah sehingga operasi penting dilindungi dari gangguan oleh bagian sistem yang tidak penting. Namun, tidak cukup hanya mengisolasi fungsi keamanan ke dalam komponennya saja. Setiap komponen keamanan penting harus, semaksimal mungkin, dirancang atau difaktorkan ulang untuk menghilangkan fungsionalitas apa pun yang bukan bagian dari fungsi penegakan keamanannya.

Salah satu alasan utama mengapa perangkat lunak yang terlalu rumit sulit untuk dikelola adalah karena perangkat lunak tersebut hampir selalu dikerjakan oleh banyak pengembang, sering kali pada waktu yang berbeda selama masa pakai produk. Karena perangkat lunak ini terlalu rumit untuk dipahami oleh satu orang, fitur dan penyelesaian cacat sama-sama ditangani dengan dugaan dan tambal sulam. Cacat sering kali tidak diperbaiki, dan cacat baru ditambahkan saat pengembang berupaya memperbaiki masalah lainnya.

Komponenisasi juga memberikan kemampuan bagi perancang sistem untuk membuat perubahan spesifik pelanggan dengan cara yang metodis. Dengan berfokus pada kebutuhan pelanggan dan pasar, perancang dapat membuat perubahan dengan menukar sebagian kecil komponen dibandingkan dengan sebagian besar dasar perangkat lunak. Hal ini meminimalkan tugas pengujian regresi dengan mengurangi dampaknya terhadap sistem secara keseluruhan. Jika desainer selalu mempertimbangkan sikap komponenisasi dan definisi antarmuka ini, perbaikan dapat dilakukan seiring waktu dengan risiko rendah.

Komponenisasi memberikan banyak manfaat, termasuk peningkatan kemampuan pengujian, kemampuan audit, isolasi data, dan pembatasan kerusakan.

Komponenisasi dapat mencegah kegagalan pada satu komponen berubah menjadi kegagalan sistem. Komponenisasi juga dapat secara drastis mengurangi biaya pengembangan dan biaya sertifikasi, jika memungkinkan, dengan memungkinkan pengembang untuk menerapkan proses pengembangan yang lebih ketat pada komponen-komponen yang tidak penting sekaligus meningkatkan tingkat jaminan untuk bagian-bagian penting, yang seringkali merupakan persentase yang relatif kecil dari keseluruhan sistem.

Membagi sistem menjadi komponen-komponen memerlukan antarmuka yang terdefinisi dengan baik. Daripada memodifikasi potongan kode yang sama, pengembang harus mendefinisikan antarmuka yang sederhana dan jelas untuk komponen dan hanya menggunakan antarmuka komponen yang terdokumentasi dengan baik (atau setidaknya dipahami dengan baik) untuk berkomunikasi dengan komponen lain. Komponenisasi memungkinkan pengembang untuk bekerja lebih mandiri sehingga lebih efisien, meminimalkan waktu yang dihabiskan dalam pertemuan di mana pengembang berusaha menjelaskan perilaku perangkat lunak mereka. Memfaktorkan ulang proyek perangkat lunak besar dengan cara ini dapat memakan waktu. Namun, jika hal ini tercapai, semua pembangunan di masa depan akan lebih mudah dikelola.

Pastikan tidak ada satu pun partisi perangkat lunak yang lebih besar dari yang dapat dipahami sepenuhnya oleh satu pengembang. Setiap partisi harus memiliki pengelola partisi yang terkenal. Salah satu cara untuk memastikan bahwa pengembang memahami siapa yang memiliki partisi mana adalah dengan memelihara daftar manajer partisi yang mudah diakses dan hanya dimodifikasi oleh personel manajemen yang tepat. Manajer partisi adalah satu-satunya orang yang berwenang untuk melakukan modifikasi pada partisi atau memberikan hak kepada pengembang lain untuk melakukan modifikasi. Dengan memiliki kepemilikan yang jelas atas setiap baris kode dalam proyek, pengembang tidak tergoda untuk mengedit kode yang tidak memenuhi syarat untuk ditangani. Pastikan semua pengembang mengetahui siapa pengelola komponennya.

Manajer komponen mengembangkan, seiring waktu, pemahaman komprehensif tentang partisi yang mereka miliki, memastikan bahwa modifikasi di masa depan dilakukan dengan pengetahuan lengkap tentang konsekuensi dari memodifikasi perangkat lunak apa pun di dalam partisi tersebut.

3.3.1 Komponenisasi Runtime

Biasanya perwujudan komponen dalam sistem komputer target berupa program tunggal yang dapat dieksekusi. Contoh komponen termasuk aplikasi Windows .EXE dan proses

POSIX/UNIX. Oleh karena itu, perangkat lunak kompleks yang terdiri dari beberapa komponen harus selalu digunakan bersama dengan sistem operasi yang menggunakan perlindungan memori untuk mencegah kerusakan ruang memori satu komponen oleh partisi lain. Komunikasi antar-komponen biasanya dilakukan dengan konstruksi penyampaian pesan sistem operasi standar.

Sistem operasi tertanam (dan mikroprosesor) yang berbeda memiliki kemampuan yang berbeda-beda dalam hal menerapkan pemisahan yang ketat antar komponen. Misalnya, sistem operasi real-time yang kecil mungkin tidak menggunakan unit manajemen memori komputer sama sekali; beberapa aplikasi perangkat lunak tidak dapat dilindungi satu sama lain, dan sistem operasi itu sendiri berisiko mengalami kelemahan dalam kode aplikasi. Sistem operasi model memori datar ini tidak cocok untuk sistem perangkat lunak yang rumit dan terpartisi. Sistem operasi desktop tujuan umum seperti Linux dan Windows menggunakan perlindungan memori dasar, di mana partisi dapat ditetapkan proses yang dilindungi dari korupsi oleh unit manajemen memori, namun tidak memberikan jaminan keras tentang ketersediaan memori atau sumber daya waktu CPU.

Untuk sistem yang aman, sistem operasi tertanam harus menyediakan partisi aplikasi yang ketat baik dalam ruang maupun waktu. Aplikasi yang rusak tidak dapat menghabiskan memori sistem, sumber daya sistem operasi, atau waktu CPU karena perangkat lunak yang salah dibatasi secara ketat pada kuota sumber daya penting yang ditetapkan. Kuota benar-benar memengaruhi semua memori yang digunakan, termasuk memori heap untuk runtime C/C++, memori yang digunakan untuk blok kontrol proses dan objek sistem operasi lainnya, serta memori tumpukan runtime proses. Selain itu, kebijakan partisi memberikan kuota waktu eksekusi yang ketat dan kontrol ketat atas akses ke sumber daya sistem seperti perangkat I/O dan file. Partisi aplikasi yang lebih ketat pada tingkat sistem operasi memastikan bahwa manfaat kebijakan manajemen partisi yang digunakan dalam proses pengembangan dapat direalisasikan selama runtime. Jika memungkinkan, gunakan sistem operasi yang menggunakan partisi aplikasi yang sebenarnya.

3.3.2 Catatan tentang Proses versus Thread

Ketika pengembang memfaktorkan perangkat lunak yang tertanam ke dalam komponen, perwujudan alami dari komponen runtime adalah thread. Thread adalah aliran eksekusi yang berbagi satu ruang alamat dengan thread lainnya. Di sebagian besar sistem operasi modern, ruang alamat memiliki setidaknya satu thread default, dan ruang alamat dengan thread tunggal ini sering disebut proses. Karena mudah dibuat dan cara sebagian besar pemrogram tertanam belajar menggunakan konkurensi, thread sering kali digunakan secara berlebihan. Selain itu, pengembang sistem tertanam sering mempunyai kesan yang salah bahwa proliferasi proses akan menghabiskan terlalu banyak sumber daya sistem dibandingkan dengan thread. Meskipun bobot thread tentu saja lebih ringan dibandingkan proses full-blown, perbedaan tersebut menjadi semakin tidak penting dalam sistem tertanam modern. Alasan lain penggunaan thread yang berlebihan dapat dikaitkan dengan fakta bahwa sistem operasi real-time asli yang dibuat pada tahun 1980an dan awal 1990an tidak mendukung proses yang dilindungi memori sama sekali. Pengembang menjadi terbiasa dengan thread, dan warisan mereka tetap hidup.

Berlawanan dengan kepercayaan umum, desainer harus mengupayakan rasio satu banding satu antara thread dan proses. Dengan kata lain, setiap komponen yang dilindungi memori harus berisi jumlah thread minimum. Alasan utamanya adalah proses multi-thread sering kali menjadi penyebab masalah sinkronisasi halus yang mengakibatkan kerusakan memori, kebuntuan, dan kesalahan lainnya. Penggunaan proses memori virtual memaksa pengembang untuk membuat antarmuka komunikasi antar-proses yang terdefinisi dengan baik antar komponen. Setiap komponen dapat diuji unitnya secara independen dengan menggunakan antarmuka ini. Filosofi komponen tanpa thread ini menghindari beberapa kerentanan paling buruk yang mengganggu perangkat lunak tertanam.

3.4 HAK ISTIMEWA TERKECIL

Komponen harus diberi akses hanya pada sumber daya tersebut (misalnya jalur komunikasi, perangkat I/O, layanan sistem, informasi) yang benar-benar diperlukan.

Kontrol akses harus bersifat wajib untuk sumber daya dan informasi sistem yang penting. Seperti yang dibahas di Bab 2, sistem yang tidak aman biasanya mengizinkan program apa pun mengakses sistem file, meluncurkan program lain, dan memanipulasi perangkat sistem.

Misalnya, kerentanan buffer overflow browser memungkinkan penyerang mengakses file apa pun karena browser web memiliki hak istimewa untuk mengakses seluruh sistem file. Tidak ada alasan mengapa browser web harus memiliki akses tak terkekang ke seluruh sistem file. Peramban web harus memiliki akses tulis hanya ke direktori khusus peramban (di mana pengguna dapat dengan hati-hati memutuskan apa yang dapat keluar dari kotak pasir), atau semua permintaan penulisan peramban memerlukan persetujuan pengguna melalui kotak dialog. Akses baca dapat dibatasi pada daftar putih file yang diketahui diperlukan untuk pengoperasian browser. Tumpukan runtime browser web tidak boleh dieksekusi.

Setiap komponen dari keseluruhan sistem harus dirancang dengan mempertimbangkan hak istimewa yang paling kecil, dan yang terbaik adalah memulai tanpa hak istimewa dan bekerja sesuai kebutuhan daripada memulai dengan wastafel dapur dan mengurangi hak istimewa. Sebagai contoh lain, mari kita pertimbangkan fungsi umum sistem operasi dalam meluncurkan proses baru. Salah satu metode UNIX asli, fork(), membuat duplikat dari proses induk, memberikan semua hak istimewa yang sama kepada anak (misalnya, akses ke deskriptor file, memori) seperti induknya. Pengembang kemudian harus menutup deskriptor dan mencoba membatasi kemampuan anak tersebut. Hal ini memerlukan pengetahuan masa depan yang tidak realistis tentang semua sumber daya sistem yang dapat diakses oleh suatu proses. Oleh karena itu, kesalahan dalam penggunaan antarmuka ini sering kali menyebabkan kerentanan yang serius.

Dalam sistem operasi yang aman, mekanisme pembuatan proses default menetapkan anak tanpa akses ke kemampuan induk apa pun untuk memori, perangkat, atau sumber daya lainnya. Pencipta dapat secara sistematis memberikan kemampuan kepada anak, membangun proses hak istimewa yang sangat terbatas. Anak tersebut juga harus

mendapatkan sumber daya memori fisiknya dari induknya, memastikan bahwa suatu proses tidak dapat menguras sistem atau mempengaruhi proses penting lainnya dengan fork bomb.

3.5 PROSES PEMBANGUNAN YANG AMAN

Untuk komponen penting yang mendukung keselamatan dan keamanan, proses pengembangan perangkat lunak harus memenuhi tingkat jaminan yang jauh lebih tinggi daripada yang digunakan untuk komponen tujuan umum.

Pengembang sistem tertanam yang tidak terbiasa dengan proses pengembangan yang aman harus mempelajari standar pengembangan dengan jaminan tinggi yang telah terbukti digunakan untuk mensertifikasi sistem tertanam yang penting. Dua standar penting adalah DO-178B Level A (standar untuk memastikan keselamatan sistem penting penerbangan di pesawat komersial) dan ISO/IEC 15408 (Kriteria Umum) EAL6/7 atau setara. Proses pengembangan dengan jaminan tinggi akan mencakup banyak kontrol, termasuk manajemen konfigurasi, standar pengkodean, pengujian, desain formal, bukti formal fungsionalitas penting, dan sebagainya.

Mari kita pertimbangkan kasus di mana seorang pemrogram nakal mempunyai kemampuan untuk menginstal ke dalam perangkat lunak mesin pesawat suatu kondisi sehingga mesin akan mati pada waktu dan tanggal tertentu. Perangkat lunak ini mungkin berada di semua mesin untuk kelas pesawat tertentu. Salah satu aspek dari proses pengembangan yang aman adalah memiliki tim sistem dan perangkat lunak terpisah yang mengembangkan sistem kendali mesin pesawat yang berlebihan. Dengan cara ini, kesalahan sistemik atau tidak disengaja yang dibuat oleh satu tim dapat dikurangi melalui jalur pengembangan yang sepenuhnya berbeda. Kemandirian sistem, perangkat lunak, dan tim pengujian sesuai dengan standar juga berkontribusi terhadap proses pengembangan yang aman ini.

Mengkaji perbedaan utama antara proses pembangunan dengan jaminan tinggi dan proses pembangunan dengan jaminan rendah dapat menjadi sebuah pembelajaran. Bab ini mencakup perbandingan kelas standar keamanan dalam Common Criteria, standar internasional untuk mengevaluasi keamanan sistem TI.

3.5.1 Manajemen Perubahan

Sebuah proyek perangkat lunak mungkin kuat dan dapat diandalkan pada saat peluncuran pertama, hanya untuk bertahan dari perubahan yang membusuk pada tahun-tahun berikutnya karena fitur-fitur baru, bukan bagian dari desain asli, diretas, menyebabkan kode menjadi sulit untuk dipahami, dipelihara, dan dipelihara. dan tes. Tuntutan waktu ke pasar memperburuk masalah, mempengaruhi pengembang untuk melakukan perubahan secara tergesa-gesa sehingga merugikan keandalan.

Aspek yang sangat penting dalam memelihara perangkat lunak yang aman dalam jangka panjang adalah dengan memanfaatkan sistem manajemen perubahan yang efektif.

Beberapa dasar manajemen perubahan kualitas, seperti penggunaan sistem manajemen konfigurasi untuk mengontrol dan mencatat perubahan serta mengelola cabang dan rilis kode, diasumsikan dan tidak dibahas dalam buku ini.

3.5.2 Tinjauan Sejawat

Aspek penting dari manajemen perubahan yang efektif adalah penggunaan tinjauan kode sejawat. Urutan tinjauan kode sejawat yang umum terdiri dari penulis kode yang mengembangkan presentasi yang menjelaskan perubahan kode diikuti dengan pertemuan tatap muka dengan satu atau lebih pengembang dan manajer pengembangan yang terlibat dalam proyek. Pengembang menyajikan desain perangkat lunak yang dimaksud, dan yang lain mencoba menyodok kodenya. Pertemuan-pertemuan ini bisa sangat menyakitkan dan memakan waktu. Audiens terkadang merasa terdorong untuk memilih setiap baris kode untuk menunjukkan kehebatan mereka.

Gunakan tinjauan kode asinkron dengan korespondensi email atau pertemuan langsung yang dikontrol dengan cermat.

Komponenisasi secara drastis mengurangi waktu yang diperlukan untuk peninjauan kode karena pakar kode hampir selalu memodifikasi komponen mereka sendiri. Perdebatan mengenai keputusan desain biasanya dihindari. Selain itu, kami menganjurkan tinjauan sejawat tatap muka terbatas serta tinjauan korespondensi jika memungkinkan. Manajer partisi menerapkan perubahan pada salinan lokal perangkat lunak, memilih rekan yang sesuai untuk meninjau perubahan, dan kemudian membuat permintaan peninjauan melalui email. Segera setelah itu, tetapi pada waktu yang tepat bagi pengulas, pengulas meninjau perbedaan kode di mejanya dan kemudian mengirimkan kembali komentar melalui email ke penulis. Ketika penulis menerima indikasi email bahwa perubahan disetujui, perangkat lunak berkomitmen pada sistem manajemen konfigurasi. Jika pengulas menolak modifikasi, penulis harus memperbaiki setiap kekurangan yang ditemukan atau, jika dia tidak setuju dengan penilaian tersebut, mengajukan permohonan kepada manajer umum untuk menjadi wasit.

Sistem manajemen konfigurasi harus menyediakan kemampuan untuk menentukan identifikasi pengguna pengulas sebagai bagian dari komentar penerapan. Misalnya, CVS mengizinkan skrip dijalankan selama penerapan; skrip diberikan komentar komit yang diurai untuk identifikasi pengguna. Jika identifikasi pengguna yang valid tidak ditemukan, sistem manajemen konfigurasi akan menolak penerapannya. Dengan demikian, sistem manajemen konfigurasi dapat digunakan untuk mengotomatiskan penerapan kebijakan peninjauan kode. Tanpa sistem otomatis seperti itu, tidak ada jaminan bahwa pengembang tidak akan melakukan perubahan yang belum diperiksa dengan benar.

Gunakan sistem manajemen konfigurasi untuk mengotomatiskan penerapan tinjauan sejawat untuk setiap modifikasi pada kode penting. Pencatatan identifikasi peninjau dalam sistem manajemen konfigurasi juga menyediakan jejak kertas elektronik untuk auditor sertifikasi keamanan.

Keuntungan lain dari partisi adalah kemampuan untuk meminimalkan kebutuhan proses di seluruh sistem. Dalam setiap proyek perangkat lunak besar, terdapat kontinum kekritisan di antara berbagai bagian kode. Sebagai contoh, mari kita perhatikan sistem laser excimer yang digunakan dalam produksi semikonduktor. Laser itu sendiri dikendalikan oleh aplikasi perangkat lunak real-time yang sangat penting. Jika aplikasi ini salah, laser pada

gilirannya mungkin gagal, menghancurkan semikonduktor. Selain itu, sistem berisi aplikasi komunikasi yang menggunakan CORBA melalui TCP/IP untuk menerima perintah dan mengirim data diagnostik melalui jaringan.

Jika aplikasi komunikasi gagal, maka sistem mungkin menjadi tidak tersedia atau data diagnostik mungkin hilang, namun tidak ada kemungkinan laser mengalami kegagalan fungsi. Jika kedua aplikasi dibangun menjadi satu sistem monolitik di mana semua kode dijalankan dalam ruang memori yang sama, maka seluruh konten perangkat lunak harus dikembangkan pada tingkat kualitas dan keandalan tertinggi. Namun, jika aplikasi dipartisi, pengembangan aplikasi komunikasi non-kritis dapat dilakukan dengan tingkat ketelitian yang lebih rendah, sehingga menghemat waktu pemasaran dan biaya pengembangan. Terapkan tingkat ketelitian proses, termasuk tinjauan kode dan kontrol lainnya, yang sepadan dengan tingkat kekritisan komponen.

Jelasnya, kami tidak menganjurkan kebebasan untuk semua komponen yang tidak dianggap penting; manajemen harus menggunakan penilaian mengenai pengendalian mana yang diterapkan pada berbagai tim perangkat lunak. Ketika pengendalian proses pada aplikasi non-kritis dikurangi, waktu pemasaran sistem secara keseluruhan dapat ditingkatkan tanpa mengorbankan keandalan yang penting.

Tinjauan Sejawat Berorientasi Keamanan

Sebagian besar tinjauan sejawat dihabiskan untuk mencari bug pengkodean, kelemahan desain, dan pelanggaran standar pengkodean. Meskipun aktivitas ini berkontribusi pada perangkat lunak yang lebih andal dan aman, sebagian besar organisasi perangkat lunak tertanam tidak melakukan tinjauan berdasarkan analisis keamanan secara khusus.

Saat pengembang mempresentasikan desain atau perangkat lunak baru, peninjau harus mempertimbangkan karakteristik yang relevan dengan keamanan. Misalnya:

- **Hak istimewa yang paling kecil**: Dapatkah perangkat lunak difaktorkan ulang sedemikian rupa sehingga komponen yang paling tidak penting diberikan hak istimewa yang paling sedikit dalam hal akses terhadap sumber daya? Mengurangi hak istimewa suatu komponen akan mengurangi permukaan serangannya dan mengurangi persyaratan jaminannya, sehingga meningkatkan efisiensi dalam pengembangan dan sertifikasi (jika berlaku).
- **Potensi serangan**: Bayangkan dalam konteks penyerang yang tinggal di sistem (malware) atau eksternal (yang ditularkan melalui jaringan): di manakah titik akses dan kelemahan dalam sistem, dan bagaimana penyerang dapat berupaya untuk mengkompromikannya? Seperti dalam poker, kesuksesan membutuhkan penempatan diri dalam kerangka acuan lawan dan pelatihan untuk berpikir seperti lawan. Seiring waktu, pengembang dengan pola pikir ini menjadi mahir dalam memprediksi potensi serangan dan oleh karena itu dapat menerapkan kontrol untuk mencegah kegagalan keamanan.
- **Serangan canggih**: Sekalipun kode yang ditinjau tidak melindungi jaringan listrik, mari kita pertimbangkan masalah keamanan tingkat lanjut seperti saluran samping dan rahasia, keamanan transmisi, dan kerusakan DMA melalui periferal sistem. Pengembang yang dilatih untuk mempertimbangkan ancaman serangan yang canggih

akan lebih siap menangani komponen yang memerlukan ketahanan tinggi terhadap ancaman tersebut.

Dengan menjadikan keamanan sebagai bagian dari tinjauan sejawat, manajemen akan menciptakan budaya fokus keamanan di seluruh tim pengembangan.

Faktanya, karena tinjauan sejawat mencakup sebagian besar interaksi kelompok dalam organisasi pembangunan, tinjauan sejawat merupakan tempat yang ideal untuk menumbuhkan kewaspadaan yang diperlukan untuk membangun sistem tertanam yang aman.

3.5.3 Keamanan Alat Pengembangan

Telur Paskah adalah pesan, lelucon, atau kemampuan yang sengaja tidak didokumentasikan yang dimasukkan ke dalam program oleh pengembang program, sebagai tantangan tambahan bagi pengguna atau sekadar untuk bersenang-senang. Telur paskah umumnya ditemukan di video game. Alat pengemasan Linux apt-get memiliki telur sapi ini:

```
> apt-get moo
         (__)
         (oo)
   /------\/
  / |    ||
 * /\---/\
   ~~   ~~
....“Have you mooed today?”...
```

Imut-imut. Lucu. Namun bagaimana jika pengembang bermaksud memasukkan sesuatu yang berbahaya? Bagaimana sebuah organisasi dapat dilindungi dari ancaman orang dalam ini? Bagaimana organisasi dapat memastikan bahwa malware tidak disisipkan oleh middleware pihak ketiga atau kompiler yang digunakan untuk membuat perangkat lunak?

Pengembang dan pengguna memerlukan kepastian asal bit: keyakinan bahwa setiap bit biner perangkat lunak produksi berasal dari versi kode sumber yang dikenal baik.

Ini adalah aspek penting dari keamanan perangkat lunak yang tidak pernah dipertimbangkan oleh banyak pengembang sistem tertanam. Standar keamanan dan keselamatan dengan jaminan tinggi, seperti DO-178B Level A (keselamatan pesawat) dan *Common Criteria Evaluated Assurance* Level 7 (keamanan TI), memerlukan kemampuan untuk membuat ulang bagian yang tepat dari perangkat lunak produksi akhir dari manajemen konfigurasi sistem. Item tambahan, bukan hanya kode sumber produk, harus dikelola konfigurasinya. Misalnya, file pendukung apa pun (misalnya skrip) yang digunakan untuk pembuatan gambar produksi harus dikontrol dengan ketat. Dan rantai alat yang digunakan untuk membangun perangkat lunak juga harus tercakup. Kegagalan untuk mengontrol secara ketat seluruh sistem pengembangan dapat menyebabkan kerentanan serius, baik yang tidak disengaja maupun berbahaya.

Studi Kasus: Peretasan Thompson

Salah satu subversi tersebut dilakukan oleh Ken Thompson dan dilaporkan dengan terkenal dalam pidato penerimaan Turing Award-nya. Thompson memasukkan pintu belakang ke UNIX dengan menumbangkan kompiler yang digunakan untuk membangun UNIX.

Modifikasi Thompson menyebabkan verifikasi kata sandi login UNIX cocok dengan string yang dipilih Thompson selain validasi database normal. Intinya, Thompson mengubah program login UNIX yang dulu terlihat seperti ini:

```
int login(unsigned int uid, char *password)
{
    if (strcmp(pass_dbase(uid), password) == 0)
        return true: // password match. login ok
    else
        return false; // password mismatch, login fail
}
```

menjadi sesuatu yang terlihat seperti ini:

```
int login(unsigned int uid, char *password)
{
  if (strcmp(pass_dbase(uid). password) == 0 ||
      strcmp("ken_thompson", password) == 0))
    return true; // password match, login ok
  else
    return false; // password mismatch, login fail
}
```

Namun, mengubah kode sumber UNIX akan terlalu mudah dideteksi, jadi Thompson memodifikasi kompiler untuk memasukkan pintu belakang. Dengan penyisipan kompiler, pemeriksaan kode sumber UNIX tidak akan cukup untuk mendeteksi bug. Trojan kompiler akan menjadi fragmen kode yang memeriksa pohon sintaksis internal dari program yang dikompilasi, mencari urutan kode pemeriksaan kata sandi login tertentu dan menggantinya dengan pintu belakang:

```
if (!strcmp(function_name(). "login")) {
    if (OBJ_TYPE(obj) == IF_STATEMENT &&
        OBJ_TYPE(obj->left) = FUNCTION &&
        !strcmp(OBJ_NAME(obj->left), "strcmp")) {
    Object func=GET_ARG(1, obj->left);

    if (OBJ_TYPE(func) == FUNCTION) &&
        !strcmp(OBJ_NAME(func),"pass_dbase")) {
    // insert back door
    obj = MAKEOBJ(ORCMP, obj,
        MAKEOBJ(FUNCTION, "strcmp",
        MAKEOBJ(STRING, "ken_thompson").
        GET_ARG(2. obj->left);
        }
    }
}
```

Jika kompiler dikelola konfigurasinya dan/atau ditinjau oleh rekan sejawat, perubahan Thompson mungkin terdeteksi melalui inspeksi. Namun jika kode sumber kompiler tidak berada dalam manajemen konfigurasi atau sangat rumit, Trojan dapat luput dari perhatian selama beberapa waktu. Juga, siapa yang berpikir untuk mempertanyakan kode yang dilakukan oleh Ken Thompson yang terhormat? Salah satu pelajaran yang dapat diambil adalah bahwa mereka yang memiliki kepercayaan paling besar dapat menyebabkan kerugian paling besar. Argumen lain untuk menerapkan mentalitas yang paling tidak mendapat hak istimewa di seluruh departemen teknik.

Dengan asumsi bahwa Trojan dalam kompiler mungkin terdeteksi, Thompson mengambil langkah lebih jauh dan mengajari kompiler untuk menambahkan Trojan ke dalam dirinya sendiri (dua tingkat tipuan). Dengan kata lain, Trojan kompiler sebelumnya tidak dimasukkan ke dalam kode sumber kompiler, melainkan ke dalam kode objek kompiler saat mengkompilasi sendiri. Trojan tersebut sekarang dikatakan dapat mereproduksi dirinya sendiri. Meskipun hal ini mungkin terdengar canggih, sebenarnya tidak sulit jika pengembang sudah memiliki pemahaman dasar tentang cara kerja kompiler (yang tentu saja dilakukan oleh Ken Thompson): cukup temukan fase kompiler yang sesuai dan masukkan fragmen kode sebelumnya ke dalam pohon sintaksis target. ketika targetnya adalah kompiler.

Ada cara untuk mendeteksi serangan lebih lanjut ini. Inspeksi kode sekali lagi merupakan metode yang jelas. Pendekatan lain adalah dengan membangun kompiler dari sumber (di bawah manajemen konfigurasi), membangun dari sumber yang sama lagi dengan kompiler baru ini, dan mengharuskan dua biner kompiler yang dibangun identik bit demi bit. Metode perbandingan alat biner ini, yang disebut bootstrapping, adalah metode yang murah dan efektif untuk mendeteksi beberapa kelas kerentanan alat pengembangan. Dengan Trojan Thompson yang baru saja dimasukkan ke dalam kode sumber kompiler, biner pertama tidak akan berisi kode Trojan, tetapi biner kedua akan berisi kode Trojan, menyebabkan pengujian bootstrap gagal. Tentu saja, pendekatan ini hanya berfungsi jika kompiler dan kompiler dari kompiler memiliki back end prosesor target yang sama, yaitu kompiler adalah hosting mandiri. Karena sebagian besar sistem UNIX memiliki kompiler yang dihosting sendiri, pengujian generasi ini efektif.

Namun, untuk menutupi jejaknya lebih jauh, Thompson menghapus kode sumber Trojan kompiler, hanya menyisakan biner kompiler yang ditumbangkan yang dipasang sebagai kompiler sistem default. Pengujian bootstrapping berikutnya akan gagal mendeteksi subversi karena kompiler generasi pertama dan kedua berisi kode biner Trojan.

Serangan ini menunjukkan betapa canggihnya penyerang dapat menggagalkan keamanan alat pengembangan yang baik sekalipun. Idealnya, kami ingin membuktikan secara formal korespondensi antara kode sumber suatu alat dan kode objek kompilasi yang dihasilkannya. Alternatif praktisnya adalah dengan memerlukan bootstrap setiap kali komponen rantai alat default diganti. Melakukan pengujian bootstrap untuk setiap kompiler baru akan menghasilkan rantai kepercayaan yang akan mencegah subversi Thompson jika proses ini telah dilakukan sebelum serangannya.

Validasi kondisi/cakupan keputusan (MC/DC) yang dimodifikasi dari program login UNIX juga akan mendeteksi peretasan Thompson karena perbandingan dengan rangkaian kata sandi pintu belakang Thompson tidak akan pernah berhasil dalam pengujian normal.

Pengujian MC/DC dibahas lebih lanjut pada bab ini. Namun, strategi pertahanan mendalam yang baik tidak boleh berasumsi bahwa pengujian akan menemukan segala bentuk subversi alat pengembangan.

Tentu saja, sistem manajemen konfigurasi harus dilindungi dari gangguan, baik melalui serangan jaringan jarak jauh atau serangan fisik terhadap komputer yang menampung sistem konfigurasi. Beberapa organisasi mungkin mengharuskan database manajemen konfigurasi aktif disimpan dalam brankas yang aman, hanya dapat diakses oleh personel keamanan yang berwenang. Jika sebuah organisasi tidak memikirkan keamanan pembangunan, sekaranglah waktunya untuk memulai.

3.5.4 Pengodean Aman

Banyak aspek pengembangan produk tertanam yang aman berlaku langsung pada proses pembuatan perangkat lunak. Bagian berikut membahas sejumlah teknik pengkodean aman yang penting. Karena sekitar 80% sistem tertanam menggunakan C atau C++, sebagian besar saran dapat diterapkan langsung pada bahasa pemrograman ini atau bahasa pemrograman serupa. Namun, sebagian besar nasihat tersebut bersifat umum untuk bahasa apa pun. Buku ini tidak mencoba melakukan perbandingan signifikan antara bahasa pemrograman tingkat tinggi. Pertama, sebagian besar organisasi sistem tertanam dihadapkan pada basis kode warisan besar yang ditulis dalam serangkaian bahasa pemrograman tertentu dan staf teknik yang terlatih dalam bahasa tersebut dan perangkat terkaitnya; memilih bahasa baru bukanlah suatu kemewahan. Kedua, kemanjuran bahasa pemrograman lebih dari sekadar masalah yang berkaitan dengan keamanan, dan oleh karena itu perbandingan bahasa pemrograman apa pun yang masuk akal akan jauh melampaui pokok bahasan buku ini. Bagi pembaca yang tertarik dengan perbandingan semacam itu, tersedia banyak sumber online dan cetak yang membahas subjek ini. Tempat yang baik untuk memulai adalah entri Wikipedia, “Perbandingan bahasa pemrograman.”[2]

Standar Pengkodean

Sebagian besar standar dan panduan sertifikasi keselamatan dan kualitas mendukung penggunaan standar pengkodean yang mengatur cara pengembang menulis kode.[3] Beberapa di antaranya bahkan lebih jauh lagi dengan merekomendasikan atau mengharuskan aturan khusus dimasukkan dalam standar pengkodean. Tujuan dari standar pengkodean adalah untuk meningkatkan keandalan dengan menyebarkan praktik pengkodean cerdas. Misalnya, standar pengkodean mungkin berisi aturan yang membantu pengembang menghindari konstruksi bahasa yang berbahaya, membatasi kompleksitas fungsi, dan menggunakan gaya sintaksis dan komentar yang konsisten. Aturan-aturan ini dapat secara drastis mengurangi terjadinya kelemahan, membuat perangkat lunak lebih mudah untuk diuji, dan meningkatkan pemeliharaan jangka panjang.

Mengembangkan dan menerapkan standar pengkodean yang mengatur pengembangan perangkat lunak untuk semua komponen penting. Bukan hal yang aneh jika standar pengkodean berkembang dan meningkat seiring waktu. Misalnya, tim pengembangan mungkin menemukan alat baru yang dapat meningkatkan keandalan kode dan

[2] http://en.wikipedia.org/wiki/Comparison_of_programming_languages.
[3] DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA, Inc.; 1992.

merekomendasikan agar manajemen menambahkan persyaratan agar alat ini digunakan selama proses pengembangan.

Hal ini juga tidak biasa untuk melihat standar pengkodean yang terdiri dari aturan panduan yang penegakannya dilakukan terutama dengan tinjauan kode manusia. Mengembangkan standar pengkodean baru dengan lusinan aturan yang harus diverifikasi secara manual adalah cara yang pasti untuk mengurangi efisiensi pengembang, meskipun hal itu meningkatkan keandalan kode.

Memaksimalkan penggunaan verifikasi otomatis standar pengkodean; meminimalkan penggunaan aturan pengkodean yang diverifikasi secara manual.

Banyak penganalisis kode statis, dan beberapa kompiler, dapat mengotomatiskan sebagian besar standar pengkodean aman yang umum. Selain itu, meskipun beberapa aturan standar pengkodean bersifat khusus untuk suatu bahasa, ada beberapa aturan yang berlaku secara universal atau hampir universal yang harus menjadi bagian dari standar pengkodean berkualitas tinggi. Dengan asumsi mereka meningkatkan kualitas perangkat lunak, aturan standar pengkodean terbaik adalah aturan yang penerapannya dapat diotomatisasi dan dapat diterapkan pada proyek perangkat lunak apa pun. Beberapa contoh diberikan dalam bab ini.

Melarang peringatan kompiler. Kompiler dan komponen rantai alat lainnya (misalnya, linker/loader) sering kali mengeluarkan peringatan, dibandingkan menghentikan pembangunan dengan kesalahan fatal. Peringatan adalah indikator bagi pengembang bahwa suatu konstruksi mungkin secara teknis sah tetapi dipertanyakan, seperti melakukan tendangan sudut dari bahasa yang tidak didefinisikan dengan baik. Konstruksi seperti itu tidak jarang menjadi penyebab bug yang tidak kentara. Untuk memastikan bahwa pengembang tidak mengabaikan peringatan secara sengaja atau tidak sengaja, beri tahu kompiler untuk menganggap semua peringatan sebagai kesalahan. Banyak kompiler mempunyai pilihan seperti itu.

Manfaatkan pengaturan bahasa kompiler yang paling ketat untuk keamanan dan keandalan. Penyusun juga cenderung memberikan berbagai tingkat keketatan dalam hal penafsiran standar bahasa. Selain itu, beberapa kompiler mampu memperingatkan pengembang tentang konstruksi yang secara teknis legal namun berbahaya. Misalnya, Asosiasi Keandalan Perangkat Lunak Industri Motor (MISRA) telah menerbitkan pedoman untuk penggunaan bahasa C dalam sistem kritis,[4] dan beberapa kompiler dapat secara opsional menerapkan beberapa atau semua pedoman ini yang pada dasarnya merupakan subset dari bahasa tersebut dengan mengecuali kan konstruksi yang diyakini mengarah pada ke perangkat lunak yang tidak dapat diandalkan. Beberapa pedoman MISRA bersifat nasihat dan mungkin menghasilkan peringatan, bukan kesalahan; sekali lagi, jika aturan MISRA diaktifkan, kompiler harus dipaksa untuk menghasilkan kesalahan build yang fatal pada konstruksi yang tidak patuh.

Para penulis tentu saja tidak merekomendasikan agar semua organisasi pembangunan mengadopsi kepatuhan penuh terhadap MISRA sebagai bagian dari standar pengkodean mereka. Sebaliknya, ada alasan bagus untuk tidak menerapkan seluruh standar. Setelah manajemen memutuskan untuk mengaktifkan pemeriksa aturan MISRA yang akan memaksa

[4] MISRA-C:2004. Guidelines for the Use of the C Language in Critical Systems. The Motor Industry Software Reliability Association; October 2004.

pembuatan produk gagal pada konstruksi kode sumber yang tidak sesuai, pengembang harus segera mengedit kode untuk memperbaiki masalah yang ditemukan. Fase pengeditan ini memerlukan biaya: waktu yang dihabiskan untuk mengubah kode, biaya pengujian ulang, dan risiko penambahan kelemahan baru selama proses pengeditan. Oleh karena itu, manajemen harus berhati-hati ketika mengadopsi aturan pengkodean baru. Studi kasus berikut menunjukkan kebutuhan ini.

Studi Kasus: MISRA C:2004 dan MISRA C++:2008

Seperti hampir semua standar yang berhubungan dengan bahasa, MISRA mempunyai banyak peraturan yang baik dan beberapa peraturan yang mungkin dipertanyakan atau tidak sesuai untuk beberapa kelas pengguna dan aplikasi. MISRA 2004, dengan 141 aturan, menetapkan beberapa pedoman yang meragukan dalam standar asli MISRA 1998. Jika MISRA digunakan sebagai bagian dari standar pengkodean, penerapan hanya sebagian saja dapat diterima; namun, bagian tersebut harus dipertimbangkan dan disetujui secara hati-hati oleh manajemen. Penting juga bahwa pemeriksa MISRA (sering kali dibangun langsung ke dalam kompiler) dapat secara selektif mengaktifkan dan menonaktifkan aturan tertentu dalam modul dan fungsi kode individual.

Berikut ini adalah contoh beberapa aturan MISRA yang menunjukkan beberapa kelemahan bahasa pemrograman C dan bagaimana penggunaan MISRA secara selektif akan membantu menghindarinya:

1 Aturan 7.1: Konstanta oktal (selain nol) dan rangkaian escape oktal tidak boleh digunakan. Contoh berikut menunjukkan kegunaan aturan ini:

```
a | = 256;
b | = 128;
c | = 064
```

Pernyataan pertama menetapkan bit kedelapan dari variabel a. Pernyataan kedua menetapkan bit ketujuh dari variabel b. Namun, pernyataan ketiga tidak menetapkan bit keenam dari variabel c. Karena konstanta 064 dimulai dengan 0, maka dalam standar C ditafsirkan sebagai nilai oktal. Oktal 64 sama dengan 0x34 dalam heksadesimal; pernyataan tersebut dengan demikian menetapkan bit kedua, keempat, dan kelima dari variabel c.

Karena bilangan oktal berkisar dari nol hingga tujuh, pengembang dengan mudah salah mengartikannya sebagai bilangan desimal. MISRA menghindari masalah ini dengan mengharuskan semua konstanta ditentukan sebagai angka desimal atau heksadesimal.

2 Aturan 8.1: Fungsi harus memiliki deklarasi prototipe dan prototipe harus terlihat pada definisi dan pemanggilan fungsi. Pembahasan informatif MISRA untuk aturan ini mencakup rekomendasi yang masuk akal agar prototipe fungsi untuk fungsi eksternal dideklarasikan dalam file header dan kemudian disertakan oleh semua file sumber yang berisi definisi fungsi atau salah satu referensinya. Perlu dicatat bahwa pemeriksa MISRA mungkin hanya memvalidasi bahwa beberapa deklarasi prototipe ada untuk panggilan ke suatu fungsi. Pemeriksa mungkin tidak dapat memvalidasi bahwa semua referensi ke fungsi tertentu

didahului oleh prototipe yang sama. Prototipe yang tidak cocok dapat menyebabkan bug berbahaya, yang lebih buruk daripada tidak memiliki prototipe apa pun.

Sebagai contoh, mari kita pertimbangkan definisi fungsi C dan referensi kode berikut, masing-masing terletak di file sumber terpisah:

File1:

```
void read_temp_sensor(float *ret)
{
    *ret = *(float *)0xfeff0;
}
```

File2:

```
float poll_temperature(void)
{
extern float read_temp_sensor(void);
    return read_temp_sensor();
}
```

Fragmen kode sebelumnya benar-benar legal ANSI/ISO C. Namun, perangkat lunak ini akan gagal karena referensi dan definisi read_temp_sensor tidak kompatibel (yang pertama ditulis untuk mengambil nilai kembalian fungsi, dan yang terakhir ditulis untuk mengembalikan nilai melalui parameter referensi).

Salah satu praktik pengkodean yang buruk yang dijelaskan dalam contoh sebelumnya adalah penggunaan deklarasi fungsi eksternal di dekat kode yang berisi referensi. Meskipun ANSI C yang ketat memerlukan deklarasi prototipe, cakupan deklarasi ini tidak tercakup dalam spesifikasi. Aturan MISRA 8.6, “fungsi harus dideklarasikan pada cakupan file,” berupaya mencegah kesalahan pengkodean ini dengan tidak mengizinkan deklarasi fungsi pada tingkat kode fungsi. Namun, fragmen kode berikut akan lulus tes MISRA ini namun gagal dengan cara yang sama seperti contoh sebelumnya:

```
extern float read_temp_sensor(void);
    float poll_temperature(void)
    {
        return read_temp_sensor();
    }
```

Meskipun MISRA tidak secara eksplisit melarang deklarasi fungsi di luar file header, pembatasan ini merupakan tambahan standar pengkodean yang disarankan. Mendeklarasikan semua fungsi dalam file header tentu saja membuat kesalahan ini lebih kecil kemungkinannya namun tetap gagal: file header yang berisi deklarasi tidak boleh digunakan dalam file sumber yang berisi definisi yang tidak kompatibel.

Hanya ada satu cara untuk menjamin bahwa deklarasi dan prototipe definisi cocok: mendeteksi ketidaksesuaian menggunakan analisis seluruh program. Analisis ini dapat dilakukan oleh penganalisis kode statis atau oleh linker/loader program lengkap. Kami menjelaskan pendekatan linker di sini sebagai ilustrasi tentang bagaimana rantai alat

berkualitas tinggi dapat menjadi sangat penting dalam menegakkan standar pengkodean. Saat mengkompilasi fragmen kode yang disebutkan di atas, kompiler dapat memasukkan beberapa penanda ke dalam file objek keluarannya, seperti simbol khusus dalam tabel simbol atau entri relokasi khusus, yang menjelaskan tanda tangan dari tipe kembalian dan tipe parameter yang digunakan dalam pemanggilan fungsi. Ketika definisi fungsi dikompilasi, kompiler juga mengeluarkan tanda tangan untuk definisi tersebut. Pada waktu tautan, ketika gambar akhir yang dapat dieksekusi dibuat, linker/loader membandingkan tanda tangan untuk fungsi dengan nama yang sama dan menghasilkan kesalahan jika ada tanda tangan yang tidak kompatibel terdeteksi. Pemeriksaan tambahan ini akan menambah overhead yang dapat diabaikan pada waktu pembuatan (linker sudah harus memeriksa referensi fungsi untuk melakukan relokasi) namun menjamin kompatibilitas parameter fungsi dan tipe kembalian sehingga meningkatkan keandalan dan kualitas perangkat lunak yang dihasilkan.

Salah satu keuntungan utama dari pendekatan pemeriksaan waktu tautan adalah kemampuan untuk mencakup perpustakaan (dengan asumsi perpustakaan dikompilasi dengan fitur ini) yang kode sumbernya mungkin tidak tersedia untuk analisis statis.

3 Aturan 8.9: Pengidentifikasi dengan hubungan eksternal harus mempunyai tepat satu definisi eksternal. Aturan ini serupa dengan aturan sebelumnya. Definisi variabel yang tidak cocok dapat menyebabkan kerentanan yang tidak dapat ditangkap oleh kompiler bahasa standar. Mari kita perhatikan contoh berikut di mana suhu variabel hanya boleh bernilai antara 0 dan 255:

File1:

```
#include <stdio.h>
unsigned int temperature;
int main(void)
{
    set_temp();
printf("temperature = %d\n", temperature);
return 0;
}
```

File2:

```
unsigned char temperature;
void set_temp(void)
{
    Temperature = 10:
}
```

Tanpa pemeriksaan kesalahan tambahan di luar standar C, program ini akan dibangun tanpa kesalahan meskipun definisi suhu tidak sesuai. Pada mesin big-endian dengan tipe int 32-bit dan tipe char 8-bit, fungsi ini akan dijalankan sebagai berikut:

```
Temperature = 167772160
```

Seperti contoh sebelumnya dengan prototipe fungsi, analisis antar modul diperlukan untuk mendeteksi ketidakcocokan ini. Dan sekali lagi, linker/loader adalah alat yang masuk akal untuk melakukan pemeriksaan ini.

4 Aturan 8.11: Kelas penyimpanan statis yang ditentukan harus digunakan dalam definisi dan deklarasi objek dan fungsi yang memiliki hubungan internal. Dua pemrogram dapat menggunakan variabel dengan nama yang sama untuk tujuan independen dalam modul independen dalam program yang sama.

Modifikasi variabel pada satu modul akan merusak instance modul lainnya dan sebaliknya. Selain itu, variabel global mungkin lebih terlihat oleh penyerang (jika, misalnya, tabel simbol global untuk program tersebut tersedia), sehingga membuka peluang untuk mengubah data penting dengan malware. Aturan MISRA 8.11 dirancang untuk mencegah hal ini dengan menerapkan kebijakan yang secara umum baik yaitu membatasi ruang lingkup deklarasi hingga batas minimum yang disyaratkan.

Meskipun aturan MISRA 8.9 dan 8.11 akan mencegah berbagai bentuk kesalahan definisi dan penggunaan yang tidak sesuai, aturan tersebut tidak akan mencegah semua kejadian seperti itu. Contoh lain dari resolusi simbolik yang tidak tepat berkaitan dengan penggunaan definisi perpustakaan yang diekspor secara tidak sengaja. Perpustakaan sering kali digunakan untuk mengumpulkan modul kode yang menyediakan serangkaian fungsi terkait. Faktanya, penggunaan perpustakaan untuk mengumpulkan perangkat lunak yang dapat digunakan kembali di seluruh proyek layak disebutkan dalam standar pengkodean. Misalnya, sebagian besar sistem operasi dilengkapi dengan pustaka C, misalnya libc.so, yang menyediakan dukungan untuk runtime C, termasuk manipulasi string, manajemen memori, dan fungsi input/output konsol. Sebuah proyek perangkat lunak yang kompleks kemungkinan besar mencakup berbagai perpustakaan khusus proyek.

Pustaka ini mengekspor fungsi yang dapat dipanggil dengan kode aplikasi. Masalah keandalan muncul karena fakta bahwa pengembang perpustakaan dan pengembang aplikasi mungkin tidak secara akurat memprediksi atau menentukan antarmuka perpustakaan yang diekspor secara apriori. Perpustakaan dapat mendefinisikan fungsi-fungsi yang terlihat secara global yang dimaksudkan untuk digunakan hanya oleh modul lain dalam perpustakaan. Namun begitu fungsi-fungsi ini ditambahkan ke namespace global pada waktu link, linker dapat menyelesaikan referensi yang dibuat oleh aplikasi yang tidak dimaksudkan untuk mencocokkan definisi di perpustakaan.

Sebagai contoh, mari kita pertimbangkan sebuah aplikasi yang menggunakan fungsi cetak. Pengembang aplikasi membayangkan penggunaan perpustakaan pencetakan yang disediakan oleh tim manajemen printer. Namun, tim manajemen font membuat perpustakaan, juga digunakan oleh pengembang aplikasi, yang menyediakan serangkaian fungsi manipulasi font. Tim manajemen font mendefinisikan fungsi cetak yang dimaksudkan untuk digunakan oleh modul lain dalam perpustakaan manajemen font. Namun, jika tidak ada fasilitas untuk membatasi namespace perpustakaan (penggunaan fasilitas tersebut, jika tersedia, harus tercakup dalam standar pengkodean), fungsi cetak perpustakaan font mungkin

secara tidak sengaja digunakan oleh linker untuk menyelesaikan masalah pencetakan. referensi yang dibuat oleh pengembang aplikasi, menyebabkan sistem gagal.

Oleh karena itu, masalah ini mungkin perlu diselesaikan dengan sesuatu selain front end kompiler. Salah satu metodenya adalah dengan menggunakan program utilitas rantai alat yang menyembunyikan definisi perpustakaan sehingga digunakan oleh linker ketika menyelesaikan referensi intra-perpustakaan tetapi diabaikan ketika menyelesaikan referensi ekstra-perpustakaan. Platform Windows menggunakan file ekspor perpustakaan yang ditentukan pengguna untuk mencapai pemisahan ini.[5] Saat membuat DLL Windows, pengembang menentukan fungsi mana yang diekspor. Fungsi yang tidak disertakan dalam file ekspor tidak akan digunakan untuk menyelesaikan referensi aplikasi.

Beberapa bahasa tingkat tinggi, seperti C++ dan Ada, melakukan pekerjaan yang lebih baik dalam menegakkan konsistensi tipe dan spasi nama secara otomatis dibandingkan bahasa lain seperti C. Pilihan bahasa mungkin membuat aturan standar pengkodean tertentu menjadi mudah untuk diterapkan.

5 Aturan 16.2: Fungsi tidak boleh menyebut dirinya sendiri, baik secara langsung maupun tidak langsung. Meskipun fungsi rekursif langsung mudah dideteksi, dan hampir selalu merupakan ide buruk dalam sistem tertanam yang terbatas sumber daya atau kritis terhadap keselamatan karena risiko stack overflow, rekursi tidak langsung bisa jauh lebih sulit dideteksi. Aplikasi canggih dengan grafik panggilan kompleks dan panggilan melalui penunjuk fungsi mungkin berisi rekursi tidak langsung yang tidak disadari. Ini adalah kasus lain di mana penganalisis antar-modul, seperti linker/loader, diperlukan untuk mendeteksi siklus dalam grafik panggilan program. Menangani semua kasus pemanggilan fungsi tidak langsung, seperti penunjuk fungsi yang ditetapkan secara dinamis, tabel penunjuk fungsi, dan fungsi virtual C++, bisa sangat sulit untuk alat otomatis karena ambiguitas fungsi potensial yang mungkin direferensikan oleh penunjuk ini. Pengembang harus mencoba kasus uji sederhana dengan pemeriksa MISRA untuk melihat batasan apa yang dimilikinya. Jika vendor alat tidak dapat meningkatkan atau menyesuaikan alat untuk memenuhi kebutuhan spesifik, pengembang harus mempertimbangkan pilihan alat lain atau mengadopsi aturan standar pengkodean yang lebih ketat untuk membatasi penggunaan bentuk pemanggilan fungsi tidak langsung yang bermasalah.

MISRA untuk C++ dirilis pada tahun 2008[6] dan, seperti yang diharapkan, memiliki tumpang tindih yang signifikan dengan MISRA C. Namun, standar MISRA C++ mencakup 228 aturan, sekitar 50% lebih banyak dari standar MISRA C. Dasar tambahan mencakup aturan yang berkaitan dengan fungsi virtual, penanganan pengecualian, namespace, parameter referensi, akses ke data kelas yang dienkapsulasi, dan aspek lain yang spesifik untuk bahasa C++.

6 Aturan 9-3-2: Fungsi anggota tidak boleh mengembalikan pegangan non-const ke data kelas. Contoh sederhana dari kelas yang tidak patuh adalah sebagai berikut:

---

[5] Working with Import Libraries and Export Files. Microsoft Development Network. http://msdn.microsoft.com/library/default.asp?url=/library/enus/vccore/html/_core_working_with_import_libraries_and_export_files.asp.

[6] MISRA C++:2008. Guidelines for the Use of the C++ Language in Critical Systems. The Motor Industry Software Reliability Association; June 2008.

```
#include <stdint.h>
class temperature
{
public:
     int32_t &gettemp(void) { return the_temp; }
private:
     int32_t the_temp;
}
int main(void)
{
     temperature t;
     int32_t &temp_ref =t.gettemp();
     temp_ref = 10;
     return 0:
}
```

Salah satu tujuan desain utama bahasa C++ adalah untuk mempromosikan antarmuka yang bersih dan mudah dipelihara dengan mendorong penggunaan penyembunyian informasi dan enkapsulasi data. Kelas C++ biasanya dibentuk dengan kombinasi data internal (pribadi) dan fungsi anggota kelas. Fungsi-fungsi tersebut menyediakan antarmuka terdokumentasi untuk klien kelas, memungkinkan pelaksana kelas untuk memodifikasi struktur data internal dan implementasi anggota tanpa mempengaruhi portabilitas klien. Fungsi anggota kelas sebelumnya gettemp mengembalikan alamat struktur data internal. Akses langsung data internal ini oleh klien melanggar prinsip berorientasi objek C++. Peningkatan nyata (dan penerapan yang sesuai dengan MISRA) dari kelas sampel sebelumnya adalah sebagai berikut:

```
#include <stdint.h>
class temperature
{
     public:
     int32_t gettemp(void) { return the_temp; }
     void settemp(int32_t t) { the_temp = t; }
     private:
     int32_t the_temp;
}
int main(void)
{
     temperature t;
     t.settemp(10);
     return 0;
}
```

Jika pemilik kelas suhu memutuskan bahwa hanya delapan bit data yang diperlukan untuk menyimpan suhu yang valid, maka dia dapat memodifikasi kelas internal tanpa mempengaruhi klien kelas:

```
#include <stdint.h>
class temperature
{
public:
    int32_t gettemp(void) { return the_temp; }
    void settemp(int32_t t) ( the_temp = t; )
private:
    int8_t the_temp:
}
```

Implementasi yang tidak patuh memerlukan modifikasi pada kode sisi klien karena perubahan ukuran.

C++ tertanam

Sejumlah fitur lanjutan di C++, seperti pewarisan berganda, dapat mengakibatkan pemrograman rawan kesalahan, sulit dipahami dan dipelihara, serta tidak dapat diprediksi atau tidak efisien dalam hal jejak dan kecepatan eksekusi. Karena kelemahan ini, konsorsium vendor semikonduktor dan alat pengembangan menciptakan spesifikasi subset C++ yang disebut C++ Tertanam yang telah digunakan secara luas selama lebih dari satu dekade.[7]

> *Tujuan dari C++ Tertanam adalah untuk memberikan pengembang sistem tertanam yang berasal dari latar belakang bahasa C dengan peningkatan bahasa pemrograman yang membawa manfaat utama berorientasi objek dari C++ tanpa beban risikonya.*

C++ yang tertanam menghilangkan fitur C++ berikut:

- Warisan berganda
- Kelas dasar virtual
- Pemeran gaya baru
- Penentu yang bisa berubah
- Ruang nama
- Identifikasi jenis waktu proses (RTTI)
- Pengecualian
- Templat

Salah satu contoh alasan untuk C++ Tertanam adalah kesulitan dalam menentukan waktu eksekusi dan jejak penanganan pengecualian C++. Ketika pengecualian terjadi, kode penanganan pengecualian yang dihasilkan oleh kompiler akan memanggil destruktor pada semua objek otomatis yang dibuat sejak blok percobaan yang berlaku dijalankan. Jumlah dan waktu eksekusi rantai destruktor ini mungkin sangat sulit diperkirakan dalam aplikasi yang canggih.

Selanjutnya, kompiler menghasilkan kode penanganan pengecualian untuk melepaskan tumpukan panggilan yang menghubungkan pengendali ke blok percobaan aslinya. Jejak tambahan ini mungkin signifikan dan sulit diprediksi. Karena runtime C++ standar dikompilasi untuk mendukung penanganan pengecualian, fitur ini menambahkan kode yang membengkak ke program yang bahkan tidak menggunakan mekanisme penanganan

[7] The Embedded C++ Specification, Version WP-AM-003. The Embedded C++ Technical Committee; October 1999.

pengecualian coba dan tangkap.[8] Untuk alasan ini, pustaka runtime yang dibuat khusus mendukung subset bahasa yang dikurangi biasanya menyertai rantai alat C++ Tertanam.

Kekhawatiran terhadap jejak kaki juga menyebabkan templat C++ tidak disertakan dalam standar C++ Tertanam; dalam beberapa kasus, kompiler mungkin membuat instance sejumlah besar fungsi dari templat, yang menyebabkan pengasapan kode yang tidak terduga. Tentu saja, beberapa fitur yang dihapus ini bisa sangat berguna. Penggunaan templat secara hati-hati dapat menghindari penggembungan kode yang tidak perlu sekaligus membuktikan antarmuka kode sumber yang lebih sederhana dan lebih mudah dipelihara.

Karena alasan ini, banyak kompiler menyediakan varian C++ Tertanam yang memungkinkan organisasi pengembangan untuk menambahkan kembali fitur-fitur yang mungkin dapat diterima untuk pengembangan yang kritis terhadap keamanan, terutama jika fitur-fitur tersebut digunakan dengan bijaksana (seperti menerapkan beberapa atau semua aturan MISRA C++). Misalnya, kompiler C++ Green Hills Software menyediakan opsi untuk mengizinkan penggunaan templat, pengecualian, dan fitur individual lainnya dengan dialek C++ Tertanam (bersama dengan mengaktifkan pemeriksaan MISRA).

Pengendalian Kompleksitas

Banyak yang telah dipublikasikan mengenai manfaat mengurangi kompleksitas pada tingkat fungsi. Memecah modul perangkat lunak menjadi fungsi-fungsi yang lebih kecil membuat setiap fungsi lebih mudah untuk dipahami, dipelihara, dan diuji.[9] Kita dapat menganggap ini sebagai meta-partisi: menerapkan paradigma komponenisasi perangkat lunak yang disebutkan di atas pada tingkat yang lebih rendah, terprogram. Aturan pengkodean batasan kompleksitas mudah diterapkan pada waktu kompilasi dengan menghitung metrik kompleksitas dan menghasilkan kesalahan waktu kompilasi ketika metrik kompleksitas terlampaui. Sekali lagi, karena kompiler sudah menelusuri pohon kode, kompiler tidak memerlukan waktu build tambahan yang signifikan untuk menerapkan komputasi kompleksitas sederhana, seperti metrik McCabe yang populer. Karena kompiler menghasilkan kesalahan aktual yang menunjukkan fungsi yang melanggar, pengembang tidak dapat secara tidak sengaja membuat kode yang melanggar aturan.

Gunakan alat otomatis untuk menerapkan metrik kompleksitas maksimum dan memastikan bahwa maksimum ini bermakna (misalnya nilai McCabe 20). Mengadopsi aturan standar pengkodean yang memungkinkan nilai kompleksitas McCabe sebesar 200 tidak ada gunanya; sebagian besar basis kode lama akan patuh meskipun memiliki kode mirip spageti yang sulit dipahami, diuji, dan dipelihara. Pemilihan nilai kompleksitas maksimum tertentu masih dapat diperdebatkan. Jika basis kode yang ada termodulasi dengan baik, suatu nilai dapat dipilih yang memungkinkan sebagian besar kode yang dipartisi dengan benar untuk dikompilasi; kode masa depan akan dijaga dengan standar ketat yang sama.

Ketika metrik kompleksitas diterapkan pada basis kode besar yang sebelumnya belum pernah dianalisis, kemungkinan besar sejumlah kecil fungsi besar akan gagal dalam uji kompleksitas. Manajemen kemudian perlu mempertimbangkan risiko penguraian fungsi yang besar dengan risiko perubahan kode sama sekali. Memodifikasi sepotong kode yang,

[8] Haden M. Embedded C++ Slashes Code Size and Boosts Execution. http://www.ghs-rtos.com/wp/ ec++article2.html; 1998.
[9] Watson AH, McCabe TJ. NIST Special Publication 500-235: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric; September 1996.

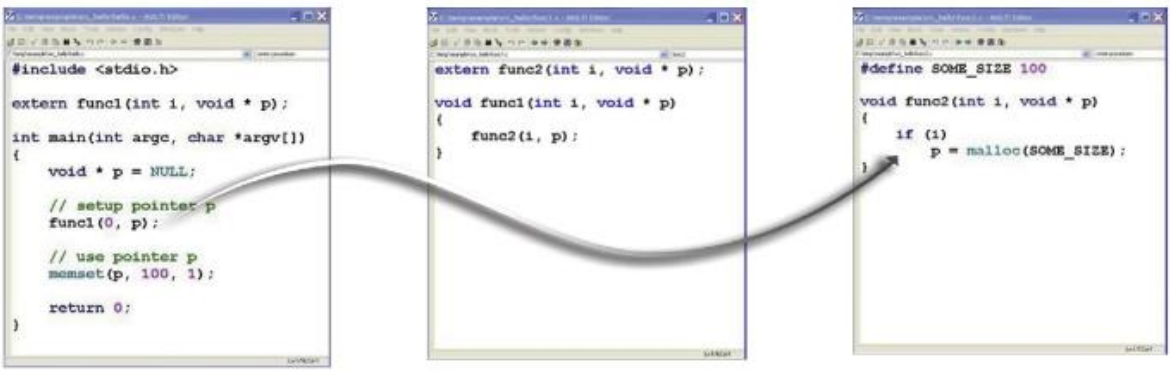
meskipun rumit, diterapkan dengan baik (terbukti digunakan) dan menjalankan fungsi penting dapat mengurangi keandalan dengan meningkatkan kemungkinan munculnya cacat. Alat penegakan kompleksitas harus memberikan kemampuan untuk mengizinkan pengecualian terhadap aturan penegakan kompleksitas untuk fungsi tertentu yang memenuhi profil ini. Pengecualian, tentu saja, harus selalu disetujui oleh manajemen dan didokumentasikan. Standar pengkodean tidak boleh mengizinkan pengecualian untuk kode yang dikembangkan setelah penerapan aturan pengkodean. Jenis kebijakan standar pengkodean ini sesuai dengan semangat mereka sekaligus memaksimalkan efisiensi, memungkinkan mereka untuk diterapkan secara efektif dalam proyek-proyek lama.

Analisis Kode Sumber Statis

Penganalisis kode sumber statis berupaya menemukan urutan kode yang, bila dijalankan, dapat mengakibatkan buffer overflow, kebocoran sumber daya, atau banyak masalah keamanan dan keandalan lainnya. Penganalisis kode sumber efektif dalam menemukan kelas kelemahan signifikan yang tidak terdeteksi oleh kompiler selama pembuatan standar dan sering kali juga tidak terdeteksi selama pengujian runtime.

Kebanyakan penganalisis kode sumber statis menggunakan jenis ujung depan kompiler yang sama dengan yang digunakan untuk mengkompilasi kode. Faktanya, idealnya, penganalisis kode sumber statis harus diintegrasikan dengan compiler sehari-hari untuk memaksimalkan penggunaan dan mengurangi kompleksitas rantai alat. Selain itu, pemeriksaan terintegrasi memungkinkan penguraian kode sumber dilakukan hanya sekali, bukan dua kali. Penggunaan ujung depan kompiler adalah hal yang wajar karena penganalisis memanfaatkan algoritme aliran data kompiler yang sudah ada untuk menjalankan misi pencarian bugnya.

Kompiler tipikal akan mengeluarkan peringatan dan kesalahan untuk beberapa masalah kode potensial dasar, seperti pelanggaran standar bahasa atau penggunaan konstruksi yang ditentukan implementasi. Sebaliknya, penganalisis kode sumber statis melakukan analisis program lengkap, menemukan bug yang disebabkan oleh interaksi kompleks antara potongan kode yang mungkin tidak berada dalam file sumber yang sama (lihat Gambar 3.1).



Gambar 3.1 Analisis statis antar modul.

Penganalisis menentukan jalur eksekusi potensial melalui kode, termasuk jalur ke dalam dan melintasi panggilan subrutin, dan bagaimana nilai objek program (seperti variabel

mandiri atau bidang dalam agregat) dapat berubah di seluruh jalur ini. Objek bisa berada di memori atau di register mesin.

Penganalisis mencari berbagai jenis kekurangan. Ini mencari bug yang biasanya dikompilasi tanpa kesalahan atau peringatan. Berikut ini adalah daftar beberapa kesalahan umum yang akan dideteksi oleh penganalisis kode sumber statis modern:

Potensi dereferensi penunjuk NULL

Akses di luar area yang dialokasikan (misalnya array atau buffer yang dialokasikan secara dinamis); atau dikenal sebagai buffer overflow

Menulis ke memori yang berpotensi hanya-baca

Membaca objek yang berpotensi tidak diinisialisasi

Kebocoran sumber daya (misalnya kebocoran memori dan kebocoran deskriptor file)

Penggunaan memori yang telah dibatalkan alokasinya

Penggunaan memori di luar cakupan (misalnya, mengembalikan alamat variabel otomatis dari subrutin)

Kegagalan menetapkan nilai kembalian dari subrutin

Buffer dan aliran bawah array

Penganalisis juga memiliki pengetahuan tentang berapa banyak fungsi perpustakaan runtime standar yang berperilaku. Misalnya, penganalisis mengetahui bahwa subrutin seperti free harus meneruskan pointer ke memori yang dialokasikan oleh subrutin seperti malloc. Penganalisis menggunakan informasi ini untuk mendeteksi kesalahan dalam kode yang memanggil atau menggunakan hasil panggilan ke fungsi-fungsi ini.

Membatasi Positif Palsu

Penganalisis juga dapat diajarkan tentang properti subrutin yang ditentukan pengguna. Misalnya, jika sistem alokasi memori khusus digunakan, penganalisis dapat diajarkan untuk mencari penyalahgunaan sistem ini. Dengan mengajari penganalisis tentang properti subrutin, pengguna dapat mengurangi jumlah positif palsu. Positif palsu adalah potensi cacat yang diidentifikasi oleh penganalisis yang sebenarnya tidak terjadi selama eksekusi program. Tentu saja, salah satu tujuan desain utama penganalisis kode sumber statis adalah meminimalkan jumlah positif palsu sehingga pengembang dapat meminimalkan waktu untuk melihatnya. Jika alat analisa menghasilkan terlalu banyak kesalahan positif, maka alat tersebut menjadi tidak relevan karena para insinyur akan mengabaikan hasilnya. Penganalisis kode sumber statis modern jauh lebih baik dalam membatasi kesalahan positif dibandingkan penganalisis UNIX tradisional seperti lint. Namun, karena penganalisis statis tidak mampu memahami semantik program secara lengkap, maka tidak mungkin menghilangkan kesalahan positif sepenuhnya. Dalam beberapa kasus, cacat yang ditemukan oleh penganalisis mungkin tidak mengakibatkan kesalahan program yang fatal, namun dapat menunjukkan konstruksi yang meragukan yang harus diperbaiki untuk meningkatkan kejelasan kode. Contoh yang baik dari hal ini adalah penulisan ke variabel yang tidak pernah dibaca lagi.

Studi Kasus: Aplikasi Keamanan Internet Sumber Terbuka

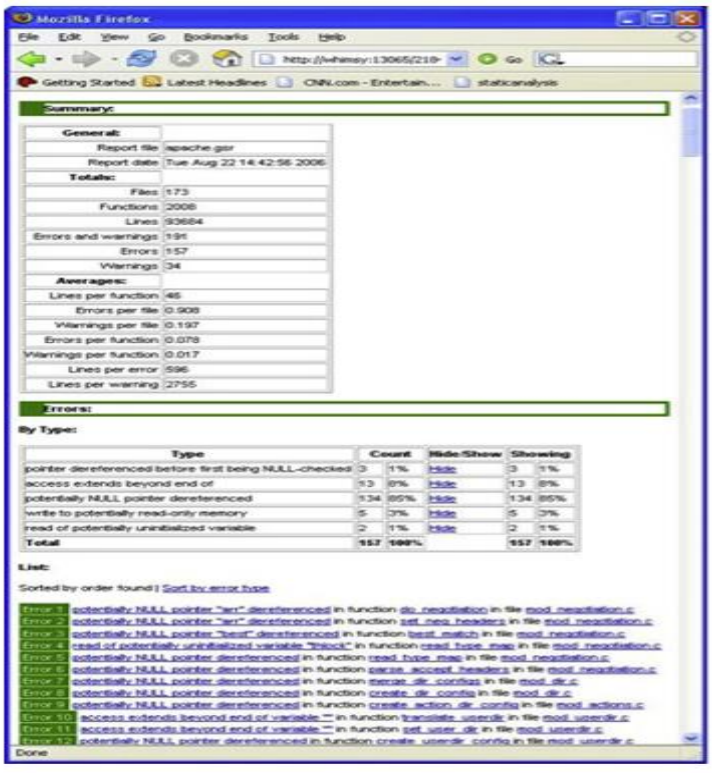
Untuk membantu menunjukkan jenis kesalahan pengkodean yang dapat dideteksi dan dicegah secara efisien menggunakan analisis kode sumber statis, kami mempertimbangkan studi kasus dari tiga aplikasi sumber terbuka yang populer dan kritis terhadap keamanan, yaitu

Apache, OpenSSL, dan sendmaild yang dianalisis menggunakan sumber statis DoubleCheck dari Green Hills Software penganalisis kode.

Server protokol transfer hiperteks (HTTP) sumber terbuka Apache adalah server web paling populer di dunia, yang mendukung sebagian besar situs web di Internet. Mengingat keberadaan Apache di mana-mana dan ketergantungan dunia pada Internet, keandalan dan keamanan Apache merupakan perhatian penting bagi kita semua. Cacat serius pada Apache dapat menyebabkan ketidaknyamanan yang luas, kerugian finansial, atau lebih buruk lagi. Server web Apache terdiri dari sekitar 200.000 baris kode, 80.000 pernyataan individual yang dapat dieksekusi, dan 2.000 fungsi.

OpenSSL adalah implementasi open source dari Secure Sockets Layer (SSL) dan Transport Layer Security (TLS) serta perpustakaan algoritma kriptografi yang komprehensif. TLS adalah implementasi ulang SSL modern, meskipun SSL sering digunakan sebagai istilah umum yang mencakup kedua protokol. TLS dan OpenSSL dibahas lebih rinci di Bab 5.

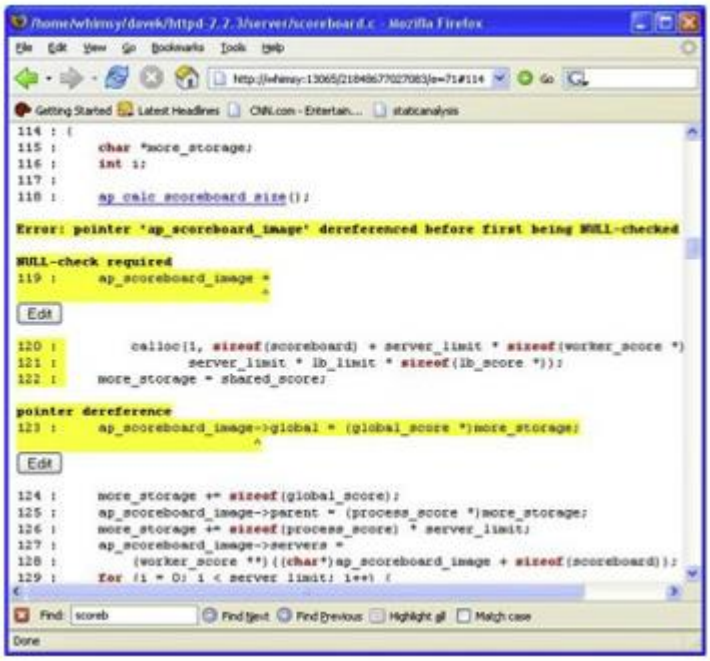
SSL membentuk dasar dari sebagian besar komunikasi aman di Internet. Misalnya, SSL memungkinkan pengguna mengirim informasi kartu kredit pribadi dengan aman dari browser mereka ke server jarak jauh pedagang online. Selain terlibat erat dengan komunikasi data, OpenSSL berisi implementasi berbagai algoritma kriptografi yang digunakan untuk mengamankan data dalam perjalanan. OpenSSL tersedia untuk Windows; namun, OpenSSL adalah implementasi SSL standar untuk Linux dan UNIX di seluruh dunia. Selain itu, karena ketentuan lisensinya yang liberal (bukan GPL), OpenSSL telah digunakan sebagai dasar untuk sejumlah penawaran komersial. Seperti Apache, OpenSSL adalah landasan komunikasi Internet aman di seluruh dunia. Cacat pada perangkat lunak ini dapat menimbulkan dampak buruk yang luas.



Gambar 3.2 Laporan ringkasan penganalisa kode sumber statis.

OpenSSL terdiri dari sekitar 175.000 baris kode, 85.000 pernyataan individual yang dapat dieksekusi, dan 5.000 fungsi.

Meskipun penggunaannya menurun, sendmail adalah salah satu perangkat lunak server surat elektronik paling populer yang digunakan di Internet. Sendmail telah menjadi agen transfer surat elektronik de facto untuk sistem UNIX (dan selanjutnya, Linux) sejak awal 1980an. Mengingat ketergantungan pada surat elektronik, stabilitas dan keamanan sendmail tentu menjadi perhatian penting bagi banyak orang. Nama sendmail mungkin membuat orang berpikir bahwa aplikasi ini tidak terlalu rumit. Siapapun yang pernah mencoba mengkonfigurasi server sendmail tahu sebaliknya. Sendmail terdiri dari sekitar 70.000 baris kode, 32.000 pernyataan individual yang dapat dieksekusi, dan 750 fungsi.



Gambar 3.3 Tampilan kerentanan kode sumber dalam konteks.

Output dari Penganalisis Kode Sumber Statis

Banyak penganalisis kode sumber terkemuka menghasilkan serangkaian halaman web yang intuitif, didukung oleh server web terintegrasi. Pengembang dapat menelusuri ringkasan tingkat tinggi dari berbagai kelemahan yang ditemukan oleh penganalisis (lihat Gambar 3.2) dan kemudian mengklik hyperlink untuk menyelidiki masalah tertentu. Di dalam tampilan masalah tertentu, kesalahan ditampilkan sejajar dengan kode di sekitarnya, sehingga mudah dipahami (lihat Gambar 3.3). Nama fungsi dan objek lainnya diberi hyperlink untuk kemudahan penelusuran kode sumber. Karena halaman web dijalankan di bawah server web, hasilnya dapat dengan mudah dibagikan dan dijelajahi oleh anggota tim pengembangan mana pun. Bagian berikut memberikan contoh kelemahan aktual di Apache, OpenSSL, dan sendmail yang ditemukan oleh DoubleCheck. Hasilnya dikelompokkan berdasarkan jenis kesalahan, dengan satu atau lebih contoh dari setiap jenis kesalahan per bagian.

Potensi Akses Pointer NULL

Sejauh ini kelemahan paling umum yang ditemukan oleh penganalisis di ketiga rangkaian yang diuji adalah potensi akses pointer NULL. Banyak kasus yang melibatkan panggilan ke subrutin alokasi memori yang diikuti dengan akses dari pointer yang dikembalikan tanpa terlebih dahulu memeriksa pengembalian NULL. Ini adalah masalah ketahanan. Idealnya, semua kegagalan alokasi memori ditangani dengan baik. Jika ada kehabisan memori sementara, layanan mungkin terputus-putus tetapi tidak berhenti. Hal ini sangat penting khususnya untuk program server seperti Apache dan sendmail. Algoritma dapat diperkenalkan untuk mencegah penolakan layanan dalam kondisi kelebihan beban seperti yang disebabkan oleh serangan berbahaya.

Server web Apache, sendmail, dan OpenSSL semuanya menggunakan alokasi memori dinamis perpustakaan runtime C secara boros. Tidak seperti Java, yang melakukan pengumpulan sampah otomatis, alokasi memori dinamis menggunakan runtime C standar mengharuskan aplikasi itu sendiri menangani potensi kesalahan kehabisan memori. Jika alokasi memori gagal dan mengembalikan pointer NULL, referensi pointer berikutnya yang tidak dijaga dijamin akan menyebabkan crash fatal.

Dalam file sumber Apache scoreboard.c, kami memiliki pernyataan alokasi memori berikut:

```
ap_scoreboard_image = calloc(1,sizeof(scoreboard) +
    server_limit * sizeof(worker_score *) +
    server_limit * 1b_limit * sizeof(1b_score *)):
```

Jelasnya, ukuran alokasi memori ini bisa sangat besar. Sebaiknya pastikan alokasi berhasil sebelum mereferensikan konten ap_scoreboard_image. Namun, segera setelah pernyataan alokasi, kami menggunakan ini:

```
ap_score_board_image->global = (global_score
    *)more_storage:
```

Dereferensinya tidak dijaga, membuat aplikasi rentan terhadap crash yang fatal. Contoh lain dari Apache dapat ditemukan di file mod_auth_digest.c:

```
 MNentry = client_list->table[idx];
prev = NULL;
while (entry->next){/* find last entry */
    prev = entry;
    entry = entry->next;
    ...
}
```

Entri variabel direferensikan tanpa syarat di awal perulangan. Hal ini saja tidak akan menyebabkan penganalisa melaporkan kesalahan. Pada titik jalur eksekusi ini, penganalisis tidak memiliki bukti atau petunjuk spesifik bahwa entri mungkin NULL atau tidak valid. Namun, pernyataan berikut muncul setelah perulangan:

```
If (entry) {
...
}
```

Dengan memeriksa penunjuk entri NULL, pemrogram telah mengindikasikan bahwa entri tersebut mungkin NULL. Menelusuri ke belakang, penganalisis sekarang melihat bahwa dereferensi sebelumnya ke entri di bagian atas loop adalah kemungkinan referensi NULL. Contoh serupa berikut terdeteksi dalam aplikasi sendmail, dalam file queue.c, di mana kode tersebut secara tanpa syarat melakukan dereferensi variabel penunjuk tempqfp:

```
errno - sm_io_error(tempqfp);
```

sm_io_error adalah makro yang menyelesaikan pembacaan bidang flag tempqfp->f_. Nanti di fungsi yang sama, kita mendapat pemeriksaan NULL ini:

```
if (tempqfp != NULL)
     sm_io_close(tempqfp, SM_TIME_DEFAULT);
```

Selain itu, tidak ada intervensi penulisan ke tempqfp setelah dereferensi yang dicatat sebelumnya. Pemeriksaan NULL, tentu saja, menyiratkan bahwa tempqfp bisa jadi NULL; jika itu masalahnya, kodenya akan salah. Jika penunjuk dalam praktiknya tidak pernah menjadi NULL, maka pemeriksaan tambahan tidak diperlukan dan menyesatkan. Kecerobohan yang kelihatannya tidak berbahaya, bisa berubah menjadi kegagalan besar dalam kondisi tertentu. Di sendmail, ada banyak contoh lain dari dereferensi penunjuk yang tidak dijaga yang didahului atau diikuti dengan pemeriksaan NULL.

Contoh terakhir dalam kategori ini berasal dari OpenSSL, dalam file ssl_lib.c:

```
if (s->handshake_func=0) {
     SSLerr(SSL_F_SSL_SHUTDOWN. SSL_R_UNINITIALIZED):
}
```

Tak lama kemudian, kami mendapat pemeriksaan NULL pada pointer s:

```
if ((s != NULL) && !SSL_in_init(s))
```

Sekali lagi, pemrogram memberi tahu kita bahwa s bisa jadi NULL, namun penghormatan sebelumnya tidak dijaga.

Buffer Underflow

Buffer underflow didefinisikan sebagai upaya untuk mengakses memori sebelum buffer atau array yang dialokasikan. Mirip dengan buffer overflow, buffer underflows menyebabkan masalah berbahaya karena kerusakan memori yang tidak terduga. Cacat berikut pada file queue.c di sendmail ditemukan oleh analisis statis:

```
if ((qd =- 1 || qg =- 1) &&
     type != 120)
     ...
else {
     switch (type) {
     ...
     case 120:
          if (bitset(QP_SUBXF,
               Queue[qg]->qg_qpaths[qd].qp_subdirs))
               ...
     }
}
```

Seperti yang Anda lihat, pernyataan if menyiratkan bahwa qd atau qg mungkin menjadi e1 ketika tipenya 120. Namun dalam pernyataan switch berikutnya, selalu dieksekusi ketika tipenya 120, array Queue diindeks tanpa syarat melalui variabel qg. Jika qg adalah e1, ini akan menjadi arus bawah. Program ini tidak dipelajari secara mendalam untuk menentukan apakah qg memang bisa menjadi e1 ketika tipenya 120 dan karenanya mencapai kesalahan. Namun, jika qg tidak bisa menjadi e1 ketika tipenya 120, maka pemeriksaan if awal salah, menyesatkan, dan/atau tidak diperlukan.

Contoh lain dari buffer underflow ditemukan di file ssl_lib.c di OpenSSL:

```
P=buf;
sk = s->session->ciphers;
for (i=0; i < sk_SSL_CIPHER_num(sk); i++) {
     ...
     *(p++)=':';
}
p[-1] = '\0':
```

Penganalisis memberi tahu kita bahwa underflow terjadi ketika kode ini dipanggil dari file s_server.c. Dari tampilan situs panggilan di s_server.c, jelas bahwa penganalisis telah mendeteksi bahwa buf menunjuk ke awal buffer yang dialokasikan secara statis. Oleh karena itu, dalam kode ssl_lib.c, jika tidak ada cipher di cipher stack sk, maka akses p[e1] adalah underflow. Hal ini menunjukkan perlunya analisis antar-modul, karena tidak ada cara untuk mengetahui apa yang direferensikan buf tanpa memeriksa pemanggilnya. Jika dalam praktiknya jumlah cipher tidak bisa sama dengan 0, maka perulangan for harus diubah menjadi perulangan do untuk memperjelas bahwa perulangan harus selalu dieksekusi setidaknya sekali (memastikan bahwa p[e1] tidak tidak mengalir ke bawah).

Masalah lainnya adalah potensi buffer overflow. Tidak ada pemeriksaan yang dilakukan pada kode ssl_lib.c untuk memastikan bahwa jumlah cipher tidak melebihi ukuran parameter buf. Daripada mengandalkan konvensi, praktik pemrograman yang lebih baik adalah dengan meneruskan buf dan kemudian menambahkan kode untuk memeriksa bahwa overflow tidak terjadi.

Kebocoran Sumber Daya Dalam file speed.c di OpenSSL:

```
fds=malloc(multi*sizeof *fds);
```

fds adalah pointer lokal dan tidak pernah digunakan untuk mengosongkan memori yang dialokasikan sebelum kembali dari subrutin. Selain itu, fds tidak disimpan dalam variabel lain yang nantinya dapat dibebaskan. Jelas, ini adalah kebocoran memori. Serangan penolakan layanan yang sederhana pada OpenSSL adalah dengan memanggil atau menyebabkan dipanggilnya perintah kecepatan hingga seluruh memori habis.

Banyak yang berpendapat bahwa kualitas kode aplikasi open source populer tersebut diperkirakan relatif tinggi. Seperti yang dikatakan seseorang, "Dengan berbagi kode sumber, pengembang sumber terbuka membuat perangkat lunak lebih tangguh. Program digunakan dan diuji dalam konteks yang lebih beragam daripada yang dapat dihasilkan oleh seorang pemrogram, dan bug akan terungkap yang tidak akan ditemukan jika tidak ada."[10] Sayangnya, dalam aplikasi perangkat lunak yang kompleks seperti Apache, tidak mungkin semua kelemahan dapat dideteksi. ditemukan dengan pemeriksaan manual. Selain studi kasus ini, penganalisis kode statis komersial lainnya telah berhasil digunakan pada aplikasi sumber terbuka besar, termasuk sistem operasi Linux, untuk menemukan banyak kerentanan keamanan laten.

Banyak mekanisme tersedia untuk membantu perjuangan meningkatkan kualitas perangkat lunak, termasuk peningkatan paradigma pengujian dan desain. Namun penganalisis kode sumber otomatis adalah salah satu teknologi yang paling menjanjikan. Penggunaan analisis statis harus menjadi bagian wajib dari setiap proses pengembangan perangkat lunak organisasi yang sadar akan keamanan.

Alat Analisis Statis Mana yang Sebaiknya Digunakan Organisasi?

Jawaban terbaik untuk pertanyaan ini adalah organisasi pengembangan harus menggunakan banyak alat dari vendor berbeda. Penggunaan empiris dalam tim evaluasi keselamatan dan keamanan perangkat lunak pemerintah telah menunjukkan bahwa sebagian besar kelemahan perangkat lunak yang terdeteksi oleh satu penganalisis statis tidak akan terdeteksi oleh alat lain, dan sebaliknya. Banyak bentuk analisis statis program lengkap pada dasarnya sulit dilakukan, memerlukan algoritme heuristik yang disetel dengan cermat untuk memberikan hasil berkualitas tinggi.

Cakupan terbaik untuk deteksi cacat perangkat lunak melalui analisis statis memerlukan banyak alat dari beberapa vendor untuk digunakan secara bersamaan.

Selain akurasi, terdapat perbedaan besar dalam waktu eksekusi penganalisis statis. Pendekatan yang direkomendasikan adalah dengan menggunakan setidaknya satu analisis runtime yang efisien selama pembuatan perangkat lunak sehari-hari yang dijalankan oleh masing-masing pengembang dan membuang sisa alat yang tersedia ke eksekusi offline yang dapat memberi tahu tim pengembangan secara asinkron tentang kelemahan yang ditemukan. Karena beberapa kompiler menyertakan analisis statis program lengkap, tim pengembangan harus mempertimbangkan untuk mengaktifkan fitur ini sebagai opsi kompilasi untuk semua build.

[10] DiBona C, Ockman S, Stone M, editors. Voices from the Open Source Revolution. Sebastol, CA: O'Reilly; 1999.

Pusat Kesehatan Perangkat dan Radiologi (CDRH) Badan Pengawas Obat dan Makanan A.S. menggunakan penganalisis kode sumber statis sebagai alat forensik untuk membantu menemukan penyebab kegagalan perangkat medis.[11] Dalam beberapa kasus, beberapa penganalisis statis yang berbeda digunakan secara bersamaan. Demikian pula, Badan Keamanan Nasional AS (NSA) menggunakan beberapa penganalisis statis untuk membantu melakukan penilaian kerentanan keamanan pada perangkat lunak.

Organisasi pengembangan harus mempertimbangkan untuk mengevaluasi berbagai produk baik untuk efisiensi eksekusi maupun kualitas keluaran pada basis kode yang sama. Pilih kombinasi alat yang memberikan cakupan deteksi cacat yang sangat baik sekaligus menawarkan efisiensi waktu eksekusi yang cukup untuk memungkinkan pengembang menggunakan setidaknya satu dari alat tersebut pada setiap kompilasi.

Membuat Standar Pengkodean Tertanam Organisasi yang Disesuaikan

Sejauh ini dalam bab ini kita telah membahas pentingnya memiliki standar pengkodean berkualitas tinggi untuk membatasi kerentanan dan oleh karena itu meningkatkan keandalan dan keamanan perangkat lunak. Kita telah membahas bagaimana standar pengkodean memerlukan penerapan berbagai teknik kualitas kode yang telah terbukti, termasuk analisis kode sumber statis, pemeriksaan aturan MISRA, dan kontrol kompleksitas. Kami juga telah menekankan (berulang kali, dan kami belum selesai) betapa pentingnya mengotomatiskan sebanyak mungkin penerapan standar pengkodean ini; jika rantai alat sehari-hari pengembang selalu menerapkan standar pengkodean, maka teknik keamanan perangkat lunak akan berasimilasi ke dalam pikiran semua insinyur dan manajer. Sama seperti pegolf profesional yang mengandalkan memori otot untuk menciptakan ayunan berulang, pengembang perangkat lunak tertanam harus memikirkan keamanan perangkat lunak setiap hari untuk memastikan bahwa penyimpangan tidak terjadi.

Mempersiapkan Biaya Retroflt Satu Kali

Sayangnya, tugas menulis standar pengkodean yang memerlukan semua ide hebat ini dan mengaktifkan penganalisis perangkat lunak ini dalam rantai alat tidaklah sesederhana atau semudah kedengarannya.

> *Untuk organisasi yang baru mengenal penganalisis statis, MISRA, dan alat penegakan kualitas kode otomatis lainnya yang dibahas dalam bab ini, manajer dan tim pengembangan mereka perlu mempersiapkan diri menghadapi penerapan awal yang mungkin sulit.*

Misalnya, mengaktifkan penganalisis kode sumber statis untuk pertama kalinya dalam basis kode yang besar hampir pasti akan mengidentifikasi ratusan, bahkan ribuan, rangkaian kode yang bermasalah. Beberapa di antaranya tidak diragukan lagi merupakan kesalahan positif yang harus diatasi dengan memodifikasi kode untuk menenangkan penganalisis. Banyak kelemahan nyata yang akan ditemukan, dan organisasi akan terdorong oleh pemberantasan kelemahan tersebut, meskipun peninjauan dan koreksi terhadap kelemahan yang teridentifikasi mungkin memerlukan investasi sumber daya yang signifikan. Namun demikian, pengalaman menunjukkan bahwa setelah basis kode dibuat menjadi "*standar pengkodean yang bersih*", menjaganya tetap seperti itu akan menjadi rutinitas.

---

[11] "CDRH Software Forensics Lab: Applying Rocket Science to Device Analysis"; Chloe Taft, "Medical Devices Technology, " The Gray Sheet (October, 2007).

Mengizinkan Pengecualian yang Disetujui Manajemen untuk Mengurangi Regresi

Pengembang harus sangat berhati-hati saat memodifikasi perangkat lunak yang sudah ada sejak lama (dan mungkin sudah digunakan sejak lama). Beberapa penelitian menunjukkan bahwa penganalisis dan pemeriksa statis berpotensi membahayakan keamanan perangkat lunak dengan menimbulkan kelemahan baru saat memperbaiki masalah yang teridentifikasi. [12]Risiko ini sangat tinggi ketika melakukan retrofit pada standar pengkodean baru atau alat penganalisis baru ke basis kode.

Dalam beberapa kasus, mungkin lebih bijaksana untuk menonaktifkan pemeriksaan untuk perangkat lunak tertentu daripada mengambil risiko memodifikasinya. Misalnya, manajemen memutuskan untuk menyesuaikan aturan baru yang membatasi semua fungsi hingga nilai kompleksitas McCabe maksimum sebesar 20. Basis kode yang besar mungkin mencakup lusinan atau bahkan ratusan fungsi yang pada awalnya gagal memenuhi metrik ini. Beberapa subrutin yang mengganggu mungkin mudah untuk difaktorkan ulang. Yang lainnya mungkin sulit dan berisiko. Faktanya, beberapa pelanggar terburuk yang meminta penulisan ulang mungkin merupakan pelanggar yang salah untuk diubah, terutama jika mereka menyediakan fungsi yang sangat penting bagi keamanan. Jika perangkat lunak tersebut menyertakan algoritma kriptografi AES yang dikodekan dengan tangan yang telah dengan susah payah dikembangkan agar tahan terhadap serangan saluran samping dan telah melalui sertifikasi FIPS, mungkin pendekatan terbaik adalah membiarkan fungsi ini sebagai pengecualian yang terdokumentasi terhadap standar pengkodean.

> *Manajemen harus mengizinkan pengecualian terhadap aturan standar pengkodean untuk bagian tertentu dari perangkat lunak penting dimana risiko dan biaya retrofit jelas melebihi manfaatnya.*

Tentu saja, penilaian ini bersifat subyektif, dan manajemen harus berhati-hati agar tidak terjebak dalam penggunaan pengecualian sebagai cara untuk menghindari pengeluaran sumber daya.

Standar Bahasa Tidak Pernah Sempurna

Sebelumnya di bab ini, kita membahas bagaimana C++ Tertanam dibuat untuk menyediakan dialek subset yang lebih sederhana dan efisien dari standar bahasa C++ yang lengkap. Kemudian kami menjelaskan bagaimana beberapa organisasi mungkin menganggap bagian ini terlalu membatasi. Ini adalah contoh yang baik tentang perlunya menyesuaikan alat kualitas perangkat lunak otomatis, terutama yang didasarkan pada standar terbuka yang umum.

Standar MISRA memberikan contoh bagus lainnya. Keputusan manajemen yang kejam untuk mematuhi MISRA hampir pasti tidak praktis, menimbulkan banyak perselisihan internal, dan menimbulkan beban kontraproduktif di seluruh organisasi pembangunan. Misalnya, bayangkan salah satu proyek perangkat lunak tertanam Anda menggunakan alokasi memori dinamis secara berlebihan. Banyak pengembang C++ akan tertawa jika diminta untuk mengikuti standar yang melarang semua penggunaan operator new dan delete. Namun inilah yang disyaratkan oleh kesesuaian MISRA (Peraturan 18-4-1 MISRA C++:2008 dan aturan 20.4 MISRA C:2004). Meskipun masuk

[12] Boogerd C, Moonen L. Assessing the Value of Coding Standards: An Empirical Study. IEEE International Conference on Software Maintenance; October 2008. 277e86.

akal untuk menerapkan aturan ini untuk aplikasi hard real-time atau aplikasi penting keamanan tertentu yang memerlukan determinisme lengkap, banyak aplikasi tujuan umum lainnya yang dapat memperoleh manfaat dari alokasi memori dinamis. Manajemen, melalui konsultasi dengan tim pengembangannya, harus memeriksa setiap peraturan MISRA secara bergantian dan menentukan peraturan mana yang mewakili pilihan yang aman dan praktis. Meskipun disarankan untuk memasukkan kumpulan default ini ke dalam standar pengkodean, pemrogram harus siap untuk mengizinkan pengecualian khusus proyek serta pengecualian yang disetujui manajemen di setiap bagian kode. Kebanyakan alat pemeriksaan MISRA otomatis akan memungkinkan pemilihan aturan individual dalam standar.

Aturan lain yang mendapat kekecewaan di beberapa organisasi perangkat lunak tertanam adalah MISRA C:2004 20.9: “Perpustakaan input/output <stdio.h> tidak boleh digunakan dalam kode produksi.” Alasannya menyatakan bahwa sejumlah besar fungsi I/O standar C menyertakan perilaku yang tidak terdefinisi dan/atau ditentukan implementasi. Lebih jauh lagi, “Dalam dokumen ini diasumsikan bahwa [stream dan file I/O] biasanya tidak diperlukan dalam sistem tertanam.” Salah satu contoh nyata dari fungsi penting keamanan yang menjadi lebih mudah dengan I/O standar C adalah penggunaan sprintf untuk membuat pesan yang membentuk catatan log audit. Tanpa penggunaan beberapa fungsi I/O standar C yang cukup konservatif, pemrogram mungkin terpaksa membuat logging audit atau kode penting keamanan lainnya yang lebih kompleks (dan karena itu lebih rentan terhadap kesalahan dan kerentanan keamanan).

Beberapa aturan MISRA didasarkan pada gagasan masuk akal yang mungkin terlalu membatasi dan memerlukan penyesuaian. Contoh yang bagus adalah sebagai berikut:

Aturan 20.1: “Operator penugasan tidak boleh digunakan dalam ekspresi yang menghasilkan nilai Boolean.”

Fragmen kode berikut ditandai sebagai tidak sesuai:

```
if (a=b) { ...}
```

Apakah pemrogram benar-benar bermaksud menetapkan dan menguji nilai a secara bersamaan? Atau mungkin pemrogram bermaksud membandingkan a dengan b tetapi mengabaikan tanda sama kedua yang merupakan kelemahan umum. Ini adalah kasus yang jelas mewakili perangkat lunak yang dibangun dengan buruk yang mungkin mengindikasikan kesalahan pemrogram. Namun, urutan kode berikut juga tidak diizinkan oleh MISRA:

```
if((a-b)<-c) { . . .}
```

Konstruksi ini jauh lebih masuk akal. Dalam hal ini, penggunaan tugas oleh programmer lebih eksplisit, berbeda dari perbandingan, dan karenanya tidak mungkin terjadi kesalahan. Menggabungkan tugas dengan kondisional dengan cara ini sebenarnya dapat meningkatkan

keterbacaan dan pemeliharaan. Sayangnya, tidak ada standar bahasa yang mengizinkan contoh sebelumnya, namun tidak mengizinkan contoh sebelumnya. Beberapa alat pemeriksa kualitas perangkat lunak menyediakan banyak koleksi opsi khusus seperti ini. Pilihan terbaik bagi programmer adalah berbicara dengan penyedia alat dan melihat kemampuan penyesuaian seperti apa yang mungkin dilakukan di bidang ini.

Studi Kasus: Mode Standar Green Hills

Green Hills Software telah menerapkan pemeriksaan MISRA, analisis statis, pengendalian kompleksitas, dan banyak teknik penegakan kualitas perangkat lunak otomatis lainnya ke dalam alat pemeriksaan yang diterapkan secara internal, kompiler dan rantai alat Green Hills. Meskipun Green Hills telah menyediakan kemampuan ini kepada pelanggannya, standar pengkodean sebenarnya yang digunakan dalam organisasi pengembangan Green Hills mencakup serangkaian fasilitas otomatis yang dirancang dengan cermat yang mewakili sistem keamanan perangkat lunak yang komprehensif namun praktis. Sistem ini telah terbukti digunakan selama 30 tahun dan digunakan untuk membuat perangkat lunak yang bersertifikat tingkat keselamatan dan keamanan tertinggi.

Baru-baru ini, Green Hills memutuskan untuk membuat seperangkat aturan khusus yang tersedia bagi pengguna rantai alat Green Hills menggunakan opsi build khusus yang menentukan opsi pemeriksaan kode terbaru dan terbaik yang tersedia: –coding_standard¼ghstd<year>. Misalnya, –coding_standard¼ghsstd2010 adalah opsi untuk kumpulan aturan yang direkomendasikan dalam rilis rantai alat yang dibuat pada tahun 2010. Pada rilis berikutnya, pengguna memiliki opsi untuk mengaktifkan kumpulan aturan 2010 yang kompatibel atau mencoba versi yang lebih baru. Aturan ditentukan dalam file profil standar pengkodean khusus (.csp) yang dapat dikustomisasi oleh pengguna akhir. Organisasi pengembangan dapat mengambil file ghsstd2010.csp dan memodifikasinya untuk membuat yourstandard.csp lalu menambahkan opsi berikut untuk membangun perangkat lunak: –coding_standard¼standar Anda.csp. Sebagai contoh, nomor diagnostik 187 yang ditentukan dalam ghstd2010.csp mengaktifkan versi MISRA C:2003 Rule 20.1 yang telah dibahas sebelumnya dan lebih longgar dan disebut sebagai “penggunaan '=' di mana '==' mungkin dimaksudkan.”

Mari kita lihat contoh lain tentang bagaimana penyesuaian alat kualitas perangkat lunak otomatis dapat bermanfaat. Mari kita perhatikan aturan MISRA C:2004 berikut:

Aturan 15.3: Klausa terakhir dari pernyataan peralihan akan menjadi klausa default.
Alasannya adalah untuk memaksa programmer menangani semua kemungkinan nilai ekspresi switch. Namun, contoh berikut menunjukkan situasi di mana persyaratan ini berlebihan:

```
typedef enum { red, yello, blue } colors;
void carcolors(colors c)
{
     switch (c) {
     case red:
          printf("red\n"):
          break:
     case yellow:
          printf("yellow\n");
```

```
            break;
        case blue:
            printf("blue\n");
            break;
        }
    }
```

Kode sebelumnya tidak sesuai dengan MISRA dan akan ditandai sebagai kesalahan. Namun, karena ekspresi switch bertipe enumerasi yang rentang nilai yang mungkin tercakup sepenuhnya dalam klausa kasus pernyataan, maka penambahan klausa default tidak diperlukan:

```
typedef enum { red. yellow. blue } colors;
void carcolors(colors c)
{
    switch (c) {
    case red:
        printf("red\n");
        break;
    case yellow:
        printf("yellow\n");
        break:
    case blue:
        printf("blue\n"):
        break:
    default:
        printf("Unexpected color!\n"):
        break;
}
```

Faktanya, penambahan klausa default pada cuplikan sebelumnya akan menambahkan kode yang tidak dapat dijangkau (ironisnya, melanggar Aturan MISRA C:2004 14.1, yang melarang kode yang tidak dapat dijangkau) dan membuat fungsi tersebut membingungkan pembaca dan pengelola. Karena alasan ini, opsi ecoding_standard¼ghstd2010 sedikit melonggarkan aturan MISRA dan mengizinkan tidak adanya klausa default jika dan hanya jika semua nilai ekspresi yang mungkin tercakup dalam klausa kasus pernyataan switch lainnya. Contoh penyesuaian lainnya, yang juga melibatkan pernyataan switch, terkait dengan MISRA C:2004 lainnya:

Aturan 15.2: “Pernyataan pemutusan tanpa syarat akan mengakhiri setiap klausul peralihan yang tidak kosong.”

Alasannya adalah pemrogram terkadang lupa menambahkan pernyataan break, menyebabkan aliran kontrol jatuh ke klausa kasus berikutnya dengan konsekuensi yang tidak diinginkan. Meskipun Aturan 15.2 secara umum baik untuk diikuti, terkadang pernyataan switch lebih rapi jika klausa kasus dapat diabaikan untuk menghindari duplikasi kode. Namun, pengembang harus menunjukkan niatnya, misalnya, dengan komentar yang tepat sebelum

titik kegagalan. Untuk mengatasi situasi ini, opsi ecoding_standard¼ghstd2010 mengizinkan kesalahan jika dan hanya jika komentar FALLTHRU dalam bahasa sehari-hari disertakan. Menarik untuk dicatat bahwa bahasa C#, secara default, tidak mengizinkan kesalahan klausa kasus. Dan C#, seperti MISRA, tidak mengizinkan pengecualian terhadap aturan ini. Contoh berikut tidak sesuai dengan MISRA namun diizinkan oleh opsi standar pengkodean default Green Hills:

```
switch (i) {
     case 0:
     handle_zero();
          /* FALLTHRU */
case 1:
case 2:
case 3:
     process(i):
     audit("handle_input", i);
     post_buffer();
     break;
default:
     audit("Unexpected input", i);
     reset();
     break;
}
```

Ketika pengembang mempertimbangkan cara menulis kode ini dengan cara yang sesuai dengan MISRA, pendekatan yang paling jelas adalah menduplikasi kode dalam klausa 1e3 untuk klausa 0 atau membuat fungsi baru yang berisi kode untuk dibagikan oleh kasus 0e3 klausa. Namun, alternatif-alternatif ini menimbulkan kompleksitas yang tidak perlu. Komentar FALLTHRU memperjelas niat pengembang tanpa kode tambahan. C# memiliki solusi yang masuk akal menggunakan pernyataan goto yang secara sintaksis ilegal di C atau C++ dan tidak sesuai dengan MISRA:

```
switch (i) {
     case 0:
     handle_zero();
     goto case 1;
     case 1:
     case 2:
     case 3:
          process(i):
          audit("handle_input". i):
          post_buffer():
          break:
     default:
          audit("Unexpected input". i):
          reset();
          break;
}
```

Ketika pengembang mempertimbangkan cara menulis kode ini dengan cara yang sesuai dengan MISRA, pendekatan yang paling jelas adalah menduplikasi kode dalam klausa 1e3 untuk klausa 0 atau membuat fungsi baru yang berisi kode untuk dibagikan oleh kasus 0e3 klausa. Namun, alternatif-alternatif ini menimbulkan kompleksitas yang tidak perlu. Komentar FALLTHRU memperjelas niat pengembang tanpa kode tambahan. C# memiliki solusi yang masuk akal menggunakan pernyataan goto yang secara sintaksis ilegal di C atau C++ dan tidak sesuai dengan MISRA:

> *Sangat penting bagi manajemen untuk memutuskan arah yang akan diambil, berinvestasi pada biaya dan jadwal retrofit awal, dan kemudian melanjutkan jalur tersebut sampai standar pengkodean berkualitas tinggi tertanam kuat dalam sistem pengembangan dan diindoktrinasi ke seluruh organisasi teknik.*

Menerapkan proses baru seperti memasuki babak playoff dalam bola basket profesional. Pemenangnya sering kali bukanlah tim penembak terbaik, melainkan tim yang memiliki ketabahan untuk memainkan pertahanan yang kuat dan keras kepala untuk setiap permainan di setiap pertandingan dalam seri tersebut. Manajer harus keras kepala dan menuntut kepatuhan sampai proyek selesai.

Studi Kasus: Standar Pengkodean C Tertanam Netrino

Netrino adalah perusahaan konsultan sistem tertanam yang presidennya, Michael Barr, telah menulis standar pengkodean perangkat lunak tertanam untuk pemrogram C. [13]Untuk organisasi pengembangan yang ingin membuat standar pengkodean untuk pertama kalinya, Standar Pengkodean C Tertanam adalah bacaan yang bermanfaat dan titik awal yang sangat baik.

Standar pengkodean Netrino mendedikasikan sekitar setengah aturannya untuk masalah gaya, mulai dari penggunaan kurung kurawal, spasi, dan penyelarasan hingga konvensi penamaan dan gaya komentar. Meskipun pengembang yang baru mengenal standar pengkodean mungkin terkejut melihat penekanan pada konten yang bisa dibilang subjektif, pentingnya standar gaya tidak boleh diremehkan. Aturan-aturan ini mendorong keterbacaan dan pemeliharaan kode yang pada akhirnya berkontribusi terhadap keandalan dan keamanan perangkat lunak. Banyak aturan gaya perangkat lunak yang dapat diterapkan secara otomatis dengan penghias kode seperti Uncrustify, sehingga mengurangi beban kerja tinjauan sejawat.

Standar pengkodean Netrino juga berupaya untuk menunjukkan kapan aturan harus diterapkan dengan alat otomatis. Misalnya, satu aturan bagus, 4.3.c, “Setiap file sumber harus bebas dari file penyertaan yang tidak digunakan” disarankan untuk standar pengkodean apa pun. File penyertaan asing terkadang dapat mengganggu efisiensi kode (misalnya, referensi dalam file header dapat menyebabkan kode pustaka ditarik ke dalam tautan padahal tidak demikian), meningkatkan kompleksitas kode tanpa manfaat, dan membuat perangkat lunak lebih sulit dipelihara (misalnya , perubahan pada file header dapat mempengaruhi file sumber termasuk itu). Standar Netrino menunjukkan bahwa aturan ini dapat diterapkan secara otomatis dengan lint, penganalisis statis sumber terbuka yang ringan. Meskipun lint (dan saudara-saudaranya, seperti splint)[14] mungkin memerlukan investasi yang signifikan dalam hal konfigurasi untuk menghindari kesalahan positif, alat ini mungkin merupakan tambahan yang

[13] Barr M. Embedded C Coding Standard. Netrino; 2009.
[14] www.splint.org.

masuk akal untuk kotak alat pengembang, bersama dengan penganalisis statis yang lebih profesional yang berspesialisasi dalam menemukan bug perangkat lunak yang canggih sambil meminimalkan positif palsu.

Standar Netrino mengharuskan penggunaan dialek C99 jika tersedia. Pada tulisan ini, C99 adalah versi terbaru bahasa pemrograman C yang diratifikasi[15] dan menambahkan beberapa fitur yang diinginkan untuk kualitas kode, seperti kemampuan untuk menggunakan gaya C++ // komentar dan kemampuan untuk mendeklarasikan variabel iterasi loop dalam pernyataan loop awal. Namun, sebelum mengizinkan C99, pengembang harus mempertimbangkan semua tambahannya; fitur seperti array dengan panjang variabel mungkin sangat tidak diinginkan (menyebabkan penggunaan tumpukan runtime yang tinggi secara tidak terduga dan meningkatkan kemungkinan overflow). Mereka harus berkonsultasi dengan spesialis perangkat lunak tertanam seperti Netrino dan/atau pemasok alat pengembangan untuk melihat tingkat konfigurasi C99 yang mungkin dilakukan dalam rantai alat. Pada saat penulisan ini, revisi lain dari bahasa C, yang disebut C1X,[16] sedang dalam pengerjaan. Revisi terbaru C++, C++11, dirilis pada Agustus 2011.[17] Salah satu tambahan keandalan perangkat lunak yang diterima pada C++11 adalah static_assert, pernyataan waktu kompilasi. Meskipun hal ini dapat diimplementasikan pada versi C atau C++ yang lebih lama dengan menggunakan tipu muslihat typedef, penerimaan terhadap standar ini akan mendorong pengembang untuk memanfaatkannya dengan baik. Tidak seperti pernyataan runtime, yang menimbulkan overhead eksekusi (seringkali menyebabkan pernyataan tersebut dihilangkan dari produk akhir), pernyataan statis tidak berdampak pada eksekusi program. Pengembang C++ didorong untuk mengeksplorasi standar C++11 dan berkonsultasi dengan pemasok alat pengembangan mereka untuk ketersediaan produk yang sesuai.

Standar Netrino melarang lompatan tanpa syarat: goto, lanjutkan, dan penggunaan pernyataan break untuk keluar dari perulangan. Meskipun ini adalah aturan yang masuk akal dan tentunya merupakan ide bagus untuk kode baru, pengalaman menunjukkan bahwa menerapkan aturan ini ke basis kode yang besar bisa berbahaya. Beberapa basis kode hanya menggunakan lompatan ini secara berlebihan, terutama pernyataan goto. Aturan ini juga merupakan salah satu aturan yang kemungkinan besar akan mendapat argumen kuat mengenai penggunaan yang masuk akal dari pernyataan goto untuk memusatkan kode penanganan kesalahan yang, menurut desain, memerlukan penghentian pemrosesan tanpa syarat. Misalnya, fungsi driver perangkat Integrated Device Electronics (IDE) open source berikut (bagian dihilangkan agar singkatnya) yang menangani banyak kasus penulisan ke port I/O menggunakan tidak kurang dari 18 pernyataan goto, yang semuanya digunakan untuk mengakhiri diproses karena kondisi kesalahan fatal dan menjalankan beberapa pekerjaan pembersihan:

```
static void ide_ioport_write(void *opaque. uint32_t addr. uint32_t val)
{
    IDESLate *ide_if - opaque:
    IDESLaLe *s:
```

[15] ISO/IEC 9899-1999 (December 1999).
[16] CdThe C1X Charter. Document WG14N1250 (June 2007).
[17] ISO/IEC 14882:2011. Programming LanguagedC++ (August 2011).

```
int unit, n;
int 1ba48 - 0;
#ifdef DEBUG_IDE
printf("IDE: write addr-0x%x val-0x%02x\n", addr, val);
#endif
addr &= 7:
switch(addr) {
    case WIN_READ:
    case WIN_READ_ONCE:
    if (!s->bs)
        goto abort_cmd;
    ide_cmd_1ba48_Lransform(s. 1ba48):
    s->req_mb_sectors - 1:
    ide_scctor_read(s);
    break;
    case WIN_MULTREAD:
    if (!s->mult_sectors)
        goto abort_cmd:
    ide_cmd_lba48_transform(s, 1ba48):
    s->req_nb_sectors - s->mult_sectors;
    ide_sector_read(s);
    break;
    case CFA_WRITE_MULTI_WO_ERASE:
    if (!s->mult_sectors)
        goto abort_cmd:


    ide_cmd_lba48_transform(s, 1ba48):
    s->error - 0:
    s->status - SEEK_STAT | READY_STAT;
    s->req_mb_sectors = s->mult_sectors:
    n= s->nsector:
    if (n > s->req_nb_sectors)
    n- s->req_nb_sectors;
    ide_transfer_start(s, s->io_buffer, 512 * n,
    ide_sector_write);
    s->media_changed - 1;
    break:
    case WIN READDMA:
    case WIN READDMA_ONCE:
    if (!s->bs)
        goto abort_cmd;
    ide_cmd_lba48_transform(s, 1ba48);
    ide_sector_read_dma(s):
    break:
    case WIN WRITEDMA:
    case WIN WRITEDMA_ONCE:
    if (!s->bs)
        goto abort_cmd;
    ide_cmd_lba48_transform(s, 1ba48);
    ide_sector_write_dma(s):
```

```
s->media_changed = 1:
break:
case WIN_SETFEATURES:
if (!s->bs)
    goto abort_cmd;
switch(s->feature) {
case Oxcc: /* reverting to power-on defaults enable */
case 0x66: /* reverting to power-on defaults disable */
case 0x02: /* write cache enable */
case 0x82: /* write cache disable */
case Oxaa: /* read look-ahead enable */
case 0x55: /* read look-ahead disable */
case 0x05: /* set advanced power management mode */
case 0x85: /* disable advanced power management mode */
case 0x42: /* enable Automatic Acoustic Mode */
case 0xc2: /* disable Automatic Acoustic Mode */
s->status = READY_STAT | SEEK_STAT:
ide_set_irq(s):
break:
case 0x03: ( /* set transfer mode */
uint8_t val - s->nsector & 0x07;
switch (s->nsector >> 3) {
case 0x00: /* pio default */
case 0x01: /* pio mode */
put_lel6(s->identify_data + 63.0x07):
put_le16(s->identify_data + 88.0x3f):
break:
case 0x04: /* mdma mode */
put_le16(s->identify_data + 63,0x07 | (1 << (val + 8))):
put_lel6(s->identify_data + 88.0x3f):
break:
case 0x08: /* udma mode */
put_le16(s->identify_data + 63,0x07);
put_le16(s->identify_data + 88,0x3f | (1 << (val + 8))):
break:
default:
goto abort_cmd:
S->status - READY_STAT | SEEK_STAT;
ide_set_irq(s);
break:
default:
goto abort_cmd:
}
break;
case WIN_SRST:
if (!s->is_cdrom)
goto abort_cmd:
ide_set_signature(s):
s->status - 0x00: /* NOTE: READY is _not_set */
s->error - 0x01;
break;
```

```
case WIN_PACKETCMD:
if (!s->is_cdrom)
goto abort_cmd:
/* overlapping commands not supported */
if (s->feature & 0x02)
goto abort_cmd;
s->status - READY_STAT:
s->atapi_dma = s->feature & 1:
s->nsector = 1:
ide_transfer_start(s. s->io_buffer, ATAPI_PACKET_SIZE.
ide_atapi_cmd):
break:
/* CF-ATA commands */
case CFA REQ EXT ERROR CODE:
if (!s->is_cf)
goto abort_cmd:
s->error - 0x09; /* miscellancous error */
s->status - READY_STAT;
ide_set_irq(s):
break:
case CFA_ERASE_SECTORS:
case CFA WEAR_LEVEL:
if (!s->is_cf)
goto abort_cmd;
if (val =CFA_WEAR_LEVEL)
s->nsector = 0:
if (val = CFA ERASE_SECTORS)
S->media_changed - 1:
s->error - 0x00:
s->status - READY_STAT;
ide_set_irq(s);
break:
case CFA_TRANSLATE_SECTOR:
if (!s->is_cf)
goto abort_cmd:
s->error - 0x00:
s->status - READY_STAT:
memset(s->io_buffer. 0. 0x200):
s->io_buffer[0x00] = s->hcyl: /* Cyl MSB */
s->io_buffer[0x01] = s->Icyl: /* Cyl LSB */
s->io_buffer[0x02] - s->select: /* Head */
s->io_buffer[0x031 - s->sector: /* Sector */
s->io_buffer[0x04] - ide_get_sector(s) >> 16;/* LBA MSB */
s->io_buffer[0x05] - ide_get_sector(s) >> 8: /* LBA */
s->io_buffer[0x06] = ide_get_sector(s) >> 0: /* LBA LSB */
s->io_buffer[0x13] = 0x00: /* Erase flag */
s->io_buffer[0x18] = 0x00: /* Hot count */
s->io_buffer[0x19] - 0x00: /* Hot count */
s->io_buffer[0xla] - 0x01; /* Hot count */
ide_transfer_start(s, s->io_buffer, 0x200,
ide_transfer_stop);
```

```
        ide_set_irq(s):
        break:
        case CFA ACCESS METADATA_STORAGE:
        if (!s->is_cf)
        goto abort_cmd;
        switch (s->feature) {
        case 0x02: /* Inquiry Metadata Storage */
        ide_cfata_metadata_inquiry(s):
        break:
        case 0x03: /* Read Metadata Storage */
        ide_cfata_metadata_read(s):
        break:
        case 0x04: /* Write Metadata Storage */
        ide_cfata_metadata_write(s):
        break:
        default:
        goto abort_cmd;
        }
        ide_transfer_start(s. s->io_buffer. 0x200.
        ide_transfer_stop):
        s->status - 0x00: /* NOTE: READY is _not_set */
        ide_set_irq(s);
        break;
        case IBM_SENSE_CONDITION:
        if (!s->is_cf)
        goto abort_cmd:
        switch (s->feature) (
        case 0x01: /* sense temperature in device */
        S->nsector - 0x50: /* +20 C */
        break;
        default:
        goto abort_cmd:
        }
        s->status - READY_STAT:
        ide_set_irq(s);
        break;
        default:
        abort_cmd:
        ide_abort_command(s):
        ide_set_irq(s):
        break:
    }
```

Meskipun kode sebelumnya tidak akan dinominasikan untuk penghargaan apa pun, contoh ini menunjukkan penggunaan pernyataan goto yang efektif. Kami menyertakan fragmen perangkat lunak yang cukup besar ini dalam teks untuk menunjukkan bahwa penyesuaian beberapa aturan dapat menimbulkan masalah.

Ketika sebuah organisasi melakukan retrofit, pendekatan yang disarankan adalah dengan secara spekulatif mengaktifkan setiap aturan, pada gilirannya, dalam pemeriksaan lokal terhadap kode proyek untuk memastikan jumlah pelanggaran dan memperkirakan upaya perbaikannya. Manajemen tidak boleh meninggalkan aturan yang baik hanya karena diperlukan rekayasa yang signifikan untuk menerapkannya. Jika memungkinkan, organisasi harus mengizinkan pengecualian manajemen yang disetujui secara individual. Namun, ketika dihadapkan pada aturan kualitas kode yang secara konseptual masuk akal dan penerapannya sulit dilakukan, organisasi juga harus mempertimbangkan pendekatan kompromi. Dalam situasi goto sebelumnya, aturan kompromi dapat diterapkan:

```
The continue statement shall not be used.
The break shall not be used to terminate a loop.

Use of the goto statement is disallowed. with the exception of
safe, centralized error handling.
```

Standar Netrino mencakup persyaratan penamaan untuk fungsi yang digunakan sebagai titik masuk untuk thread dan rutin layanan interupsi (ISR). Aturan-aturan ini adalah contoh yang sangat baik tentang bagaimana standar pengkodean yang tertanam sering kali menyertakan fitur-fitur yang tidak ditemukan dalam standar pengkodean tujuan umum. Seperti disebutkan sebelumnya, meskipun menekankan penggunaan pemeriksaan otomatis, standar Netrino mencakup persentase aturan yang wajar yang mungkin memerlukan penegakan melalui tinjauan sejawat.

Manajemen harus secara hati-hati membatasi jumlah pengendalian standar pengkodean yang harus diterapkan secara manual; jika jumlah tersebut terlalu besar, tinjauan sejawat akan menjadi sangat tidak efisien atau penegakan aturan-aturan tersebut akan menjadi lemah dan pada akhirnya tidak relevan.

Analisis Kode Dinamis

Proses pengembangan yang aman harus menggunakan analisis kode dinamis selain analisis kode statis. Sebuah contoh sederhana menunjukkan kebutuhan ini. Kode berikut akan ditandai sebagai kesalahan oleh penganalisis kode sumber statis:

```
int *getval(void)
{
return 0;
}
     void foo(void)
{
     int *b - getval();
     *b – 0;
}
```

Pointer b diinisialisasi dengan nilai kembalian dari pemanggilan fungsi yang jelas mengembalikan pointer NULL. Kemudian b, penunjuk NULL, direferensikan. Namun, kode

serupa berikut ini mungkin tidak ditandai sebagai kesalahan oleh penganalisis kode sumber statis:

```
int fd;
int *getval (void)
{
     int *tmp;
     read(fd, &tmp, sizeof(tmp));
     return tmp;
}
void foo(void)
{
     int *b - getval();
     *b – 0;
}
```

Dalam contoh ini, b juga diinisialisasi dengan nilai kembalian dari pemanggilan fungsi. Namun, kode sumber tidak memberikan indikasi potensi nilai kembalian dari pemanggilan fungsi. Secara khusus, nilai kembalian dibaca dari file. Meskipun file tersebut mungkin berisi penunjuk yang tidak valid, yang menyebabkan program ini mogok, banyak penganalisis statis akan mengadopsi pendekatan konservatif (untuk meminimalkan kesalahan positif) dan tidak akan mengasumsikan sesuatu yang spesifik tentang data yang dibaca secara eksternal.

Analisis dinamis menggunakan instrumentasi kode atau lingkungan simulasi untuk melakukan pemeriksaan kode saat dijalankan. Misalnya, program yang diinstrumentasi akan melakukan pemeriksaan sebelum dereferensi b yang memvalidasi bahwa b bukan NULL. Atau simulator dapat memvalidasi semua referensi memori untuk memeriksa penulisan ke alamat 0.

Beberapa kompiler memiliki instrumentasi analisis kode dinamis yang tersedia sebagai opsi standar. Proses pengembangan harus mengharuskan pemeriksaan ini diaktifkan pada tahap pengembangan, pengujian, dan integrasi yang sesuai. Misalnya, kompiler Perangkat Lunak Green Hills memiliki opsi echeck¼memory, yang menyebabkan jumlah maksimum instrumentasi analisis dinamis untuk berbagai bentuk kesalahan memori, termasuk dereferensi penunjuk NULL. Kode yang diinstrumentasi melakukan pemeriksaan dan kemudian memanggil fungsi diagnostik, yang disediakan oleh perpustakaan yang secara otomatis ditautkan ke program saat menggunakan opsi ini, yang memberi tahu pengguna bahwa terjadi kesalahan serta jenis dan lokasi kesalahan dalam sumbernya. kode:

```
> gcc myfile.c -check-memory
> ./a.out
Nil pointer dereference on line 15 in file myfile.c
```

Ini adalah salah satu contoh di mana program kemungkinan besar akan mogok, sehingga membantu pengembang menemukan lokasi program, meskipun analisis dinamis tidak diaktifkan. Namun, banyak jenis kegagalan lainnya yang jauh lebih berbahaya, mengarah pada korupsi halus yang mungkin luput dari perhatian atau menyebabkan kegagalan hilir yang sangat sulit ditelusuri kembali ke akar permasalahannya.

Analisis dinamis mendeteksi kesalahan pada sumbernya, mengubah kesalahan yang sulit menjadi kesalahan yang sepele. Mari kita periksa beberapa contoh kontrol analisis kode dinamis yang harus digunakan pengembang selama pengembangan dan pengujian.

Buffer Melimpah

Ada banyak bentuk kesalahan buffer overflow, banyak di antaranya tidak dapat ditangkap oleh analisis statis karena jumlah data yang ditulis ke buffer tidak diketahui pada waktu pembuatan. Berikut ini adalah contoh sederhananya:

```
int an_array[10];
void a_func(int index)
{
    an_array[index] – 0;
}
```

Jika parameter yang diteruskan ke a_ func adalah nilai yang dibaca dari file atau antrian pesan oleh pemanggil ke a_ func, sebagian besar penganalisis statis akan mengabaikan referensi array ini secara konservatif. Namun, jika indeks ternyata bernilai lebih besar dari sembilan, penganalisis dinamis akan mendeteksi kesalahannya, seperti yang ditunjukkan di sini:

```
> gcc myfile.c -check-bounds
> ./a.out
Array index out of bounds on line 50 in file myfile.c
```

Batasan Penugasan

Bahasa pemrograman C dan C++ (khususnya C) mengalami kekurangan dalam hal keamanan tipe penegakan waktu kompilasi yang kuat seperti yang disediakan oleh bahasa seperti Ada dan C#. Namun, standar pengkodean kualitas serta penggunaan analisis statis dan dinamis dapat memberikan kompensasi yang wajar atas keterbatasan bahasa ini. Overflow bilangan bulat adalah salah satu risiko pengetikan yang lemah, seperti yang ditunjukkan dalam contoh berikut:

```
void assign(unsigned int p)
{
    static volatile unsigned short s;
    S = P;
{
void myfunc(void)
{
    assign(65536);
}
```

Fragmen kode ini sepenuhnya legal ANSI C; penugasan p ke s didefinisikan untuk memotong nilai p agar sesuai dengan tipe s. Dalam implementasi tipikal, bilangan bulat pendek yang tidak ditandatangani menempati 16 bit penyimpanan, memungkinkan nilai dalam rentang 0 hingga 65.535. Namun, dalam contoh, nilai di luar rentang ini diteruskan

sebagai parameter p, yang jelas merupakan kesalahan pemrograman. Namun kompiler standar tidak akan mengeluarkan peringatan bahkan pada urutan kode sebelumnya.

Analisis dinamis dapat mendeteksi penetapan nilai yang berada di luar rentang suatu tipe, meskipun nilai tersebut dibaca secara eksternal (misalnya, dari file). Perintah dan output build penganalisis untuk contoh sebelumnya mungkin terlihat seperti berikut:

```
> gcc myfile.c -check=assignbound
> ./a.out
Assignment out of bounds on line 57 in file myfile.c
```

Kasus Hilang

Kebanyakan bahasa pemrograman penting, seperti C, C++, C#, dan Java, memiliki kontrol pemilihan saklar/kasus yang setara. Sangat sah dalam bahasa-bahasa ini untuk memiliki pernyataan switch yang lengan kasusnya tidak mencakup semua nilai yang mungkin dari tipe ekspresi kontrol. Misalnya:

```
typedef enum { red. yellow, blue. green } colors:
void carcolors(colors c)
{
   switch (c) {
        case red;
            printf("red\n");
            break;
        case yellow;
            printf("yellow\n");
            break;
        case blue;
            printf("blue\n");
            break;
   }
}
```

Terlepas dari legalitas kode sebelumnya, beberapa kompiler dan penganalisis statis akan mengeluarkan diagnostik, mengeluhkan kurangnya kasus untuk menangani nilai hijau untuk kontrol sakelar c. Misalnya, compiler GCC open source akan mengeluarkan peringatan ketika meneruskan opsi -Wall yang mengaktifkan beberapa pemeriksaan di luar standar bahasa:

```
> gcc myfile.c -Wall
myfile.c: In function "carcolors':
myfile.c:64: warning: enumeration value 'blue' not handled in
switch
```

Beberapa programmer akan menyertakan case arm default sebagai kebiasaan untuk memastikan bahwa semua nilai yang mungkin dari variabel kontrol ditangani dan menghindari peringatan seperti itu. Namun, pendekatan ini tidak selalu merupakan ide bagus. Kasus yang mencakup semua dapat menyebabkan konsekuensi yang tidak diinginkan di mana

penanganan default tidak sesuai untuk semua input. Karena alasan ini, beberapa standar pengkodean dengan jaminan tinggi menghindari penggunaan kasus default jika memungkinkan dan malah mempromosikan penggunaan kasus eksplisit untuk semua nilai kontrol yang diharapkan.

Pada contoh sebelumnya, programmer mungkin mengetahui bahwa mobil hanya boleh berwarna merah, kuning, dan biru (tidak ada mobil berwarna hijau). Namun bagaimana jika suatu hari nanti mobil ramah lingkungan ditemukan? Akankah perangkat lunak diperbarui untuk mencerminkan kenyataan baru ini? Fungsi carcolor sebelumnya akan dikompilasi dan dijalankan, namun kurangnya penanganan ramah lingkungan dapat menimbulkan konsekuensi yang tidak diinginkan. Sekali lagi, dalam kasus seperti ini, analisis dinamis dapat digunakan sebagai mekanisme penegakan kualitas kode. Jika pernyataan switch memberikan nilai untuk variabel kontrolnya yang tidak cocok dengan kasus yang ada, maka penganalisis dinamis akan menghasilkan pengecualian runtime:

```
> gcc myfile.c -check-switch
> ./a.out
Case/switch index out of bounds on line 7 in file myfile.c
```

Tumpukan Luapan

Sebelumnya dalam bab ini, kita membahas aturan MISRA C 15.2, yang melarang rekursi untuk menghindari runtime stack overflow, dan bagaimana deteksi statis siklus dalam grafik panggilan program yang rumit bisa menjadi sulit karena pemanggilan fungsi tidak langsung. Selain itu, program tanpa rekursi juga dapat mengalami stack overflow hanya karena urutan pemanggilan fungsi yang panjang dan/atau penggunaan penyimpanan otomatis yang berlebihan.

Mendeteksi stack overflow sangat penting untuk keandalan dan keamanan sistem tertanam. Sistem yang tertanam sering kali memiliki keterbatasan memori, sehingga mengharuskan perancang sistem untuk secara hati-hati mengalokasikan dan meminimalkan penggunaan tumpukan untuk semua proses dan thread. Stack overflows dapat bermanifestasi dalam bentuk korupsi halus yang sulit dilacak selama pengembangan dan pengujian. Kerentanan yang berlebihan yang tidak terdeteksi selama pengembangan produk dapat menyebabkan program yang dijalankan terhenti. Penyerang yang menyadari kerentanan stack overflow dapat menggunakannya untuk menumbangkan eksekusi dengan berbagai cara. Misalnya, stack overflow yang dipicu oleh masukan yang dibuat ke satu thread dapat menimpa data di thread kedua, sehingga menyebabkan crash atau mengeksekusi malware.

Jika memungkinkan, alat analisis statis harus digunakan untuk memeriksa potensi kebutuhan memori tumpukan runtime terbesar untuk suatu program atau untuk semua thread dalam program multi-thread. Penyedia rantai alat Anda harus menyertakan alat untuk tujuan ini. Namun, karena dilema pemanggilan fungsi tidak langsung yang disebutkan di atas, potensi kebutuhan memori tumpukan runtime maksimum tidak selalu dapat dihitung secara statis.

Pada Bab 2, kita membahas bagaimana sistem operasi tertanam berkemampuan memori virtual dapat menggunakan halaman penjaga untuk mendeteksi stack overflow pada waktu proses. Bagi pengembang yang tidak menggunakan sistem operasi memori virtual, opsi

kedua untuk analisis dinamis stack overflow adalah melengkapi program dengan pemeriksaan overflow di prolog setiap pemanggilan fungsi. Fitur ini tersedia di beberapa kompiler dan mungkin tidak sesuai untuk aplikasi multi-thread. Membangun program yang melebihi tumpukannya akan menghasilkan kesalahan runtime yang sesuai, menghentikan eksekusi ketika penunjuk tumpukan pertama kali melampaui batas tumpukan runtime yang dialokasikan:

```
> gcc myfile.c -check=stack
> ./a.out
Stack overflow
```

Jika tidak ada opsi deteksi stack overflow dinamis yang terdokumentasi dalam rantai alat atau sistem operasi, pengembang harus mempertimbangkan metode mandiri berikut yang dapat bekerja dengan cukup baik. Sebagian besar sistem operasi memiliki kait untuk mengeksekusi panggilan fungsi yang ditentukan pengembang pada setiap saklar konteks sistem serta sarana membaca penunjuk tumpukan setiap thread dan lokasi segmen tumpukan yang dialokasikan pada thread. Fungsi peralihan konteks dapat dengan mudah membandingkan penunjuk tumpukan dari thread yang akan dieksekusi dengan batas tumpukan runtime thread. Pada kebanyakan komputer, tumpukan tumbuh ke bawah ke alamat yang lebih rendah, sehingga perbandingan yang menunjukkan penunjuk tumpukan di bawah bagian bawah segmen tumpukan yang dialokasikan akan menghasilkan alarm, catatan audit, dan sebagainya. Pembaca harus membaca dokumentasi sistem operasi untuk fitur kait pengalih konteks yang umum.

Kebocoran Memori

Salah satu manfaat keandalan utama yang dipuji oleh bahasa Java adalah menghindari alokasi memori heap dinamis yang dikendalikan pemrogram dengan menggunakan pengumpulan sampah otomatis. Namun, banyak aplikasi tertanam menggunakan alokasi memori dinamis dan mengalami kerentanan karena kesalahan manajemen memori yang tidak tepat. Banyak kesalahan seperti itu yang dapat dicegah melalui analisis kode dinamis. Kebocoran memori adalah salah satu kelas kesalahan manajemen memori. Kebocoran memori terjadi ketika suatu fungsi mengalokasikan memori tetapi tidak pernah melepaskannya. Jika fungsi tersebut dipanggil secara sporadis, maka hilangnya memori mungkin terjadi secara bertahap, lolos dari deteksi hingga sistem diterapkan di lapangan.

Lebih jauh lagi, jika penyerang menyadari adanya kebocoran fungsi, ia dapat memfokuskan perhatiannya pada penyebab fungsi tersebut dijalankan, menguras sumber daya memori sistem, dan menyebabkan kegagalan sistem. Penelusuran terhadap kerentanan kebocoran memori di Basis Data Kerentanan Nasional NIST mengungkap banyak kejadian pada produk komersial, termasuk peralatan keamanan. Misalnya, CVE-2010-2836 adalah kerentanan keamanan dengan tingkat keparahan tinggi yang baru-baru ini diidentifikasi dalam fitur jaringan pribadi virtual (VPN) SSL pada sistem operasi peralatan jaringan Cisco yang disebut IOS. Kerentanan ini memungkinkan penyerang jarak jauh menyebabkan penolakan layanan melalui kehabisan memori dengan memutus sesi SSL secara tidak benar.[18]

[18] National Institute of Standards and Technology National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2836.

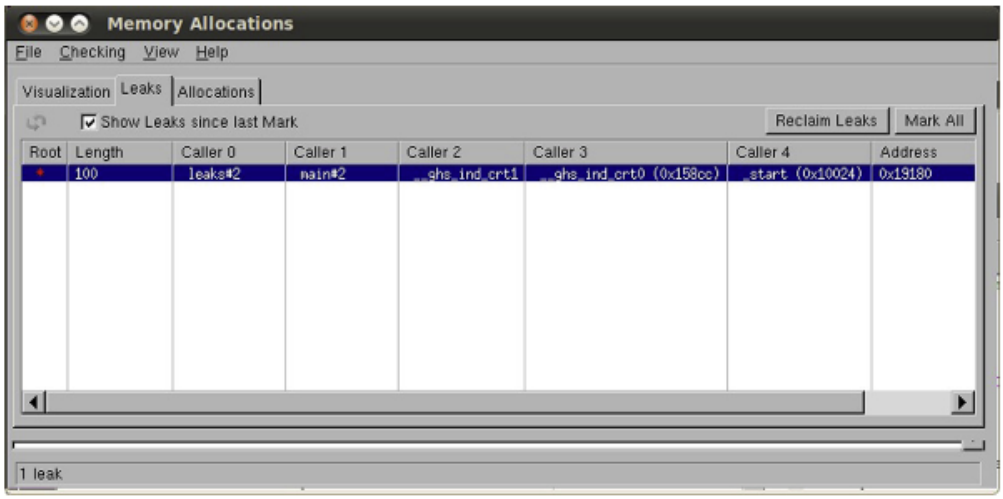
Deteksi kebocoran memori adalah bentuk analisis dinamis yang menghilangkan kerentanan kebocoran programmer. Deteksi kebocoran bekerja dengan membandingkan referensi penunjuk program dengan alokasi perpustakaan manajemen memori yang luar biasa. Referensi penunjuk suatu program mungkin berada di variabel data yang tersimpan di memori, penyimpanan tumpukan otomatis waktu proses, atau register CPU. Oleh karena itu, pendeteksi kebocoran memori biasanya ditawarkan sebagai fitur yang terintegrasi erat dari rantai alat pengembang (kompiler, pustaka runtime).

Kebocoran memori dapat terjadi kapan saja selama masa pakai program. Pustaka runtime dapat menjalankan algoritme deteksi kebocoran memorinya pada titik panggilan yang masuk akal (seperti saat memori dialokasikan atau dilepaskan). Selain itu, pengguna dapat menambahkan panggilan eksplisit ke algoritme deteksi kebocoran memori sebagai pemeriksaan kewarasan secara berkala atau pada titik tertentu dalam kode aplikasi. Deteksi kebocoran dapat dilakukan selama proses debug, selama pengujian, atau bahkan pada produk yang dikirim.

Idealnya, perpustakaan manajemen memori mampu merekam tumpukan panggilan eksekusi dalam database alokasinya. Ketika algoritme deteksi kebocoran mengidentifikasi kebocoran, tumpukan panggilan dapat dilaporkan ke pengembang, sehingga memudahkan untuk mengidentifikasi alokasi spesifik yang dibiarkan. Penganalisis kode sumber statis harus mendeteksi kesalahan kebocoran memori sederhana yang ditunjukkan berikutnya. Namun, seperti kasus-kasus lain yang dibahas dalam bagian ini, banyak bentuk kebocoran yang berada di luar jangkauan analisis statis dan memerlukan deteksi kebocoran dinamis.

```
void leak(void)
{
     char *buf =malloc(100):
     sprintf(buf, "some stuff\n"):
     printf(buf):
}
int main()
{
     leaks();
     _malloc_findleaks(); // call the leak detector
}
```

Dalam contoh sebelumnya, fungsi kebocoran mengalokasikan memori yang ditunjuk oleh variabel lokal dan tidak pernah membatalkan alokasi memori. Oleh karena itu, setelah kembali dari fungsinya, memori ini bocor. Panggilan ke pendeteksi kebocoran perpustakaan runtime akan melaporkan kebocoran tersebut:



Gambar 3.4 Deteksi kebocoran memori terintegrasi ke dalam lingkungan pengembangan perangkat lunak.

```
> gcc myfile.c -check-memory
> ./a.out
Unreferenced memory adr=0x18d40 allocated at 0x103f4 called from
0x1043c then 0x15f18 then 0x10028
```

Ketika terintegrasi dengan lingkungan pengembangan perangkat lunak, alamat tumpukan panggilan laporan deteksi kebocoran dipetakan ke lokasi kode sumber sebenarnya, sehingga memungkinkan pengembang untuk lebih mudah menemukan dan memahami sumber kebocoran (lihat Gambar 3.4). Untuk nama yang tepat dari antarmuka pemrograman aplikasi (API) deteksi kebocoran dan opsi waktu pembuatan yang digunakan untuk mengaktifkan deteksi kebocoran, pembaca harus berkonsultasi dengan pemasok rantai alat.

Kesalahan Alokasi Memori Dinamis Lainnya

Dengan kontrol terprogram atas alokasi dan dealokasi memori, ada lebih banyak cara bagi pengembang untuk melakukan kesalahan. Fungsi sederhana berikut menunjukkan beberapa contoh lagi:

```
void badalloc(void)
{
     char *buf=malloc(100);
     char localbuf[100];
     free(buf);
     free(localbuf);
     free(buf);
}
```

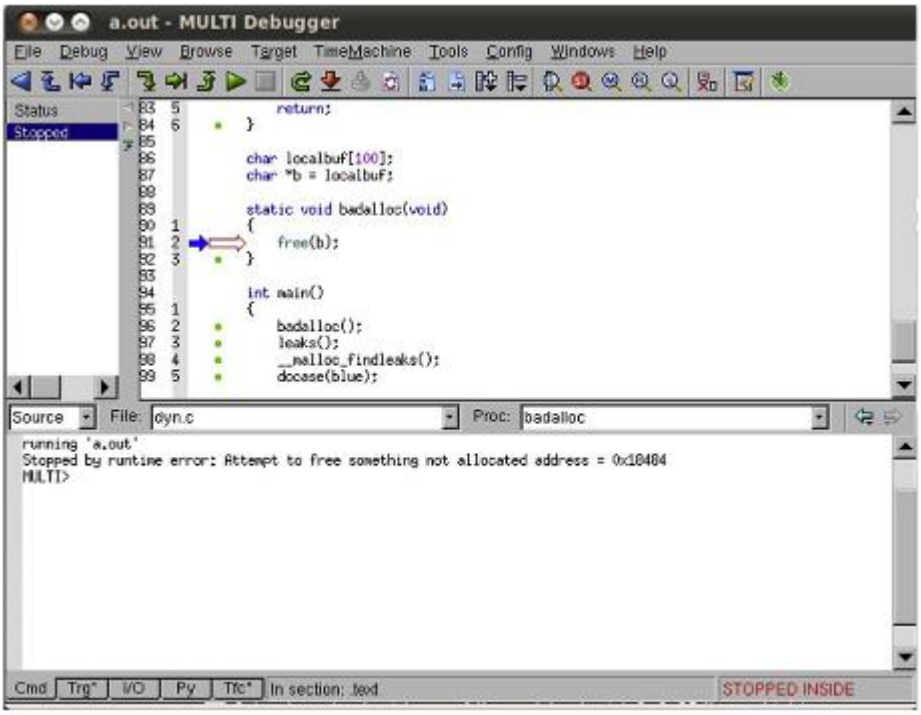
Panggilan pertama ke free(buf) baik-baik saja; itu merujuk pada buffer yang dialokasikan secara valid. Namun, panggilan kedua ke free(buf) tidak valid, karena buf telah dibatalkan alokasinya. Panggilan ke free(localbuf) juga tidak valid karena localbuf adalah buffer lokal, tidak dialokasikan menggunakan panggilan alokasi memori dinamis yang sesuai seperti malloc atau calloc. Kesalahan serupa di C++ terjadi pada operator baru dan hapus.

Sekali lagi, analisis statis dapat menemukan kesalahan dalam contoh ini, namun analisis dinamis akan menemukan kesalahan alokasi memori lain yang tidak dapat ditemukan oleh pemeriksaan statis. Misalnya, perubahan berikut akan membingungkan banyak penganalisis statis:

```
char localbuf[100];
char *b = localbuf;
void badalloc(void)
{
     free(b):
}
```

Karena variabel b sekarang didefinisikan secara global, penganalisa kode sumber statis mungkin berasumsi bahwa pengetahuannya lebih sedikit tentang variabel yang mungkin ditunjuk oleh b. Analisis dinamis mendeteksi deallokasi yang tidak valid selama eksekusi program:

```
> gcc myfile.c -check-memory
> ./a.out
Attempt to free something not allocated adr-0x18484
```



Gambar 3.5 Kesalahan analisis dinamis menghentikan program pada baris yang melanggar di debugger, sehingga memudahkan pengembang untuk menemukan dan memperbaiki kerentanan keamanan umum.

Jika analisis dinamis diintegrasikan ke dalam debugger, kegagalan sebelumnya akan lebih mudah dideteksi dan diperbaiki oleh pengembang. Seperti yang ditunjukkan pada Gambar 3.5, debugger secara otomatis dihentikan ketika kesalahan dealokasi memori terjadi,

mengarahkan pengembang ke baris kode yang melanggar. Sudah jelas bahwa manajer perangkat lunak harus mempertimbangkan kemampuan diagnostik kompiler dan rantai alat ketika memilih alat penting tersebut.

3.5.5 Pengujian dan Verifikasi Perangkat Lunak

Rangkaian pengujian yang komprehensif, termasuk pengujian fungsional, regresi, kinerja, dan cakupan, dikenal sebagai salah satu mekanisme terbaik untuk memastikan bahwa perangkat lunak dapat diandalkan dan aman.

Pengujian merupakan komponen penting dari banyak standar pengembangan dan dokumen panduan yang memiliki jaminan tinggi, seperti yang diumumkan secara resmi oleh Badan Pengawas Obat dan Makanan AS.[19] Di bidang pengujian fungsional, kami memiliki pengujian berbasis kesalahan, pengujian berbasis kesalahan, pengujian stres, pengujian kotak putih, dan pengujian kotak hitam.

Dua pendekatan pengujian hampir selalu diperlukan untuk memastikan keamanan. Pertama, semua perangkat lunak dalam komponen penting keamanan harus tercakup dalam beberapa bentuk pengujian fungsional. Cakupan diverifikasi menggunakan alat cakupan kode. Selain itu, semua perangkat lunak yang penting bagi keamanan harus dapat ditelusuri ke persyaratan komponen perangkat lunak. Perangkat lunak yang gagal ditelusuri kembali ke pengujian dan persyaratan lebih mungkin menimbulkan kerentanan keamanan laten.

Perubahan Kondisi/Cakupan Keputusan

Karena analisis cakupan kode sangat penting untuk keamanan, ada baiknya memeriksa berbagai tingkat pengujian cakupan yang dapat diterapkan pada perangkat lunak yang tertanam. Untuk membantu diskusi ini, kami mempertimbangkan persyaratan cakupan kode di lima tingkat jaminan yang ditentukan dalam standar yang digunakan oleh Administrasi Penerbangan Federal (FAA) AS untuk melakukan sertifikasi keselamatan pesawat komersial. Standar ini, yang diterbitkan oleh RTCA, diberi judul Pertimbangan Perangkat Lunak dalam Sertifikasi Sistem dan Peralatan Lintas Udara, yang biasa disebut DO-178B.[20] Faktanya, DO-178B adalah standar keamanan perangkat lunak yang paling umum digunakan dalam industri avionik di seluruh dunia.

Lima tingkat penjaminan DO-178B dalam meningkatkan tingkat kekritisannya adalah sebagai berikut:

- **Level E:** Kegagalan tidak berdampak pada keselamatan penerbangan.
- **Tingkat D:** Dampak kegagalan kecil, terlihat namun tidak kritis terhadap keselamatan penerbangan (misalnya ketidaknyamanan penumpang).
- **Tingkat C:** Dampak kegagalan sangat besar, berkaitan dengan keselamatan namun tidak parah (misalnya ketidaknyamanan penumpang namun bukan cedera).
- **Tingkat B:** Dampak kegagalan sangat parah (misalnya cedera penumpang).
- **Level A:** Dampak kegagalan bersifat bencana (misalnya kecelakaan pesawat terbang).

[19] General Principles of Software Validation. Final Guidance for Industry and FDA Staff. U.S. Food and Drug Administration, Center for Devices and Radiological Health; January 11, 2002.

[20] Document RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA; December 1992.

Persyaratan cakupan kode struktural yang sesuai dengan setiap tingkat jaminan ditunjukkan pada tabel berikut:

Tingkat Jaminan	Persyaratan Struktural
E	Tidak Ada
D	Tidak Ada
C	Penyataan
B	Keputusan
A	Kondisi/keputusan yang dimodifikasi

DO-178B Level C memerlukan cakupan pernyataan: menunjukkan bahwa setiap pernyataan program telah dijalankan setidaknya satu kali (dicakup) oleh program uji verifikasi. Cakupan pernyataan adalah apa yang disamakan oleh sebagian besar pengembang dengan istilah cakupan kode yang lebih umum.

Level B menambah cakupan pernyataan dengan cakupan keputusan, sebuah persyaratan bahwa setiap titik keputusan dalam program telah dilaksanakan dengan semua kemungkinan hasil. Misalnya, perbandingan cabang bersyarat yang berhasil (cabang diambil) dan gagal (cabang tidak diambil) setidaknya masing-masing satu kali. Terakhir, cakupan kondisi/keputusan yang dimodifikasi (MC/DC) menambah cakupan keputusan dengan bentuk cakupan kondisi khusus yang mana setiap kondisi dalam suatu keputusan harus terbukti mempunyai dampak independen terhadap hasil keputusan tersebut.

Kami menggunakan beberapa contoh kode sederhana untuk menggambarkan peningkatan ketelitian dan kualitas penegakan keamanan dari setiap pendekatan cakupan:

```
if (a||b||c) {
<code executed on true decision>
}
```

Cakupan pernyataan mengharuskan pernyataan if dieksekusi dan kode dalam blok if (dieksekusi pada keputusan yang benar) dieksekusi sepenuhnya. Karena tidak ada pernyataan yang berhubungan dengan keputusan yang salah, cakupan pernyataan tidak memerlukan kasus uji apa pun yang memaksa blok if untuk tidak dieksekusi.

Sebaliknya, cakupan keputusan memerlukan setidaknya satu pengujian untuk mengeksekusi jalur keputusan yang salah, meskipun tidak ada kode eksplisit yang terkait dengan jalur tersebut. Cakupan tambahan ini diinginkan dari sudut pandang keamanan karena ini menunjukkan bahwa pengembang telah mempertimbangkan dampak dari keputusan yang salah, yang mungkin mempunyai beberapa efek samping lainnya. Mari pertimbangkan contoh yang sedikit lebih mendetail ini:

```
uint32_t divisor = 0;
if (a || b || c) {
divisor = a | b | c;
}
result /= divisor;
```

Pernyataan pembagian akhir akan gagal (dibagi dengan nol) jika keputusan salah, namun pengujian cakupan pernyataan mungkin tidak pernah mengaktifkan jalur ini. Jika penyerang mampu mengendalikan keputusan (misalnya dengan mengendalikan nilai a, b, dan c), maka penyerang dapat menyebabkan penolakan layanan (program crash). Pengujian cakupan keputusan akan menunjukkan masalah ini sebelum dapat diterapkan. Cakupan kondisi mengharuskan setiap kondisi dalam keputusan diuji dengan nilai benar dan salah. Dua kasus uji berikut akan memaksa masing-masing dari tiga kondisi untuk mengambil nilai benar dan salah setidaknya satu kali: (a=1, b=1, c=1) dan (a=0, b=0, c =0).

Meskipun pengujian kondisi konstituen suatu keputusan mungkin tampak seperti peningkatan terhadap cakupan keputusan, cakupan kondisi bukanlah superset dari cakupan keputusan, seperti yang ditunjukkan dalam contoh ini:

```
if (a || !b) {
     <code executed on true decision>
} else {
     <code executed on false decision>
}
```

Dua kasus uji, (a=0, b=0) dan (a=1, b=1), memenuhi cakupan kondisi (kedua kondisi dieksekusi dengan input benar dan salah) tetapi mengabaikan jalur keputusan yang salah. Jelasnya, teknik cakupan keputusan dan kondisi yang digunakan secara bersamaan adalah hal yang diinginkan. Cakupan beberapa kondisi memerlukan semua kombinasi kondisi. Dengan kata lain, setiap baris tabel kebenaran suatu keputusan harus mempunyai kasus uji yang sesuai. Pada uji kasus sebelumnya dengan kondisi a, b, dan c, tabel kebenarannya adalah sebagai berikut:

a	b	c	hasil
F	F	F	F
F	F	T	T
F	T	F	T
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

Jadi, cakupan beberapa kondisi memerlukan 2n pengujian, dengan n adalah jumlah kondisi independen. Pendekatan ini dipandang tidak praktis; pengujian kondisi yang menyeluruh hanya akan memerlukan terlalu banyak kasus pengujian dan terlalu lama untuk dijalankan.

Bahasa dengan operator Boolean yang mengalami hubungan arus pendek (misalnya, C, C++, Java) mengurangi jumlah kasus pengujian yang diperlukan:

a	b	c	hasil
F	F	F	F
F	F	T	T
F	T	-	T
T	-	-	T

Namun demikian, ekspresi Boolean gabungan dapat menghasilkan ledakan yang tidak praktis dalam kasus uji di seluruh program yang realistis.

MC/DC adalah kompromi terpilih untuk sebagian besar standar keselamatan dan keamanan dengan jaminan tinggi. MC/DC mencakup cakupan keputusan dan kondisi. Namun, selain itu, MC/DC mensyaratkan bahwa setiap kondisi harus dibuktikan mempunyai dampak independen terhadap keputusan. Persyaratan kondisi yang dimodifikasi ini dipenuhi dengan memvariasikan satu kondisi sambil menjaga sisanya tetap konstan dan memverifikasi bahwa keputusannya berubah. Mari kita perhatikan contoh berikut:

```
if (a || b || c || d) {
<code executed on true decision>
}
```

MC/DC memerlukan kasus uji berikut (ditampilkan dalam bentuk tabel kebenaran):

a	b	c	d	hasil
F	F	F	F	F
T	F	F	F	T
F	T	F	F	T
F	F	T	F	T
F	F	F	T	T

Nilai yang dicetak miring di bawah setiap kondisi adalah kasus uji yang mencakup masukan benar dan salah dan menghasilkan hasil benar dan salah jika semua masukan kondisi lainnya dijaga konstan. Untuk kondisi tak berpasangan, MC/DC memerlukan kasus uji N+1, dimana N adalah jumlah kondisi Boolean. Pertumbuhan linier dalam kasus pengujian ini menjadikan MC/DC praktis untuk diterapkan, dan efektivitasnya dalam menemukan kesenjangan pengujian serta kelemahan desain telah diakui dan didokumentasikan dengan baik.[21]

Kami sangat menyarankan penggunaan pengujian cakupan MC/DC untuk komponen paling penting dari sistem tertanam, misalnya kernel sistem operasi, protokol keamanan jaringan, dan komponen kriptografi.

MC/DC mungkin berlebihan untuk beberapa aplikasi yang tidak memiliki hak istimewa. Kebalikannya adalah perancang sistem tidak boleh berasumsi bahwa cakupan MC/DC

[21] Dupuy A, Leveson N. An Empirical Evaluation of the MC/DC Coverage Criterion on the HETE-2 Satellite Software. Proceedings of the Digital Aviation Systems Conference (DASC); October 2000.

menyiratkan pengujian yang sempurna. Keterbatasan (dan perbaikan) definisi MC/DC tradisional telah dilaporkan.[22]

Ada banyak aspek lain dari suatu program untuk menguji cakupan selain alur eksekusi kode. Misalnya, jika program memiliki tipe enumerasi dengan lima kemungkinan nilai, masuk akal untuk memvalidasi bahwa kelima nilai tersebut setidaknya digunakan di suatu tempat dalam program.

Masalah lain dengan pengujian cakupan kode adalah hilangnya fidelitas saat menerjemahkan dari kode sumber ke kode mesin. Dalam beberapa kasus, lebih baik pengujian cakupan kode dilakukan pada kode mesin. Dengan melakukan hal ini, kami meningkatkan jaminan bahwa kode berbahaya tidak diinstrumentasikan sebagai bagian dari proses pembangunan (lihat studi kasus Thompson Hack di awal bab ini). Faktanya, cakupan kode mesin diwajibkan oleh beberapa sertifikasi keamanan dan keselamatan dengan jaminan tinggi. Mari kita periksa fungsi sederhana berikut:

```
int foo(int a, int b, int *arr, int n)
{
     int i;
     for (i = 0; i < n; i++) {
          arr[i] +=a / b;
     }
     return i;
}
```

Badan perulangan berisi operasi pembagian yang pembilang dan penyebutnya merupakan invarian perulangan. Kompiler ingin menghilangkan kesenjangan ini di luar loop untuk meningkatkan kinerja:

```
int foo(int a. int b. int *arr. int n)
{
int i;
int tmp = a / b;
for (i - 0; i < n: i++) {
     arr[i] += tmp;
;
return i;
}
```

Namun, pengoptimalan ini tidak diperbolehkan karena mengubah semantik fungsi. Dalam versi pra-optimasi, pembagian mungkin tidak akan pernah dieksekusi (jika loop itu sendiri tidak pernah dieksekusi, yaitu argumen n adalah nol). Dalam versi yang dioptimalkan, pembagian tersebut dijalankan tanpa syarat. Jika argumen b adalah nol, maka optimasi secara teoritis dapat menyebabkan crash program yang mungkin tidak akan terjadi. Tapi kompilernya

[22] Sergiy VA, Bowen JP. From MC/DC to RC/DC: Formalization and Analysis of Control-Flow Testing Criteria. Formal Aspects of Computing 2002;18(1).

pintar! Kebanyakan kompiler akan menghilangkan pembagian tersebut tetapi memperkenalkan perlindungan terhadap pembagian nol:

```
int foo(int a. int b. int *arr. int n)
{
int i, tmp;
if (b !- 0)
tmp = a / b;
for (i = 0; i < n; i++) (
arr[i] += tmp;
}
return i;
}
```

Kode sebelumnya menunjukkan optimasi kompiler yang divisualisasikan sebagai perubahan kode sumber. Tentu saja, kompiler melakukan optimasi ini saat ia menghasilkan kode mesin. Oleh karena itu, kode mesin berisi keputusan baru, b != 0, yang tidak ada dalam kode sumber. Alat cakupan kode yang beroperasi pada kode sumber saja akan gagal untuk mencakup keputusan tambahan ini.

Jika alat cakupan kode mampu memberikan cakupan keputusan tetapi tidak mampu MC/DC, sumber dapat dimodifikasi untuk menghilangkan kondisi majemuk. Dengan seluruh kondisi singleton, MC/DC dikurangi menjadi cakupan keputusan atau cabang:

```
if(a || b) {...}
```

dimodifikasi menjadi

```
if (a)
if (b) { ...}
```

Meskipun pendekatan ini mungkin tampak berat, hal ini dapat diterima karena jumlah kode kritis yang memerlukan pengujian level MC/DC seringkali relatif kecil. Kurangnya cakupan pengujian yang lengkap hampir selalu menunjukkan kurangnya validasi penting lainnya (misalnya pengujian fungsional), kelemahan desain, atau sekadar kode laten yang tidak diketahui dan berpotensi berdampak pada keamanan.

Hal ini dibiarkan sebagai latihan bagi pembaca untuk mengeksplorasi berbagai jenis pengujian lainnya dan keuntungan relatifnya. Selain contoh penting pengujian cakupan, panduan kami lebih berkaitan dengan integrasi metodologi pengujian ke dalam proses pengembangan untuk memaksimalkan nilainya.

Organisasi yang tidak mengikuti proses pengembangan yang ketat sering kali melakukan pengujian ad hoc yang sering kali hanya dilakukan setelah sebagian besar perangkat lunak telah dibuat. Organisasi yang mengikuti proses yang ketat sering kali memfokuskan pengujian selama proses rilis, sekali lagi setelah sebagian besar perangkat lunak telah ditulis.

Jika sistem pengujian dijalankan hanya berdasarkan permintaan, kadang-kadang, atau hanya selama proses rilis, maka kesalahan yang dapat dideteksi oleh sistem pengujian cenderung tidak diketahui dalam jangka waktu yang lama. Ketika cacat ditemukan, pengembang akan mengalami kesulitan untuk memperbaikinya dibandingkan jika cacat tersebut ditemukan pada hari sebelumnya. Dalam beberapa kasus, pengembang mungkin telah pindah ke proyek lain, jika bukan perusahaan lain, meninggalkan orang lain untuk mencoba mempelajari kode dan memperbaiki kekurangannya. Memperbaiki kelemahan yang ditemukan oleh sistem pengujian harus diprioritaskan lebih tinggi dari apa pun selain masalah dukungan pelanggan darurat; menjaga sistem tetap berjalan dengan bersih setiap saat menjamin bahwa kegagalan sistem pengujian hampir selalu merupakan kegagalan baru yang belum diperiksa oleh orang lain dan memerlukan perhatian segera.

Sistem pengujian harus dijalankan pada produk yang dikembangkan secara aktif serta produk yang sedang dikirimkan.

Poin kunci ini merupakan akibat wajar dari poin kunci sebelumnya namun cukup penting untuk ditekankan: ketika sistem pengujian digunakan selama proses pengembangan, pengembang dipaksa untuk menjaga produk dalam kondisi berfungsi setiap saat. Proyek perangkat lunak yang beralih ke pengujian ketat hanya setelah pembekuan kode akan mengalami fase pengujian yang secara keseluruhan berlangsung lebih lama karena pengembang harus bergulat dengan masalah yang dimasukkan selama berbulan-bulan waktu pengembangan. Ketika suatu produk selalu berfungsi, pembekuan kode mengarah langsung ke pengujian jaminan kualitas akhir, sehingga menghemat waktu ke pasar. Jika pengembang tidak dapat mengembangkan kode dengan cara yang mencegah kegagalan produk, maka cabang swasta dapat digunakan selama tidak dibiarkan hidup terlalu lama; mengintegrasikan cabang kode lama yang sudah jauh dari batangnya sering kali menyebabkan konflik tak terduga yang memengaruhi efisiensi seluruh tim pengembangan.

Sistem pengujian harus dapat menguji proyek perangkat lunak secara efektif dalam waktu kurang dari satu malam.

Sistem pengujian yang memerlukan waktu terlalu lama untuk dijalankan cenderung kurang dimanfaatkan atau diabaikan sepenuhnya. Pengembang harus dapat dengan cepat memvalidasi perubahan dalam semalam sebelum menerapkannya pada proyek. Selain itu, pengujian otomatis yang berjalan 24x7 pada kumpulan komputasi pengujian khusus dapat mendeteksi kekurangan dengan sangat cepat sehingga dapat diperbaiki sementara pemahaman tentang kode yang baru ditambahkan masih segar dalam ingatan pengembang. Masuk akal untuk melakukan lebih banyak tes yang dapat dijalankan dalam satu malam; namun, proses yang lebih lama harus bersaing dengan prioritas sumber daya komputasi yang lebih rendah atau dijalankan hanya berdasarkan permintaan atau pada interval yang lebih lama selama proses pengembangan. Uji coba setiap malam harus cukup baik untuk mendeteksi hampir semua kekurangan yang ditambahkan selama pengembangan.

Menentukan kapan uji coba berhasil atau gagal adalah hal yang mudah; tes yang gagal seharusnya mudah untuk direproduksi.

Pengujian harus ditulis sedemikian rupa sehingga keluaran dihasilkan hanya ketika kesalahan terdeteksi. Tes yang bersih adalah tes tanpa hasil apa pun. Paling buruk, outputnya harus kurang dari satu halaman. Seringkali, sistem pengujian menghasilkan keluaran yang banyak, sehingga menyulitkan pengembang untuk memastikan dengan cepat status pengujian yang dijalankan. Hasil tes yang sulit dievaluasi dengan cepat cenderung tidak efektif dan diabaikan.

Ketika pengujian gagal, keadaan sebenarnya dari sistem perangkat lunak dan masukan atau proses apa pun yang harus digunakan untuk mereproduksi kesalahan yang ditemukan harus ditampilkan dengan jelas dalam keluaran pengujian. Jika pengembang tidak mampu mereproduksi kegagalan pengujian secara efisien, sistem pengujian akan cenderung diabaikan. Reproduksibilitas adalah kunci untuk memaksimalkan tingkat di mana pengembang dapat memperbaiki kelemahan yang ditemukan oleh sistem pengujian dan membawa produk perangkat lunak yang andal ke pasar dengan lebih cepat.

3.5.6 Efisiensi Proses Pembangunan

Meskipun terdapat bukti signifikan bahwa mengikuti proses manajemen kualitas perangkat lunak yang terstruktur dan komprehensif, keamanan meningkat dibandingkan dengan penggunaan proses yang tidak terstruktur, metodologi yang kaku ini sering kali menyebabkan hilangnya efisiensi, tertundanya waktu pemasaran, dan frustrasi dalam kehidupan sehari-hari pengembang dan pengembang perangkat lunak. manajer. Kuncinya adalah menerapkan teknik yang sesuai dengan persyaratan standar kualitas dengan jaminan tinggi namun dirancang dan terbukti meningkatkan efisiensi pengembangan perangkat lunak. Tujuannya adalah untuk memaksimalkan keandalan terhadap rasio biaya produksi perangkat lunak.

Ketika perangkat lunak berubah dengan cepat, efisiensi proses pembangunan menjadi komponen penting dari efisiensi pengembang. Proyek perangkat lunak yang kompleks sering kali ditandai dengan proses pembangunan yang kompleks, di mana perangkat lunak tidak hanya membutuhkan waktu lama untuk dikompilasi ulang dari awal, namun juga mungkin memerlukan banyak kompilasi ulang untuk menjalankan konfigurasi produksi yang berbeda. Misalnya, sistem perangkat lunak mungkin memiliki versi “produksi”, di mana optimalisasi kompiler diaktifkan sepenuhnya dan perangkat lunak dikonfigurasi untuk kecepatan dan keandalan maksimum; build “*debug*”, yang sistemnya mengaktifkan informasi debugging sehingga pengembang dapat dengan mudah men-debug perangkat lunak; dan build yang “*dicentang*”, di mana sistem mengaktifkan pemeriksaan kewarasan tambahan yang mungkin secara drastis mengurangi kinerja namun meningkatkan kemungkinan menemukan masalah yang tidak biasa seperti kegagalan perangkat keras RAM.

Selama pengembangan, mungkin tidak praktis bagi pengembang untuk membuat semua konfigurasi untuk menguji perubahan. Oleh karena itu, sistem autobuild harus digunakan. Ketika perubahan dilakukan pada sistem manajemen konfigurasi, server build khusus memperbarui checkout lokal perangkat lunak dan membangun kembali semua konfigurasi. Ketika pembangunan gagal, pembuat otomatis mengirimkan email otomatis ke manajer komponen dari komponen yang terpengaruh. Email juga dikirim ke orang yang bertanggung jawab atas sistem pembangunan. Sistem autobuild memastikan bahwa

perubahan yang salah yang menyebabkan kegagalan build segera terdeteksi sebelum berdampak pada pengembang lain.

Gunakan sistem autobuild untuk mendeteksi dengan cepat perubahan yang merusak build sistem.

Dengan PC dan server modern, pengembang menikmati pembangunan yang lebih cepat dari sebelumnya. Namun, seiring dengan meningkatnya kompleksitas perangkat lunak, waktu pembuatan tetap menjadi faktor penting yang berkaitan dengan efisiensi pengembang. Namun ada beberapa metode untuk mengurangi pengaruh waktu pembangunan terhadap produktivitas pengembang.

Selalu pastikan pengembang memiliki setidaknya dua proyek pengembangan untuk dikerjakan setiap saat.

Ketika perubahan dilakukan dan pembangunan yang panjang dimulai, tidak ada alasan bagi pengembang untuk menunggu pembangunan tersebut selesai. Pengembang harus selalu memiliki proyek sekunder untuk dikerjakan selama jeda yang tidak dapat dihindari, seperti yang disebabkan oleh menunggu pembangunan, tinjauan sejawat, atau banyak alasan lain mengapa proyek latar depan tertunda. Memastikan bahwa pengembang memiliki banyak proyek untuk dikerjakan setiap saat, pada akhirnya, merupakan tanggung jawab manajemen pengembang. Namun, pengembang secara alami menganggap waktu henti menunggu penyelesaian pembangunan sebagai mode operasi normal di mana pengembang tidak punya pilihan selain beristirahat. Oleh karena itu, penting untuk mengajari pengembang untuk secara proaktif meminta lebih banyak pekerjaan jika ia mendapati dirinya diblokir di semua lini.

Solusi teknis lainnya terhadap masalah pembangunan yang menunggu adalah dengan mengurangi waktu pembangunan dengan memanfaatkan kekuatan penuh dari tenaga komputer perusahaan. Situs pengembangan pada umumnya mungkin mempekerjakan banyak pengembang, mengerjakan proyek perangkat lunak yang sama atau berbeda. Mungkin ada lusinan, bahkan ratusan, PC di meja para pengembang ini. Sayangnya, PC ini biasanya menghabiskan sebagian besar waktunya menganggur.

Gunakan build terdistribusi untuk memaksimalkan pemanfaatan komputer dan meningkatkan efisiensi pengembang.

Beberapa vendor kompiler menyediakan kemampuan build terdistribusi. Minimal, manajemen harus memanfaatkan paralelisasi lokal dari operasi pembangunan pada komputer host multi-inti tunggal. Untuk benar-benar melakukan penskalaan, sistem pembangunan paralel harus mampu menginterogasi sumber daya komputasi di seluruh situs, menemukan mesin yang kurang dimanfaatkan, dan memigrasikan file yang diperlukan untuk menyelesaikan pembangunan terdistribusi ke sumber daya yang tersedia ini. Idealnya, sistem build terdistribusi tidak memerlukan konfigurasi yang signifikan. Satu-satunya informasi konfigurasi yang diperlukan untuk alat ini adalah pengetahuan tentang mesin mana di jaringan yang harus menjadi kandidat untuk berbagi beban kerja. Pembangunan yang terdistribusi sangat mengurangi waktu pembangunan, sehingga memperpendek siklus pembangunan-edit-debug yang sangat penting bagi produktivitas pengembang setiap hari. Efek samping menguntungkan lainnya dari pembangunan terdistribusi adalah potensi mengurangi biaya modal pengembang dengan memanfaatkan sumber daya komputasi di seluruh situs dengan lebih baik.

3.6 VALIDASI AHLI INDEPENDEN

Setelah diskusi panjang sebelumnya mengenai proses pengembangan yang aman, kini kita kembali ke anak tangga kelima dalam PHASE: validasi pakar independen.

Bukti jaminan intrinsik harus dievaluasi dan dikonfirmasi oleh para ahli independen.

Konsumen yang membaca tentang klaim vendor tentang "dapat disertifikasi" harus menafsirkan hiperbola tersebut sebagai "tidak bersertifikat." Misalnya, untuk mencapai sertifikasi Tingkat Jaminan yang Dievaluasi di atas lima di Amerika Serikat, suatu produk tidak hanya harus memiliki semua dokumentasi desain, pengujian, dan metode formal yang dievaluasi dengan cermat oleh laboratorium pengujian Kriteria Umum yang terakreditasi, namun juga harus menjalani pengujian penetrasi. oleh para ahli pengujian penetrasi NSA yang memiliki akses penuh terhadap kode sumber dan sumber daya tak terbatas yang dapat digunakan untuk merancang vektor serangan.

Dalam domain keselamatan pesawat AS, FAA harus mensertifikasi perangkat elektronik penting pesawat (dan perangkat lunak penyusunnya) terhadap standar RTCA/DO-178B. Sebelumnya dalam bab ini, kami telah menjelaskan lima tingkat keselamatan, A hingga E, dengan Tingkat A sebagai yang tertinggi, dimana kegagalan dalam sistem elektronik dapat berakibat fatal. Seperti disebutkan sebelumnya, DO-178B Level A menetapkan proses penjaminan yang ketat, dengan banyak kesamaan dengan persyaratan penjaminan tinggi Kriteria Umum. Dalam hal ini, evaluator ahli independen adalah perwakilan teknik yang ditunjuk (DER) dari FAA.

Badan pengatur sertifikasi keselamatan pengendalian industri, otomasi, dan sistem otomotif terkadang menggunakan standar IEC 61508 secara internasional. IEC 61508 juga mulai diterima di industri lain. Interpretasi khusus aplikasi perkeretaapian dari IEC 61508 diwujudkan dalam CENELEC EN 50128, "Perangkat lunak untuk sistem kendali dan perlindungan kereta api," dan EN 50129, "Sistem elektronik terkait keselamatan untuk persinyalan." Standar-standar ini mencakup seluruh siklus hidup pengembangan perangkat lunak dan memberikan penekanan kuat pada analisis dan mitigasi bahaya keselamatan, penegakan sistem manajemen mutu yang ketat, serta verifikasi dan validasi menyeluruh. TU¨ V, yang berbasis di Jerman, adalah otoritas sertifikasi yang diakui secara internasional untuk IEC 61508 dan standar keselamatan terkait. Exida adalah perusahaan lain yang melakukan sertifikasi IEC 61508 dan terkait.

Interpretasi khusus otomotif dari IEC 61508 diwujudkan dalam ISO 26262, yang penerapannya dalam industri otomotif masih baru. Industri kereta api dan industri lain yang menggunakan IEC 61508 harus melalui persyaratan pengembangan dan pengujian yang ketat. Dokumentasi mencakup kemampuan penelusuran hingga persyaratan keselamatan penting untuk sistem yang sedang dikembangkan dan antarmukanya ke produk akhir. Otoritas sertifikasi mengaudit seluruh proses pengembangan sistem, termasuk menyaksikan pengujian utama dan memeriksa ketertelusuran atribut sistem. Tergantung pada tingkat integritas keselamatan, arsitektur sistem bisa sederhana atau semakin kompleks. Pemasok yang dapat memulai dengan sistem operasi yang sudah bersertifikat CENELEC dapat menghemat uang dengan tidak perlu melakukan pengujian sistem (dan mungkin bahkan pengujian unit,

tergantung pada fungsi yang diperlukan) sistem operasi dan komponen runtime terkait ketika perangkat lunak itu sendiri merupakan bagian dari sistem operasi. kasus keselamatan.

Perangkat lunak yang memenuhi kelima prinsip PHASE (mudah untuk melihat betapa kekurangan salah satu prinsip tersebut bisa berakibat fatal) dapat dipercaya untuk mengelola dan melindungi aset bernilai tinggi yang digunakan dalam sistem kritis keamanan. PHASE tidak berusaha mendefinisikan secara ketat "proses pengembangan yang aman" atau secara spesifik bagaimana prinsip-prinsip yang paling tidak memiliki hak istimewa berlaku pada komponen atau sistem perangkat lunak tertentu. Penafsirannya akan bervariasi tergantung pada situasi dan tingkat keamanan yang diperlukan. Penggabungan prinsip-prinsip ini juga dapat mengurangi biaya pengembangan dalam jangka panjang, dengan menggabungkan praktik pengembangan perangkat lunak yang berfokus pada keamanan. Selain itu, PHASE dapat diterapkan secara selektif dan kreatif, untuk meningkatkan keamanan dan keandalan tanpa mengorbankan penggunaan kembali perangkat lunak lama.

Proses pengembangan dengan jaminan tinggi, seperti yang didorong oleh standar kualitas ISO/IEC serta badan pengatur yang mengatur penggunaan perangkat lunak dalam sistem keselamatan dan keamanan, umumnya diyakini menghasilkan perangkat lunak dengan keandalan yang lebih tinggi. Namun, organisasi yang mengikuti proses ini secara menyeluruh kemungkinan besar akan memasuki pasar lebih lambat dibandingkan organisasi yang mengembangkan produk yang sama tanpa harus mengikuti standar kualitas ini. Dalam bab ini, kami telah menyajikan serangkaian rekomendasi panduan, bagian dari proses yang telah terbukti digunakan selama 30 tahun di organisasi pengembangan perangkat lunak terkemuka yang kritis terhadap keamanan, dan berpendapat bagaimana kontrol ini sebenarnya dapat meningkatkan produktivitas pengembang dan mengurangi waktu untuk bekerja. pasar melalui penggunaan metodologi yang dapat menemukan kelemahan perangkat lunak dengan lebih cepat dan mengurangi ketergantungan pengembang satu sama lain dan pada birokrasi yang mematikan produktivitas.

3.6.1 Kriteria Umum

Standar internasional untuk mengevaluasi keamanan sistem TI adalah ISO/IEC 15408, yang lebih dikenal sebagai Common Criteria.

> *Dengan 26 negara peserta, standar Kriteria Umum mewakili pencapaian mengagumkan dalam memperoleh penerimaan global dan tentu saja merupakan upaya standardisasi evaluasi keamanan internasional paling komprehensif yang pernah dilakukan.*

Berdasarkan Kriteria Umum, produk dievaluasi berdasarkan Profil Perlindungan yang menentukan persyaratan fungsional keamanan dan persyaratan jaminan rangkaian produk. Persyaratan fungsional adalah kebijakan atau perlindungan keamanan yang diklaim diterapkan oleh suatu produk. Persyaratan jaminan adalah kontrol seperti manajemen konfigurasi dan pengujian yang harus diikuti oleh pengembang untuk memastikan bahwa persyaratan fungsional direalisasikan dengan benar.

> *Dalam profil perlindungan Kriteria Umum, persyaratan jaminan secara kolektif ditandai dengan tingkat keamanan (disebut Tingkat Jaminan yang Dievaluasi, atau EAL) yang mewakili keseluruhan keyakinan yang dapat dimiliki oleh pemangku kepentingan bahwa persyaratan fungsional keamanan benar-benar dipenuhi oleh produk yang sesuai.*

Ada profil perlindungan untuk firewall, aplikasi antivirus, sistem operasi, perangkat seluler, dan banyak bentuk teknologi lainnya. Profil perlindungan dapat ditulis oleh siapa saja, namun profil itu sendiri harus dievaluasi, memastikan bahwa produk diukur berdasarkan standar yang dipahami dengan baik, valid, dan diterima. Dalam praktiknya, sebagian besar profil perlindungan dibuat oleh organisasi keamanan pemerintah seperti NSA.

Meskipun standar Common Criteria memberikan kerangka kerja teknis yang sangat baik dan terbukti untuk menentukan dan mengevaluasi keamanan perangkat lunak, perangkat keras, perangkat, dan sistem, standar tersebut memiliki sejarah politik yang mengecewakan. Seperti banyak alat yang bertujuan baik, potensi kegunaannya tidak selalu berarti produktivitas. Standar Kriteria Umum telah menderita akibat pengambilan kebijakan yang buruk dan proses evaluasi yang tidak efisien. Misalnya, profil perlindungan dengan jaminan tinggi, yang persyaratannya disusun dan ditulis dalam beberapa bulan, memerlukan waktu enam tahun bagi pemerintah AS untuk mengevaluasinya karena adanya penyesuaian, perdebatan, dan birokrasi yang tiada henti. Sertifikasi produk pertama terhadap profil perlindungan ini memakan waktu empat tahun, meskipun faktanya produk tersebut telah terbukti di lapangan selama satu dekade sebelum evaluasi dan tidak ada kerentanan signifikan yang ditemukan selama evaluasi. Ketidakmampuan untuk melaksanakan sertifikasi dengan jaminan tinggi secara efisien telah menurunkan Common Criteria ke sertifikasi tingkat rendah yang memerlukan dokumen yang sangat besar dan mahal, tanpa adanya keyakinan bahwa produk yang disertifikasi aman dari penyerang yang gigih bahkan dengan tingkat kecanggihan yang moderat. Diharapkan lembaga pemerintah dapat mereformasi pendekatan mereka terhadap apa yang akan dievaluasi (menghindari validasi mahal yang tidak memberikan jaminan signifikan bagi pengguna) dan bagaimana evaluasi dilakukan, dan mendapatkan manfaat yang luar biasa dari pendekatan dan standar keamanan yang penting ini.

Namun demikian, mari kita fokus pada kontribusi penting dari Common Criteria sebagai kerangka teknis untuk validasi ahli independen. Tingkat jaminan Kriteria Umum berkisar dari EAL1 hingga EAL7. Arti umum dari level-level tersebut ditunjukkan pada tabel berikut:

EAL1	Diuji secara fungsional
EAL2	Diuji secara struktural
EAL3	Diuji dan diperiksa secara metodis
EAL4	Dirancang, diuji, dan ditinjau secara metodis
EAL5	Dirancang dan diuji secara semiformal
EAL6	Desain yang diverifikasi secara semiformal dan diuji
EAL7	Desain yang diverifikasi secara formal dan diuji

Produk keperluan umum, seperti Windows, Linux, VMware, server database Oracle, dan router Cisco, yang tidak dirancang khusus untuk tingkat jaminan tinggi, telah disertifikasi di EAL4 atau lebih rendah. Kriteria Umum menyatakan bahwa EAL4 “adalah tingkat tertinggi yang memungkinkan secara ekonomi layak untuk melakukan retrofit pada lini produk yang

ada."[23] Hal ini masuk akal karena di atas EAL4, diperlukan teknik desain yang aman dengan formalitas yang ditingkatkan.

Seperti yang dijelaskan oleh salah satu pakar keamanan, "Para pakar keamanan telah mengatakan selama bertahun-tahun bahwa keamanan produk keluarga Windows sangat tidak memadai. Sekarang ada sertifikasi pemerintah yang ketat yang mengonfirmasi hal ini."[24] Selain itu, Windows dan produk lainnya dievaluasi berdasarkan profil perlindungan yang memberikan penafian berikut: "tidak dimaksudkan untuk diterapkan pada keadaan di mana perlindungan diperlukan terhadap upaya tekun oleh pihak yang bermusuhan dan jahat. mendanai penyerang untuk melanggar keamanan sistem."[25]

Beberapa profil perlindungan pada dasarnya mengabaikan tingkat jaminan, dan malah berupaya mendefinisikan persyaratan fungsional untuk memfasilitasi standar dan meningkatkan interoperabilitas. Misalnya, NSA telah mengembangkan beberapa profil perlindungan EAL2 yang memberikan panduan umum untuk jenis kontrol keamanan yang harus dimiliki oleh kelas produk tertentu. Salah satu contohnya adalah Profil Perlindungan Pemerintah A.S. untuk Sistem Operasi Tujuan Umum dalam Lingkungan Jaringan, yang menjelaskan persyaratan untuk otentikasi pengguna, kontrol akses diskresi objek sistem operasi, layanan kriptografi, dan layanan audit.[26] Meskipun kontrol keamanan berguna untuk dipahami, pengembang tidak boleh menafsirkan bagian jaminan dari spesifikasi ini sebagai memberikan tingkat kepercayaan yang berarti terhadap fungsi keamanan tersebut.

3.6.2 Studi Kasus: Profil Perlindungan Sistem Operasi

Selain yang disebutkan di atas, pemerintah AS telah membuat beberapa profil perlindungan terkait sistem operasi lainnya. Spesifikasi sistem operasi ini memiliki tingkat keamanan yang berbeda-beda, sehingga kurang lebih sesuai untuk lingkungan ancaman tertentu. Profil sistem operasi dan lingkungan ancaman yang berlaku tercantum dalam tabel berikut:

Nama	Judul	Tingkat Keamanan	Lingkungan Ancaman
SKPP	Kernel Pemisahan diLingkungan Berkekuatan Tinggi	EAL6+ / Kekokohan Tinggi	Pengelolaan informasi rahasia dan informasi bernilai tinggi lainnya, yang kerahasiaan, integritas atau keterbukaan harus dilindungi.[27] Kehadiran agen ancaman yag canggih dan sumber daya bernilai tinggi. [28]
CAPP	Profil Perlindungan Akses Terkendali	EAL4+	Komunitas pengguna yang tidak bermusuhan dan dikelola dengan baik.[29] Upaya yang tidak disengaja atau biasa saja untuk melanggar keamanan sistem.[30]

[23] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements, Version 2.1. Section 6.2.4; August 1999.

[24] Shapiro JS. Understanding the Windows EAL4 Evaluation. Computer February 2003;36(2):103e5.

[25] Labeled Security Protection Profile, Version 1.b. Section 1.2, p. 9. http://www.niap-ccevs.org/cc-scheme/pp/PP_OS_LS_V1.b.pdf

[26] U.S. Government Protection Profile for General Purpose Operating Systems in a Networked Environment. Version 1.0; August 30, 2010.

[27] U.S. Government Protection Profile for Separation Kernels in Environment Requiring High Robustness, Version 1.03. Section 1.2, paragraph 9, p. 10. http://www.niap-ccevs.org/cc-scheme/pp/pp_skpp_hr_ v1.03.pdf.

[28] Ibid., Section 2.8, paragraph 102, p. 47.

[29] Controlled Access Protection Profile, Version 1.d. Section 1.2, p. 9. http://www.niap-ccevs.org/cc-scheme/pp/pp_os_ca_v1.d.pdf

[30] Ibid.

			Tidak dimaksudkan untuk diterapkan pada keadaan dimana perlindungan diperlukan terhadap upaya-upaya yang dilakukan oleh penyerang yang bermusuhan dan memiliki dana yang besar.[31]31
CCOPP-OS	Compartmentalixed Operation Protection profile – Operating System	EAL4	Tidak diharapkan memberi perlindungan yang memadai terhadap serangan canggih. Pengguna ssangat dipercaya untuk tidak berupaya menumbangkan sistem secara jahat atau mengeksploitasi informasi yang disimpan secara jahat diatasnya.
LSPP	Labeled Security Protection Profile	EAL4+	Komunitas pengguna yang tidak bermusuhan dan dikelola dengan baik.[32] Upaya yang tidak disengaja atau biasa saja untuk melanggar keamanan sistem.[33] Tidak dimaksudkan untuk diterapkan pada keadaan dimana perlindungan diperlukan terhadap upaya-upaya yang dilakukan oleh penyerang yang bermusuhan dan memiliki dana yang besar. [34]
SLOS	Single Level Operating System in Medium Robustness Environments	EAL4+	Cocok untuk digunakan dilingkungan yang tidak terklasifikasi.[35] Tidak sesuai untuk “informasi paling sensitif/kepemilikan organisasi” ketika terpapar pada jaringan yang dapat diakses publik.[36] Kemungkinan adanya upaya kompromi adalah sedang.[37] Motivasi agen ancaman akan rata-rata.[38]
MLOS	Multilevel Operating System in Medium Robustness Environments	EAL4+	Cocok untuk digunakan dilingkungan yang terklasifikasi.[39] Tidak sesuai untuk “kebanyakan organisasi informasi sensitif/kepemilikan” kapan terkena yang dapat diakses publik jaringan. [40] Kemungkinan adanya upaya kompromi sedang.[41] Motivasi agen ancaman akan rata-rata.[42]

[31] Ibid.
[32] Labeled Security Protection Profile, Version 1.b. Section 1.2, p. 9. http://www.niap-ccevs.org/cc-scheme/pp/PP_OS_LS_V1.b.pdf
[33] Ibid.
[34] Ibid.
[35] U.S. Government Protection Profile for Single-level Operating Systems in Medium Robustness Environments, Version 1.91. Section 1.2, paragraph 11, p. 10. http://www.niap-ccevs.org/cc-scheme/pp/PP_OS_SL_MR2.0_ V1.91.pdf
[36] Ibid.
[37] Ibid., Section 3.1, paragraph 64, p. 28.
[38] Ibid.
[39] U.S. Government Protection Profile for Multilevel Operating Systems in Medium Robustness Environments, Version 1.91. Section 1.2, paragraph 11, p. 10. http://www.niap-ccevs.org/cc-scheme/pp/pp_os_ml_mr2.0_v1. 91.pdf
[40] Ibid.
[41] Ibid., Section 3.1, paragraph 66, p. 30.
[42] Ibid.

Seperti yang dapat dilihat dari tabel ini, hanya SKPP yang sesuai untuk melindungi informasi bernilai tinggi yang terkena ancaman penyerang yang canggih dan gigih.

SKPP adalah profil perlindungan yang dibuat oleh NSA untuk menentukan persyaratan keamanan untuk sistem operasi tertanam yang "berkekuatan tinggi". Sistem operasi dengan ketahanan tinggi mengendalikan sistem komputer yang mengelola dan melindungi sumber daya bernilai tinggi dalam menghadapi serangan oleh musuh yang memiliki banyak akal (lihat tabel berikut). Menurut pedoman Departemen Pertahanan, ketahanan yang tinggi mengacu pada "layanan dan mekanisme keamanan yang memberikan perlindungan paling ketat dan tindakan penanggulangan keamanan yang ketat."[43] Sistem operasi yang sesuai dengan SKPP harus menjamin bahwa aplikasi jahat tidak dapat membahayakan (merusak, menolak layanan, mencuri informasi dari, dll.) aplikasi lain yang berjalan di komputer.

Tingkat ancaman penyerang

		Ancaman Rendah	Ancaman Sedang	Ancaman Tinggi
Nilai Aset	Bernilai Tinggi	Rendah	Sedang	Tinggi
	Nilai Sedang	Rendah	Sedang	Sedang
	Nilai Rendah	Rendah	Rendah	Rendah

Persyaratan SKPP jauh lebih ketat dibandingkan standar keamanan sistem operasi lainnya. Kepastian atau keyakinan yang dihasilkan oleh pengembang, pengguna, dan pemangku kepentingan lainnya dari evaluasi SKPP yang independen sangatlah tinggi dan belum pernah terjadi sebelumnya dalam dunia keamanan komputer. SKPP memerlukan proses pengembangan yang sangat ketat, metode formal yang memberikan bukti keamanan secara matematis, dan pengujian penetrasi oleh pakar keamanan NSA yang memiliki akses penuh terhadap kode sumber. Oleh karena itu, persyaratan jaminan SKPP merupakan contoh yang sangat baik dari standar validasi ahli independen yang akan memberikan jaminan keamanan tingkat tinggi kepada pengembang sistem tertanam dan pelanggan mereka. Meskipun produk hasil evaluasi SKPP sudah banyak digunakan, namun SKPP sendiri sudah tidak digunakan lagi untuk evaluasi produk baru. Hal ini disebabkan oleh ketidakmampuan pemerintah AS saat ini untuk melakukan evaluasi pada tingkat efisiensi biaya dan jadwal yang dapat dibenarkan dalam kondisi pasar saat ini. Meski demikian, aparat keamanan tetap dapat menggunakan bukti berbasis SKPP dalam akreditasi yang dilakukan di luar program evaluasi Common Criteria.

Mengikuti pedoman jaminan EAL6+ yang berlaku, seperti yang dicontohkan dalam SKPP, dan memperoleh evaluasi independen atas bukti jaminan terkait oleh konsultan ahli akan menjadi pendekatan yang sangat baik bagi pengembang sistem tertanam yang ingin meningkatkan standar keamanan produk mereka.

Berikut pedoman jaminan EAL6+ yang berlaku, seperti yang dicontohkan dalam SKPP, dan memperoleh evaluasi independen atas bukti jaminan terkait oleh konsultan ahli akan menjadi pendekatan yang sangat baik bagi pengembang sistem tertanam yang ingin meningkatkan standar keamanan produk mereka. Kami akan mengeksplorasi persyaratan jaminan ini,

[43] Department of Defense Instruction 8500.2, Information Assurance (IA) Implementation; February 6, 2003.

membandingkan pengendalian jaminan tinggi dan rendah, secara lebih rinci di bagian berikut. Pembaca juga dihimbau untuk membaca SKPP, terutama persyaratan penjaminannya.

Persyaratan jaminan proses pengembangan Kriteria Umum, dengan perbandingan leveling numerik antara SKPP dan CAPP (profil yang paling banyak diterapkan dalam sistem operasi perusahaan), ditunjukkan pada tabel berikut:

Requirement	Description	SKPP	CAPP	Notes
ACM_AUT	Configuration management automation	2	0	SKPP requires complete automation
ACM_CAP	Configuration management capabilities	5	3	SKPP requires advanced support
ACM_SCP	Configuration management scope	3	1	SKPP CM requires coverage of development tools
ADV_ARC	Architectural design	1	0	
ADV_FSP	Functional specification	4	1	SKPP requires *formal* specification
ADV_HLD	High-level design	4	2	SKPP requires semiformal high-level design
ADV_IMP	Implementation representation	3	0	SKPP requires rigorously defined transformation from representation to implementation
ADV_LLD	Low-level design	2	0	SKPP requires semiformal low-level design
ADV_RCR	Representation correspondence	3	1	SKPP requires *formal* correspondence
ALC_DVS	Development security	2	1	SKPP requires confidentiality and integrity measures during development
ALC_FLR	Flaw remediation	3	0	SKPP requires systematic flaw remediation
ALC_LCD	Life-cycle definition	2	0	SKPP requires a standardized life-cycle model
ALC_TAT	Tools and techniques	3	0	SKPP requires total system compliance with implementation standards
AMA_AMP	Assurance maintenance plan	1	0	
ATE_COV	Analysis of test coverage	3	2	SKPP requires complete test coverage of all functional requirements
ATE_DPT	Depth of testing	3	1	SKPP requires testing against high-level design, low-level design, and implementation
ATE_FUN	Functional testing	2	1	SKPP requires ordered functional testing
ATE_IND	Independent testing	3	2	SKPP requires independent testing of all requirements (CAPP only requires sampling)
AVA_VLA	Vulnerability assessment	4	1	SKPP requires *NSA penetration testing;* others do not

Manajemen Konfigurasi

Meskipun CAPP memerlukan penggunaan sistem manajemen konfigurasi dan kemampuan untuk membuat ulang produk dari sistem manajemen konfigurasi, SKPP mencakup otomatisasi lengkap sistem CM, termasuk kemampuan untuk secara otomatis mengidentifikasi semua aspek produk yang terpengaruh oleh modifikasi item konfigurasi apa pun. SKPP juga mensyaratkan bahwa item tambahan, bukan hanya kode sumber produk, harus dikelola konfigurasinya. Misalnya, file pendukung apa pun (misalnya skrip) yang digunakan untuk pembuatan gambar produksi harus disimpan di bawah CM, dan semua alat yang digunakan untuk menghasilkan produk yang dapat dieksekusi harus tercakup dalam CM. Hal ini sangat penting karena potensi perubahan alat (misalnya kompiler) yang menyebabkan perubahan perilaku produk yang tidak terduga. Sebuah studi kasus mengenai risiko ini telah disajikan sebelumnya dalam bab ini.

Persyaratan SKPP juga mencakup rencana dan dokumentasi terkait CM, termasuk demonstrasi tentang bagaimana tindakan keamanan terkait CM mencegah perubahan tidak sah pada produk dan bagaimana prosedur penerimaan diikuti untuk tinjauan perubahan.

Spesifikasi Fungsional

Spesifikasi fungsional mengacu pada deskripsi antarmuka dan perilaku produk. CAPP hanya memerlukan spesifikasi fungsional informal. SKPP melangkah lebih jauh dengan mensyaratkan bahwa seluruh rincian fungsi keamanan tercakup dalam spesifikasi fungsional dan spesifikasi ini dibuat dengan gaya formal. Formalitas ini merupakan persyaratan penting yang diratakan dari EAL7. Dalam kasus sistem operasi Green Hills Software INTEGRITY-178B yang berhasil dievaluasi berdasarkan SKPP, perilaku sistem operasi dimodelkan secara formal dalam bahasa fungsional yang cocok untuk pembuktian teorema otomatis. Ini memungkinkan verifikasi kebijakan keamanan terhadap model tersebut. Definisi formal kebijakan keamanan dan bukti formal kebijakan terhadap modelnya tidak diperlukan dalam profil perlindungan lainnya.

Leveling Jaminan

Untuk lebih memahami bagaimana tingkat jaminan diterapkan pada kelompok persyaratan jaminan dan komponennya, mari kita lihat contoh komponen jaminan Kriteria Umum 2.1: ADV_HLD, “Desain Tingkat Tinggi.” Menurut Kriteria Umum, “Persyaratan desain tingkat tinggi dimaksudkan untuk memberikan jaminan bahwa TOE [target evaluasi] menyediakan arsitektur yang sesuai untuk mengimplementasikan persyaratan fungsional keamanan TOE.” Jika profil proteksi mencakup persyaratan jaminan ini (sebagian besar mencakupnya), maka tingkat keparahan persyaratan tersebut akan meningkat seiring dengan tingkat jaminan:

EAL1	Tidak diperlukan desain tingkat tinggi
EAL2	ADV_HLD.1: Desain tingkat tinggi yang deskriptif
EAL3	ADV_HLD.2: Keamanan menerapkan desain tingkat tinggi
EAL4	ADV_HLD.2: Keamanan menerapkan desain tingkat tinggi
EAL5	ADV_HLD.3: Desain tingkat tinggi semiformal
EAL6	ADV_HLD.4 : Penjelasan tingkat tinggi semiformal
EAL7	ADV_HLD.5: Desain formal tingkat tinggi

Anda dapat melihat dari contoh ini bahwa persyaratan jaminan untuk komponen tertentu tidak selalu meningkat seiring dengan peningkatan tingkat jaminan. Sebaliknya, persyaratan menjadi lebih ketat bila diperlukan untuk mencapai tujuan kepercayaan pada tingkat yang baru. Komponen ADV_HLD dibagi lagi menjadi elemen-elemen yang merupakan persyaratan deskriptif individual. Ketika persyaratan meningkat di antara tingkat jaminan, elemen ditambahkan ke persyaratan yang diwarisi dari tingkat sebelumnya. Misalnya, profil perlindungan EAL5 akan menyertakan elemen ADV_HLD.3.9C:

"Desain tingkat tinggi harus menggambarkan pemisahan TOE menjadi subsistem yang menegakkan TSP [Kebijakan Keamanan TOE] dan subsistem lainnya."

Untuk produk firewall, ini mungkin berarti bahwa desain tingkat tinggi harus menunjukkan subsistem yang menerapkan aturan penyaringan lalu lintas. Untuk profil perlindungan EAL6, persyaratan ini ditambah menjadi ADV_HLD.4.10C:

"Rancangan tingkat tinggi harus membenarkan bahwa cara yang teridentifikasi untuk mencapai pemisahan, termasuk mekanisme perlindungan apa pun, sudah cukup untuk memastikan pemisahan yang jelas dan efektif antara fungsi yang menerapkan TSP dan fungsi yang tidak menerapkan TSP."

Dengan kata lain, evaluator akan mengharapkan vendor untuk memberikan bukti desain tingkat tinggi yang menunjukkan bahwa subsistem penyaringan lalu lintas dilindungi dari korupsi, subversi, dan sebagainya, oleh subsistem lain.

Untuk profil perlindungan EAL7, persyaratan ini diperluas lagi menjadi ADV_HLD.5.10C, yang sama dengan ADV_HLD.4.10C, hanya saja presentasi atau pembenarannya harus formal:

"Spesifikasi formal ditulis dalam notasi berdasarkan konsep matematika yang sudah mapan. Konsep matematika ini digunakan untuk mendefinisikan sintaksis dan semantik notasi serta aturan pembuktian yang mendukung penalaran logis. Aturan sintaksis dan semantik yang mendukung notasi formal harus menentukan cara mengenali konstruksi secara jelas dan menentukan maknanya. Perlu ada bukti bahwa kontradiksi tidak mungkin diperoleh, dan semua aturan yang mendukung notasi tersebut perlu didefinisikan atau dirujuk."[44]

Di dunia elektronik modern, klaim keamanan yang tidak berdasar dan menyesatkan adalah hal yang lumrah. Calon pelanggan menghargai pengembang yang mendukung klaim mereka dengan penilaian independen, baik yang diformalkan seperti dalam contoh Kriteria Umum, atau yang lebih informal namun ketat. Beberapa vendor keamanan tertanam memberikan konsultasi independen, evaluasi, dan penilaian tertulis untuk pelanggan mereka; pengembang kemudian dapat menggunakan bukti yang dihasilkan untuk secara dramatis

[44] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Requirements, Version 2.1. Section 10, paragraph 308; August 1999.

meningkatkan klaim mereka atas keamanan produk. Demikian pula, menggunakan komponen yang telah divalidasi secara independen merupakan pendekatan efektif lainnya untuk meningkatkan keamanan dan klaim pengembang terhadapnya. Konsep ini juga berlaku pada sistem yang kritis terhadap keselamatan. Dalam studi kasus berikut, penilaian independen yang lebih informal disertakan dalam proses pengembangan dan penerapan untuk produk tertanam dengan jaminan tinggi.

3.7 SERVER WEB JAMINAN TINGGI HAWSD

Studi Kasus

Studi kasus ini menyatukan semua konsep FASE yang dijelaskan dalam bab ini. Pada tahun 2008, Green Hills Software mengembangkan server web dengan jaminan tinggi (HAWS) untuk menghosting situs web yang kebal terhadap peretas. HAWS diterapkan sebagai aplikasi pada teknologi sistem operasi real-time yang tertanam INTEGRITAS dengan jaminan tinggi dan bersertifikat keamanan. Perancang utama server web adalah Dan Hettena, arsitek sistem operasi senior, keamanan, dan virtualisasi di Green Hills Software.

Mengapa Green Hills memutuskan untuk membuat server web dari awal daripada mencoba mengunci server web komersial seperti Apache? Apache adalah server web sumber terbuka yang sangat kuat dan sukses, menampung sebagian besar situs web di dunia pada saat tulisan ini dibuat. Alasan untuk tidak mengadopsi Apache dalam kasus khusus ini adalah karena nilai sumber daya yang akan dikelola dan dilindungi oleh portal web dianggap sangat tinggi, dan ancaman penyerang canggih yang ingin menumbangkan atau merusak portal web juga dianggap sangat tinggi. Kombinasi sumber daya bernilai tinggi yang terpapar pada lingkungan dengan ancaman tinggi memerlukan jaminan keamanan yang tinggi sehingga tidak dapat disesuaikan dengan perangkat lunak yang awalnya tidak dikembangkan untuk memenuhi standar ini.

Basis kode Apache terdiri dari ratusan ribu baris kode yang dikembangkan menggunakan proses tujuan umum komersial, bukan menggunakan standar keamanan jaminan tinggi. Ratusan kerentanan perangkat lunak telah ditemukan di Apache selama bertahun-tahun, dan kemungkinan kerentanan parah yang belum ditemukan dalam snapshot perangkat lunak apa pun dianggap sangat tinggi. Salah satu contoh kerentanan dari US CERT National Vulnerability Database adalah sebagai berikut:

> **C=CVE-2010-0425:** "MODULES/ARCH/WIN32/MOD_ i sapi.c in mod_ i sapi in teh the apache HTTP Server does not ensure that processing is complete before calling isapi_unload...which allows remote attackers to execute arbitrary code via unspecified vectors related to a crafted request."

Kebijakan keamanan utama HAWS adalah untuk memastikan bahwa konten portal web tidak dapat dirusak secara jahat dan bahwa permintaan dari klien yang sah tidak ditolak layanannya oleh klien (jahat) lainnya. Keamanan fisik komputer server web, serta kualitas layanan jaringan hulu, berada di luar cakupan tujuan desain HAWS (walaupun masalah ini telah diatasi di tingkat sistem).

Selain kebijakan integritas dan ketersediaan yang ketat ini, HAWS dirancang untuk menangani ribuan koneksi bersamaan dengan kinerja waktu respons yang baik. Penting untuk dicatat bahwa Apache adalah pilihan yang sangat baik untuk sebagian besar sistem (dan pada kenyataannya port telah dilakukan ke sistem operasi INTEGRITAS yang sama yang menampung HAWS), namun tingkat keamanan yang lebih tinggi dianggap perlu untuk aplikasi khusus ini. Mari kita bahas prinsip-prinsip PHASE secara bergantian, dengan memberikan contoh bagaimana setiap prinsip diterapkan pada pengembangan HAWS.

3.7.1 Implementasi Minimal

Berbeda dengan ratusan ribu baris kode server web komersial pada umumnya, mesin layanan web HAWS terdiri dari sekitar 300 baris kode. Prestasi ini dicapai terutama dengan mengubah logika canggih penguraian dan merespons permintaan HTTP menjadi mesin status sederhana yang beroperasi pada database informasi read-only yang telah dimuat sebelumnya yang sesuai dengan contoh tertentu dari file yang dihosting yang dilayani oleh HAWS. Potongan kode yang dibuat dengan cermat ini menyediakan layanan HTTP yang sesuai dengan HTML/1.1 (RFC 2616), versi HTTP yang umum digunakan di seluruh dunia. Situs web awal yang dihosting di HAWS dirancang secara profesional dan mencakup grafik canggih (Adobe Flash) dan JavaScript. Loop mesin status pusat ditampilkan di sini (komentar kode dan makro dihilangkan). Pembaca tidak diharapkan untuk memahami kode tersebut; itu disediakan untuk menunjukkan contoh implementasi minimal:

```
while (preempt_count > 0) {
uint64_t inst = INST_TABLE(task->pc);
uint32_t nextpc = task->pc + 1;
preempt_count -:
if (((inst >>59) & 0xf) =0xf && task->ibuf_off < task->ibuf_len &&
task->ibuf[task->ibuf_off] == (uint8_t)(inst >> 48)) {
task->ibuf_off++;
task->inst_count++:
} else {
if ((inst >> 61) && task->ibuf_off == task->ibuf_len) (
preempt_count -= 100;
if (!refill_ibuf(task))
return;
}
task->inst_count++;
c1=((inst >> 61) & 0x1) ?
task->ibuf[task->ibuf_off]: task->GPR((uint8_t)(inst>> 48));
c2=((inst >> 61) & 0x1) ?
(uint8_t)(inst >> 48): ((inst >> 20) & 0xfffff):
cond=(cop & 2) ? ((c1=c2) ^(cop&1)): ((c1<c2) |(cop&1));
if ((inst >> 62) & (1 << cond))
task->ibuf_off++;
if (cond) {
uint8_t xop8=(inst >> 40) & 0xff;
uint32_t xop20=(inst >> 0) & 0xfffff;
// Note: switch not used in order to optimize common cases
if (opx=0) {
nextpc= xop20;
} else if (opx =1) {
```

```
nextpc - task->GPR(xop8):
} else if (opx == 2) {
if (task->ibuf_off != 0)
task->ibuf[-(task->ibuf_off)] = xop8;
} else if (opx -3) {
preempt_count -= 400;
if (!continue_send(task, xop20))
return;
} else if (opx - 4) {

task->GPR(xop8) = xop20;
} else if (opx == 5) {
task->GPR(xop8) - (ticks - task->start_ticks) & 0xfffff;
} else if (opx =6) {
free_nettask(task);
return;
}
}
}
task->pc = nextpc;
}
```

Hal ini dibiarkan sebagai latihan bagi pembaca untuk menelusuri kode Apache open source, membandingkan implementasi yang digerakkan oleh mesin negara sebelumnya dengan penguraian masukan tradisional yang sarat dengan kumpulan logika kasus khusus. Namun, berhati-hatilah: perangkat lunak Apache yang menangani penanganan dan respons masukan yang sesuai dengan HTTP terdiri dari ribuan baris kode.

Dengan menghindari mekanisme parsing umum, termasuk operasi pembacaan jaringan besar dan pencocokan string, HAWS menghindari jenis logika runtime yang rentan terhadap kerentanan (misalnya buffer overflows). Terakhir, fitur ketersediaan HAWS diimplementasikan dengan kebijakan sederhana, bukan heuristik yang rumit. Meskipun demikian, kebijakan-kebijakan ini dapat mencapai tujuannya dengan baik; HAWS telah terbukti tahan terhadap pengujian penetrasi yang canggih dan belum pernah mengalami downtime yang tidak terjadwal atau laporan ketidaktersediaan layanan sejak penerapannya pada tahun 2008. Fitur ketersediaan dijelaskan lebih lanjut di bagian berikut.

3.7.2 Arsitektur Komponen

Mesin layanan web HAWS berjalan sebagai proses mode pengguna yang terisolasi dan dilindungi memori. HAWS memanfaatkan kebijakan partisi waktu dan ruang INTEGRITY yang ketat untuk menjamin bahwa komponen HAWS tidak dapat mempengaruhi komponen sistem lainnya (driver perangkat jaringan dan tumpukan TCP/IP) dan sebaliknya. HAWS tidak berbagi ruang alamat dengan proses atau plug-in lain dan tidak memasukkan kode apa pun secara dinamis (misalnya, DLL).

Setiap koneksi klien jarak jauh memiliki masa pakai maksimum yang diberlakukan secara ketat (komponenisasi temporal), dan tingkat HAWS membatasi permintaan koneksi baru untuk mencegah serangan DoS dan memberikan jaminan kualitas layanan pada koneksi yang valid.

3.7.3 Hak Istimewa Terkecil

Mesin layanan HAWS tidak diberikan sumber daya sistem selain waktu CPU dan memori yang dialokasikan dan aliran byte dua arah yang sesuai dengan permintaan dan respons HTTP. Mesin layanan tidak memiliki kemampuan untuk mengakses sistem file atau perangkat jaringan apa pun atau untuk meluncurkan program lain. Alokasi kemampuan sumber daya ini ditentukan dalam kebijakan keamanan sistem.

Mesin negara yang disebutkan di atas beroperasi pada data yang telah diproses sebelumnya (file yang disajikan) yang dipetakan secara read-only langsung ke ruang alamat mesin layanan. Akses hanya-baca melindungi terhadap kerusakan yang tidak disengaja pada data yang disajikan dan secara umum merupakan pilihan desain yang baik untuk keamanan. Selain HTTP, HAWS tidak memiliki port jaringan lain. Faktanya, konten situs web itu sendiri tidak dapat diperbarui dari jarak jauh (misalnya melalui FTP aman) karena dianggap sebagai risiko yang tidak perlu. Sebaliknya, hanya administrator keamanan lokal yang berwenang dengan akses fisik ke komputer server web yang dapat memperbarui konten.

3.7.4 Proses Pembangunan yang Aman

HAWS dikembangkan menggunakan proses jaminan tinggi yang ketat oleh grup rekayasa sistem operasi real-time Green Hills Software. Proses pengembangan mencakup banyak prinsip yang dijelaskan dalam bab ini, termasuk pengujian menyeluruh, manajemen konfigurasi yang ketat, tinjauan sejawat, penegakan otomatis standar pengkodean yang telah terbukti, dan desain formal.

3.7.5 Validasi Ahli Independen

Sebuah organisasi peretas “topi putih” independen dan terkemuka melakukan analisis dan uji penetrasi terhadap HAWS dan salah satu situs web yang dihostingnya selama jangka waktu yang lama. Organisasi ini telah melakukan penelitian keamanan tingkat rendah yang tidak memihak, canggih secara teknis, dan tidak memihak sejak tahun 1998. Perusahaan keamanan tidak dapat menemukan kerentanan apa pun, meluncurkan serangan penolakan layanan yang berhasil, atau merusak situs.

Meskipun tidak dimaksudkan sebagai pengganti Apache atau server web komersial berfitur lengkap lainnya, HAWS memberikan contoh penerapan konsep PHASE yang praktis, sukses, dan diterapkan.

3.8 DESAIN BERDASARKAN MODEL

Guy Broadfoot dan Philippa Hopcroft menulis bagian berikut tentang desain berbasis model (MDD). Guy Broadfoot adalah pendiri dan CTO Verum Software Technologies. Seorang veteran di industri pengembangan perangkat lunak, ia menggabungkan keahlian akademis yang mendalam dengan pengalaman teknis yang kuat dalam mengembangkan produk dan membalikkan proyek perangkat lunak yang gagal. Broadfoot telah merancang dan mengimplementasikan proyek perangkat lunak sejak tahun 1965, dan dalam beberapa tahun terakhir telah mengembangkan minat khusus dalam penerapan metode formal. Pada tahun 1986, Broadfoot mendirikan Silverdata, sebuah perusahaan perangkat lunak yang mengembangkan produk untuk navigasi kelautan dan survei hidrografi. Dia menjual hak produk kepada investor swasta pada tahun 1998. Dia bekerja sebagai konsultan di Mountside Software Engineering sejak tahun 1998, di mana dia bertanggung jawab untuk membangun

perencanaan proyek perangkat lunak yang ketat dan sistem pengendalian keuangan. Broadfoot lulus dengan predikat cemerlang dari Universitas Oxford pada tahun 2001 dengan gelar master di bidang rekayasa perangkat lunak.

3.8.1 Pengantar MDD

Tren yang Mendorong Penerapan MDD

Buku ini sering menekankan tren peningkatan kompleksitas perangkat lunak dan sistem yang menimbulkan kerentanan keamanan. Karena mayoritas pengembang perangkat lunak tertanam terutama menggunakan bahasa pemrograman tradisional, khususnya C dan C++, bab ini sejauh ini berfokus pada proses dan teknik untuk meningkatkan keandalan dan dengan demikian mengurangi kelemahan keamanan dalam basis kode tradisional. Namun, pendekatan lain yang semakin sukses adalah penggunaan desain berbasis model (MDD).

> *Premis MDD adalah untuk meningkatkan abstraksi pengembangan perangkat lunak dari bahasa pemrograman imperatif tingkat rendah yang penuh dengan sisi tajam dan peluang untuk menyalahkan diri sendiri ke bahasa pemodelan tingkat lebih tinggi yang mengurangi jarak antara desain dan implementasi. dan dengan demikian mengurangi kelemahan yang menyebabkan kegagalan keamanan dan keselamatan.*

Pemodelan juga lebih cocok untuk pembuktian formal spesifikasi dan kebijakan keamanan dibandingkan bahasa pemrograman tradisional. Memang benar, manfaat sampingan dari penggunaan beberapa platform MDD (yang mendukung metode formal dan pembuatan kode otomatis) adalah kemampuan untuk membuat argumen formal mengenai korespondensi antara spesifikasi, desain, dan implementasi, yang merupakan tantangan utama dalam semua pendekatan formal. Oleh karena itu, pembahasan MDD berikut ini condong pada metode yang dapat dianalisis secara formal dan oleh karena itu meningkatkan jaminan kualitas, keselamatan, dan keamanan.

Bagian ini memperkenalkan subjek desain berbasis model sebagaimana diterapkan pada pengembangan perangkat lunak tertanam. Ini menjawab pertanyaan-pertanyaan berikut:

1) Apa itu desain berbasis model?
2) Manfaat apa yang bisa diharapkan dengan menerapkannya pada pengembangan perangkat lunak tertanam? Jenis bahasa pemodelan apa yang saat ini digunakan?
3) Jenis platform MDD apa yang ada dan apa perbedaannya? Apa yang harus diperhatikan saat memilih platform MDD yang tepat? Bagaimana penggunaan platform MDD memenuhi standar keselamatan?

Dalam diskusi tentang MDD dan platform MDD, istilah verifikasi sering digunakan dan cenderung memiliki arti berbeda bagi orang yang berbeda. Menurut IEEE, [45]istilah verifikasi mempunyai dua definisi berikut:

1. Proses evaluasi suatu sistem atau komponen untuk menentukan apakah produk pada tahap pengembangan tertentu memenuhi kondisi yang ditetapkan pada awal tahap tersebut.
2. Bukti formal kebenaran program.

[45] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology; September 1990.

Kebanyakan vendor MDD menggunakan definisi pertama untuk istilah verifikasi; sepanjang bagian ini, kami melakukan hal yang sama. Dalam kasus yang dimaksudkan untuk membuktikan kebenaran program secara formal, kami menggunakan istilah verifikasi formal. Perbedaan ini penting: banyak platform MDD menawarkan fasilitas verifikasi, biasanya dalam bentuk simulasi eksekusi pada host pengembangan sebelum mengujinya pada sistem target. Fitur pembuatan kasus uji dalam satu atau lain bentuk sering kali mendukung aktivitas ini. Namun, hanya sedikit platform MDD yang menawarkan fasilitas verifikasi formal guna memberikan bukti kebenaran desain perangkat lunak, dan bagi platform MDD, ini merupakan kemampuan pembeda yang penting. Jika fasilitas ini ada, mereka menawarkan manfaat yang besar, terutama dalam desain multi-thread dan multi-core.

Kami menggunakan istilah komponen yang berarti beberapa elemen perangkat keras atau perangkat lunak fisik dari suatu sistem yang memenuhi satu atau lebih antarmuka. Kami menganggap suatu sistem terdiri dari kumpulan komponen. Istilah subsistem adalah suatu konsep pemodelan yang mendefinisikan pengelompokan komponen-komponen terkait dan/atau subsistem lainnya. Ini adalah abstraksi pemodelan untuk membantu partisi bertahap dari sistem yang besar dan kompleks. Realisasi fisik suatu subsistem selalu melalui komponen-komponen yang menyusunnya.

Metodologi pengembangan perangkat lunak secara historis cenderung berkonsentrasi pada alat yang membantu pengembang perangkat lunak dalam komposisi sistem, integrasi, dan khususnya pengujian. Berbagai teknologi berbasis komponen, seperti CORBA dan Microsoft COM, DCOM, dan .NET bertujuan untuk meningkatkan produktivitas dan kualitas pengembangan perangkat lunak dengan meningkatkan penggunaan kembali melalui abstraksi komponen di domain di mana teknologi tersebut dapat diterapkan.

Meskipun alat pengembangan yang semakin canggih, seperti kompiler pengoptimalan, debugger bahasa sumber, generator pengujian, dan platform eksekusi pengujian, telah lama tersedia untuk membantu pengembang perangkat lunak dalam implementasi dan pengujian, cara konvensional dalam mengembangkan perangkat lunak ini semakin mendapat tantangan dari: tekanan yang tiada henti untuk mengurangi waktu pemasaran dan memperkenalkan produk-produk baru dalam siklus yang semakin pendek, sering kali menyebabkan perangkat keras dan perangkat lunak dikembangkan secara paralel dan platform perangkat keras akhir tidak tersedia selama sebagian besar waktu perangkat lunak sedang dikembangkan; meningkatnya kompleksitas perangkat lunak yang tertanam, seiring dengan karakteristik produk yang penting dan nilai pengguna akhir yang diwujudkan oleh perangkat lunak; pengenalan teknologi multi-inti, menciptakan lingkungan di mana kesalahan desain perangkat lunak dan pengkodean lebih mungkin terjadi dan teknik pengujian konvensional kurang efektif; terus adanya tekanan terhadap biaya pembangunan; kecenderungan menuju lokasi pembangunan yang terdistribusi, seringkali dalam zona waktu yang berbeda.

Semua tantangan ini harus diatasi sambil mencapai tingkat kualitas yang semakin tinggi untuk memenuhi ekspektasi pengguna dan tekanan peraturan yang semakin meningkat. Pengembangan perangkat lunak konvensional membutuhkan banyak tenaga kerja dan dibatasi oleh kesalahan manusia. Menurut survei Venture Development Corporation baru-baru ini,[46] biaya tenaga kerja perangkat lunak mencakup lebih dari 50% total pengeluaran

[46] VDC Research. 2011 Embedded Engineer Survey; 2011.

untuk pengembangan sistem tertanam. Dari pengeluaran ini, menurut Ebert dan Jones, [47]pengalaman industri pada umumnya menghabiskan hingga 40% untuk pengujian dan pengerjaan ulang. Sulit membayangkan domain teknik lain yang biasanya menghabiskan 40% anggaran pengembangannya untuk mengidentifikasi dan menghilangkan cacat yang seharusnya tidak ada.

Secara historis, organisasi telah berusaha untuk mengatasi beberapa masalah ini melalui perbaikan proses, yaitu dengan memperkenalkan praktik, prosedur, dan standar yang mengatur setiap aspek spesifikasi, desain, implementasi, dan pengujian. Meskipun ketelitian ini diperlukan dalam proses pembangunan dengan jaminan tinggi, MDD dapat membantu mengurangi biaya tambahan tersebut, memungkinkan jaminan yang lebih tinggi sekaligus meningkatkan efisiensi pengembang, yang merupakan tema penting dalam bab ini. Seperti yang telah dibahas sebelumnya, penurunan efisiensi akibat overhead proses pengembangan dengan cepat menyebabkan berkurangnya laba atas investasi dan tidak mengherankan jika manajemen senior kehilangan minat. Akhirnya, melampaui ambang batas kompleksitas dan ukuran yang relatif kecil sebuah komponen, tidak mungkin mencapai jaminan keamanan dan keselamatan tinggi dengan teknik pengembangan perangkat lunak tradisional. MDD adalah alat untuk mengatasi masalah skalabilitas ini, memungkinkan pengembang perangkat lunak menjadi lebih produktif dan menghasilkan produk lebih cepat dan dengan lebih sedikit cacat.

Apa itu MDD?

Seperti disebutkan sebelumnya, desain berbasis model, atau MDD, adalah paradigma baru yang bertujuan untuk meningkatkan siklus hidup pengembangan perangkat lunak, khususnya untuk sistem perangkat lunak yang besar dan kompleks, dengan menyediakan tingkat abstraksi yang lebih tinggi untuk desain perangkat lunak dibandingkan dengan yang dimungkinkan saat ini dengan sistem konvensional. dan bahasa pemrograman yang banyak digunakan. Didukung oleh platform MDD yang sesuai, MDD mengklaim sebagai kemajuan teknologi.

MDD didasarkan pada penggunaan model secara sistematis sebagai artefak utama sepanjang siklus hidup rekayasa perangkat lunak. MDD bertujuan untuk mengangkat model fungsional ke peran sentral dalam spesifikasi, desain, integrasi, dan validasi perangkat lunak. Dalam pengembangan perangkat lunak konvensional, pernyataan definitif fungsi perangkat lunak merupakan campuran dari dokumen persyaratan fungsional informal dan kode sumber program, dan membangun korespondensi antara keduanya seringkali sulit. Dengan meningkatkan tingkat abstraksi di mana spesifikasi dan desain diungkapkan, MDD bertujuan untuk menggantikan pendekatan spesifikasi dan kode sumber dengan model fungsional yang diungkapkan dalam bahasa pemodelan yang jelas dan tepat yang dapat dipahami oleh para ahli domain yang menentukan sistem yang dihasilkan. melakukannya dan pengembang perangkat lunak yang mengimplementasikannya. Selain itu, kode yang dihasilkan secara otomatis dari model menjamin korespondensi antara spesifikasi dan implementasi.

MDD menggunakan model untuk mewakili elemen sistem, hubungan struktural di antara elemen tersebut, serta perilaku dan interaksi dinamisnya. Pemodelan hubungan struktural mendukung eksplorasi desain dan partisi sistem; pemodelan perilaku dan interaksi sangat penting untuk dapat memverifikasi desain dengan memverifikasi model dan untuk

[47] Ebert C, Jones C. Embedded Software: Facts, Figures, and Future. Computer April 2009;42(4):42e52.

pembuatan kode. Dalam praktiknya, dua poin terakhir ini merupakan sumber dari sebagian besar manfaat yang diklaim untuk MDD.

Bagi banyak pengembang perangkat lunak tertanam, pembuatan kode khususnya tampaknya sulit diterima.

> *Tanpa pembuatan kode, banyak manfaat MDD yang hilang; pada kenyataannya, memutuskan untuk mengadopsi pendekatan MDD berarti menerima pembuatan kode otomatis dari model.*

Kualitas kode yang dihasilkan dan kesesuaiannya untuk domain dan penggunaan tertentu merupakan pertimbangan penting dalam memilih platform MDD atau bahkan memutuskan untuk mengambil jalur MDD sama sekali. Kami membedakan antara dua jenis model, yaitu model yang dapat dieksekusi dan model yang dapat dieksekusi dan diverifikasi secara formal. Kami membahas masalah ini lebih lanjut di bagian berikut. Platform MDD apa pun yang menawarkan model yang tidak dapat dieksekusi memiliki nilai tambah yang kecil dan oleh karena itu diabaikan.

Potensi Manfaat MDD

Dengan meningkatkan tingkat abstraksi di mana spesifikasi dan desain diungkapkan dan menghilangkan spesifikasi fungsional informal, MDD mempunyai potensi untuk melakukannya

1. Hilangkan kesalahan spesifikasi dan desain di awal siklus pengembangan di tempat yang paling murah dan mudah untuk diperbaiki.
2. Meningkatkan tingkat otomatisasi yang dapat diterapkan pada proses pengembangan melalui pembuatan kode otomatis.
3. Memfasilitasi desain perangkat keras/perangkat lunak paralel dengan memungkinkan model diverifikasi dengan mensimulasikan perilaku eksekusi pada host pengembangan sebelum sistem target tersedia.
4. Mengurangi upaya pengujian yang diperlukan dengan menerapkan teknik verifikasi formal otomatis pada model fungsional yang dikombinasikan dengan simulasi perilaku eksekusi daripada hanya mengandalkan pengujian kode program yang diterapkan.

Tentu saja, menghilangkan kesalahan adalah pendorong utama di balik pembahasan MDD dari sudut pandang keamanan. Namun, seringkali manfaat paling berharga dari penerapan MDD pada pengembangan perangkat lunak tertanam adalah pengurangan biaya dan waktu pemasaran yang diperlukan untuk memenuhi tingkat fungsionalitas dan kualitas yang diperlukan.

Penggunaan bahasa pemodelan yang sesuai berkontribusi mengurangi cacat pada produk yang dihasilkan melalui dua cara. Pertama, hal ini mengarah pada peningkatan presisi dan pengurangan ambiguitas ketika menentukan perilaku yang diperlukan suatu sistem. Kedua, hal ini mengurangi kesenjangan abstraksi antara konsep dan bahasa pakar domain yang bertanggung jawab untuk menentukan perilaku yang diperlukan sistem dan pengembang perangkat lunak yang harus membangunnya. Hasilnya adalah spesifikasi fungsional yang secara jelas dan jelas menentukan perilaku fungsional yang diperlukan dan dapat dimengerti oleh semua pemangku kepentingan produk, pakar domain, dan pengembang.

Otomatisasi ditingkatkan terutama melalui pembuatan kode otomatis, mengurangi upaya pemrograman manual dan dengan demikian menghemat biaya dan mencegah injeksi cacat selama pemrograman. Otomatisasi lebih lanjut dapat dilakukan dengan menggunakan simulasi dan pembuatan kasus uji otomatis, yang meningkatkan cakupan pengujian dan mengurangi biaya pengujian.

Penerapan teknik verifikasi formal di industri untuk menetapkan bahwa perilaku fungsional kode yang dihasilkan dari suatu model sebagaimana dimaksud telah menimbulkan masalah. Ada beberapa platform MDD yang memberikan verifikasi otomatis sepenuhnya untuk perangkat lunak, dan pada saat tulisan ini dibuat, hanya platform Analytical Software Design (ASD) Verum yang dapat melakukan ini secara otomatis, tanpa pengetahuan khusus dan untuk sistem perangkat lunak tertanam berskala besar dengan perilaku bersamaan yang ekstensif.Jika alat tersebut tersedia dan dapat diterapkan, verifikasi otomatis yang dilakukan pada model fungsional dapat mengurangi cacat ke tingkat yang belum pernah terjadi sebelumnya dengan sangat cepat dan memberikan penghematan waktu dan biaya yang signifikan; ini secara langsung mengatasi 40% upaya pengembangan yang saat ini dihabiskan untuk menguji dan men-debug perangkat lunak. Kita kembali ke topik ini nanti di bagian ini.

3.8.2 Model yang Dapat Dieksekusi

Model harus dapat dieksekusi; diskon platform MDD mana pun jika hal ini tidak terjadi.

Selain cukup ketat dan tepat untuk memungkinkan pembuatan kode dan pembuatan kasus uji, model yang dapat dieksekusi dapat dieksekusi pada sistem pengembangan melalui simulasi di awal siklus hidup pengembangan.

Model yang dapat dieksekusi memberikan manfaat signifikan berikut ini:

1. Memberikan umpan balik yang cepat dan dini mengenai persyaratan dan spesifikasi dan berfungsi untuk memvalidasi bahwa produk yang “tepat” sedang dibuat. Inilah yang dimaksud sebagian besar vendor MDD ketika mereka menggunakan istilah verifikasi.
2. Izinkan pengujian fungsional dilakukan pada host pengembangan tanpa memerlukan akses ke perangkat lunak yang berjalan pada mesin target akhirnya. Hal ini sangat penting ketika perangkat keras dan perangkat lunak target dikembangkan secara paralel.

Dua contoh terkenal dari platform tersebut adalah produk IBM Rational Rhapsody dan Mentor Graphics BridgePoint, keduanya menggunakan bahasa pemodelan berbasis *Unified Modeling Language* (UML). Hal ini akan dibahas kemudian secara lebih rinci.

Model yang dapat dieksekusi belum tentu dapat diverifikasi secara formal, dan sebagian besar platform MDD tidak menyediakan fasilitas untuk memverifikasi secara formal kebenaran perangkat lunak yang sedang dirancang. Platform ini, tentu saja, memastikan bahwa model terbentuk dengan baik dalam arti yang sama seperti kompiler memastikan suatu program sesuai dengan sintaksis dan tata bahasa bahasa pemrograman.

Model Eksekusi yang Dapat Diverifikasi Secara Formal

Selain dapat dieksekusi, beberapa platform MDD menghasilkan model yang dapat diverifikasi secara formal. Artinya, platform MDD menyertakan alat untuk verifikasi formal

otomatis berdasarkan prinsip matematika. Simulasi, seperti halnya pengujian, adalah salah satu bentuk pengambilan sampel. Sampel adalah kumpulan kasus uji, atau contoh skenario, dan populasi adalah kumpulan semua kemungkinan jalur eksekusi melalui perangkat lunak. Dalam praktiknya, untuk pembangunan apa pun selain pembangunan yang sangat kecil, ukuran sampel secara statistik tidak signifikan dibandingkan dengan ukuran populasi. Seperti yang pernah dikatakan oleh E. W. Dijkstra, "pengujian program dapat digunakan untuk menunjukkan adanya bug, namun tidak pernah untuk menunjukkan ketidakhadirannya!"[48]

Sebaliknya, verifikasi formal menggunakan teknik matematika untuk menghitung dan memeriksa setiap jejak perilaku yang mungkin terjadi dalam suatu desain untuk menentukan apakah kumpulan properti yang ditentukan pengguna berlaku atau tidak dalam setiap keadaan yang mungkin terjadi. Jika kumpulan properti cukup besar dan lengkap dalam arti mencakup semua aspek dalam spesifikasi perangkat lunak, maka verifikasi formal menetapkan apakah suatu desain benar atau tidak sehubungan dengan spesifikasinya. Hal ini serupa dengan metode verifikasi formal yang tersedia di beberapa platform otomatisasi desain elektronik (EDA) untuk memverifikasi kekayaan intelektual desain.

Dalam praktiknya, menerapkan verifikasi formal pada desain perangkat lunak terbukti sulit. Ada dua tantangan besar yang harus diatasi:

1. Kemampuan untuk memperoleh kumpulan properti dari spesifikasi dan menentukannya dalam beberapa notasi formal, misalnya logika temporal
2. Skalabilitas

Mendapatkan serangkaian properti lengkap untuk diverifikasi sedemikian rupa sehingga semua aspek penting dari spesifikasi perangkat lunak tercakup adalah pekerjaan yang sulit dan terampil. Mengekspresikan properti dalam notasi matematika formal seperti logika temporal juga memerlukan keahlian yang tidak dimiliki sebagian besar pengembang perangkat lunak tertanam dan menghasilkan spesifikasi yang tidak dapat diakses oleh sebagian besar pemangku kepentingan proyek. Oleh karena itu, umumnya tidak mungkin untuk meninjaunya bersama dengan pakar domain dan menetapkan bahwa interpretasi tersebut benar terhadap spesifikasi perangkat lunak. Bahkan untuk beberapa platform MDD yang menyediakan semacam verifikasi formal, penggunaannya masih menantang dan tidak meluas.

Nilai dari verifikasi formal adalah bahwa ia secara simbolis memeriksa seluruh ruang keadaan dari desain perangkat lunak untuk menentukan apakah properti yang ditentukan berlaku atau tidak pada semua masukan yang mungkin. Dalam praktiknya, hal ini jarang dilakukan; hampir semua sistem perangkat lunak yang berguna terlalu rumit untuk diverifikasi secara formal secara keseluruhan. Platform MDD yang berbeda menggunakan pendekatan yang berbeda terhadap masalah ini. Beberapa platform MDD membatasi kelas desain yang dapat diverifikasi; misalnya, SCADE Suite (*Esterel Technologies*) menangani desain yang deterministik dan sinkron. ASD (*Verum Software Technologies BV*) memberikan pendekatan berbeda yang disebut verifikasi komposisional dimana keseluruhan sistem diverifikasi komponen demi komponen sedemikian rupa sehingga sifat yang telah terbukti tetap berlaku ketika komponen disusun untuk membentuk sistem yang lengkap. Pendekatan ini

[48] Dijkstra EW. Notes on Structured Programming. Technological University Eindhoven, Department of Mathematics. The Netherlands; August 1969

memungkinkan desain yang sepenuhnya konkuren dan asinkron diverifikasi secara formal, baik untuk kepatuhan terhadap properti yang ditentukan dan juga untuk menunjukkan tidak adanya kesalahan desain konkuren dan asinkron seperti kebuntuan, live lock, dan kondisi balapan. Simulasi dan pengujian tidak efektif dalam menghilangkan kesalahan tersebut.

Verifikasi formal tidak boleh dilihat sebagai alternatif terhadap verifikasi berdasarkan simulasi dan pengujian; sebaliknya, ini saling melengkapi.

Verifikasi formal dengan alat yang dirancang dengan baik memberikan sejumlah manfaat signifikan:

1. Memungkinkan umpan balik awal mengenai kebenaran desain komponen tanpa memerlukan kasus pengujian apa pun untuk ditentukan dan sebelum kode apa pun dibuat. Sebuah komponen dapat diverifikasi secara formal sebelum komponen apa pun yang digunakannya ada.
2. Memberikan umpan balik awal mengenai kerusakan sebelum kode dibuat.
3. Tidak memerlukan simulasi sehingga tidak perlu menyiapkan set tes.
4. Dapat sepenuhnya menggantikan atau secara signifikan mengurangi kebutuhan pengujian tingkat komponen (unit).
5. Mengurangi upaya integrasi dan validasi karena perangkat lunak memasuki fase tersebut dengan tingkat kerusakan yang sangat rendah.

Nilai manfaat ini meningkat dengan cepat seiring dengan berkembangnya skala dan kompleksitas perangkat lunak. Misalnya, di awal bab ini, kita telah membahas manfaat pengujian cakupan kode secara menyeluruh. Produk sampingan dari verifikasi model formal adalah cakupan beberapa kondisi yang lengkap diselesaikan secara otomatis.

3.8.3 Bahasa Pemodelan

Model harus diekspresikan dalam bahasa pemodelan dengan tata bahasa dan semantik yang didefinisikan secara formal yang mampu mengekspresikan struktur statis dan perilaku dinamis pada tingkat abstrak yang dikeluarkan dari domain pemrograman. Kita dapat membagi bahasa-bahasa ini menjadi dua kelompok:

- **Bahasa khusus vendor**: Bahasa yang dikembangkan dan dipromosikan oleh vendor tertentu dari platform MDD seperti MatLab dan Simulink dari MathWorks, Esterel dari Esterel Technologies, dan bahasa ASD yang digunakan dalam ASD:Suite Verum Software Technologies.
- **Bahasa terstandar**: Bahasa yang ditentukan oleh kelompok industri pengguna yang berminat dan vendor platform MDD dan paling umum didasarkan pada Unified Modeling Language (UML).

Dalam praktiknya, perbedaan yang signifikan antara bahasa pemodelan khusus vendor dan bahasa berbasis UML yang tidak bergantung pada vendor, tentu saja, adalah tingkat penguncian vendor yang dihasilkan dari penggunaannya. Bahasa berbasis UML memiliki tujuan yang patut dipuji yaitu menjadi independen terhadap vendor dan, pada prinsipnya, menawarkan kemungkinan pertukaran model antar platform. Bahasa khusus vendor umumnya tidak memfasilitasi pertukaran model antar platform.

Bahasa pemodelan berbasis UML semakin banyak diadopsi oleh platform MDD dan alat lain dari beberapa vendor besar dan distandarisasi. Karena meningkatnya adopsi oleh vendor platform MDD, bahasa pemodelan berbasis UML dibahas lebih lengkap di bagian

berikut. Beberapa bahasa khusus vendor akan dibahas secara singkat nanti bersama dengan platform MDD yang mendukungnya.

Bahasa Pemodelan Terpadu

Unified Modeling Language, atau UML, adalah bahasa pemodelan berbasis grafis dengan tujuan umum yang muncul dari bidang pemrograman berorientasi objek dan dimaksudkan untuk mendeskripsikan perangkat lunak. *Object Management Group* (OMG) menstandardisasi UML, dan spesifikasinya tersedia secara bebas untuk diunduh di Internet publik.[49] Platform MDD semakin banyak menggunakan bahasa pemodelan yang berasal dari atau sangat terkait dengan UML.

Standar UML mengatur representasi grafis UML.
Karena UML distandarisasi, platform MDD berbasis UML cenderung menampilkan modelnya dengan cara yang serupa, sehingga memudahkan pengguna untuk memahami dan menggunakan platform MDD dari vendor yang berbeda.

Standar UML juga menetapkan format yang dapat dibaca mesin dalam bentuk file XML. Harapannya adalah hal ini akan menghasilkan model yang dapat dipertukarkan antar vendor platform MDD. UML versi 2.0 mendefinisikan 12 tipe diagram berbeda beserta elemen grafisnya. Ada 5 tipe diagram struktural dan 7 tipe diagram perilaku.

Diagram struktural mewakili elemen sistem dan hubungan struktural tingkat kode statis di antara mereka. Jenis diagram struktural ini adalah:

(1) **Diagram kelas**: Kelas adalah entitas waktu desain yang mewakili konsep dasar dalam suatu sistem dan hubungan di antara mereka. Kelas mewakili entitas runtime yang disebut objek; setiap objek adalah turunan dari beberapa kelas. Diagram kelas digunakan untuk memodelkan hubungan statis dan partisi fungsional suatu sistem ke dalam komponen-komponennya.

(2) **Diagram paket**: Paket adalah wadah serba guna untuk mengelompokkan elemen UML terkait, termasuk paket lainnya. Mereka biasanya digunakan untuk mengatur diagram kelas, meskipun mereka dapat mengelompokkan apa saja. Arti pengelompokan bergantung pada model (dan pemodel). Paket memainkan peran penting dalam mempartisi dan mengatur model sistem besar karena menyediakan mekanisme pelingkupan untuk nama; setiap paket mewakili namespace yang berbeda.

(3) **Diagram struktur komposit**: Saat memodelkan sistem besar, seseorang sering kali mengulangi hubungan kompleks seperti model entitas dunia nyata atau pola desain yang dapat diulang; ini disebut struktur komposit dan mewakili bagian-bagian yang dapat digunakan kembali dalam suatu model.

(4) **Diagram komponen**: Ini mirip dengan diagram kelas tetapi alih-alih berhubungan dengan konsep desain, diagram ini mewakili hubungan antara bagian fisik dari suatu sistem yang disebut komponen.

(5) **Diagram penerapan**: Diagram ini menunjukkan disposisi fisik sistem yang kompleks antara node pemrosesan fisik yang berbeda dalam lingkungan dunia nyata.

Diagram perilaku menangkap variasi interaksi dan keadaan sesaat dalam suatu model saat model dijalankan sepanjang waktu, melacak bagaimana sistem akan bertindak dalam

[49] Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1; August 2011.

lingkungan dunia nyata, dan mengamati dampak dari suatu operasi atau peristiwa, termasuk hasilnya. Tujuh jenis diagram perilaku adalah:

a) **Diagram kasus penggunaan**: Diagram ini digunakan untuk memodelkan interaksi pengguna/sistem. Mereka mendefinisikan perilaku, persyaratan, dan batasan dalam bentuk skrip atau skenario.

b) **Diagram interaksi**: Jenis ini sebenarnya adalah kumpulan diagram yang berfokus pada interaksi antara elemen sistem yang dimodelkan. Anggota keluarga adalah:
   - Diagram urutan: Diagram ini merupakan diagram yang paling umum dan paling familiar bagi sebagian besar pengembang. Mereka menunjukkan aliran kontrol sebagai urutan interaksi antara elemen sistem, biasanya antara contoh komponen.
   - Diagram komunikasi: Diagram komunikasi cenderung menekankan ketergantungan kontrol antar elemen sistem dibandingkan urutan terjadinya interaksi.
   - Diagram ikhtisar interaksi: Ini adalah diagram interaksi yang disederhanakan dan merupakan persilangan antara diagram urutan dan diagram aktivitas. Kegunaannya adalah untuk merangkum keseluruhan aliran kendali melalui sistem yang dimodelkan.
   - Diagram waktu: Diagram ini fokus pada perilaku interaksi waktu tertentu. Mereka digunakan untuk mewakili aliran peristiwa berbasis waktu dalam sistem dan untuk memodelkan perilaku real-time.

c) **Diagram diagram keadaan**: Ada dua jenis diagram ini: yang pertama digunakan untuk memodelkan perilaku evolusi elemen sistem, biasanya komponen, sebagai mesin keadaan terbatas. Yang lainnya digunakan untuk memodelkan keadaan suatu protokol antara berbagai elemen sistem.

d) **Diagram aktivitas**: Diagram ini digunakan untuk mewakili interaksi antara proses yang terjadi secara bersamaan. Perilaku keadaan entitas sistem yang berinteraksi direpresentasikan dalam "jalur berenang" dan interaksi di antara mereka. Diagram ini menunjukkan bagaimana entitas sistem, biasanya komponen, berjalan secara paralel satu sama lain dan titik sinkronisasi di antara mereka. Perilaku keadaan yang ditampilkan dalam diagram aktivitas biasanya merupakan abstraksi dari perilaku keadaan yang ditunjukkan dalam bagan keadaan individu dan mewakili perilaku keadaan yang terlihat secara eksternal antar entitas. Titik sinkronisasi direpresentasikan sebagai operasi "fork" dan "join".

UML mempunyai keuntungan karena UML dikenal secara luas, meskipun tidak secara universal, di seluruh komunitas pengembangan perangkat lunak. Orientasi grafis UML membuatnya menarik bagi pengguna dan vendor platform MDD. Selain itu, ada banyak sumber daya yang tersedia seperti buku dan kursus pelatihan untuk membantu pendatang baru memulai. Misalnya, pembaca diarahkan ke website OMG (www.omg.org).

UML memiliki kelemahan karena lemah dalam menggambarkan perilaku fungsional dan cenderung menyajikan dunia dalam konsep-konsep dari domain pemrograman berorientasi objek. Penggunaan istilah dan konsep seperti kelas, objek, agregasi, dan pewarisan menyulitkan non-pemrogram untuk memahami secara pasti apa yang ditentukan.

Hal ini bertentangan dengan salah satu keuntungan utama pendekatan MDD, yaitu penggunaan model yang lebih abstrak yang dinyatakan dalam istilah spesifik untuk produk yang sedang dikembangkan membantu komunikasi dan pemahaman bersama antara pemangku kepentingan yang berbeda dalam lingkungan pengembangan produk, termasuk pelanggan, insinyur dari berbagai negara. disiplin ilmu, pakar domain, dan pengembang perangkat lunak. Pengembang perangkat lunak yang tidak terbiasa pemrograman dalam bahasa berorientasi objek seperti C++ juga cenderung menemukan konsep yang sulit untuk dipahami pada awalnya; sebagian besar perangkat lunak tertanam masih dikembangkan dalam C, dan sebagian besar pengembang perangkat lunak tertanam cenderung berasal dari latar belakang tersebut.

Kesulitan dalam mendeskripsikan secara tepat perilaku fungsional yang diperlukan mengakibatkan vendor alat cenderung mendefinisikan varian UML mereka sendiri untuk mengatasi kesenjangan ini.

Bahasa Pemodelan Sistem

UML menyediakan mekanisme ekstensi yang disebut Profil, yang memungkinkan UML disesuaikan dan diperluas untuk membentuk varian bahasa pemodelan khusus domain dan/atau platform yang terlihat sebagai anggota keluarga UML. Salah satu varian UML disebut System Modeling Language, atau SysML. Awalnya dimulai sebagai proyek sumber terbuka dan sekarang distandarisasi oleh OMG, SysML adalah bahasa pemodelan tujuan umum untuk rekayasa sistem. Juga berbasis grafis, SysML mendukung spesifikasi, analisis, desain, verifikasi, dan validasi sistem kompleks yang mencakup perangkat keras dan perangkat lunak.

Sebagai bahasa pemodelan untuk tujuan rekayasa sistem, SysML menawarkan peningkatan berikut dibandingkan UML:

1. SysML kurang berpusat pada perangkat lunak. Ini mendukung pemodelan persyaratan, struktur, perilaku, dan batasan untuk memberikan gambaran yang kuat tentang sistem perangkat keras/perangkat lunak yang kompleks, komponennya, dan lingkungannya. Dibandingkan dengan UML, SysML tidak terlalu bergantung pada konsep pemrograman berorientasi objek untuk merepresentasikan ide; hal ini memudahkan non-programmer dan insinyur dari disiplin ilmu lain untuk belajar dan memahami.
2. SysML adalah bahasa yang lebih kecil dari UML. Jumlah total tipe diagram dikurangi menjadi sembilan, tujuh di antaranya digunakan kembali dari UML dan dua di antaranya, diagram persyaratan dan diagram parameter, merupakan hal baru di SysML untuk mendukung manajemen persyaratan, penelusuran persyaratan, dan analisis kuantitatif.
3. SysML memiliki semantik yang lebih fleksibel dan ekspresif, memungkinkan pemodelan perilaku yang lebih baik. Ini juga mencakup representasi berbasis fungsi, berbasis pesan, dan berbasis negara.
4. SysML memberikan dukungan yang lebih luas dan fleksibel untuk notasi tabel dalam bentuk tabel alokasi. Tabel alokasi menyediakan sistem pendukung untuk mempartisi dan mengalokasikan perilaku, struktur, dan batasan fisik dan kinerja antar elemen sistem. Hal ini sangat berguna selama fase desain arsitektur ketika suatu sistem dipartisi menjadi subsistem dan komponennya.

Sebagai konsekuensinya, vendor platform MDD semakin banyak yang mengadopsi SysML. Pengenalan yang baik tentang SysML adalah buku Panduan Praktis untuk SysML.[50]

UML yang dapat dieksekusi

Varian UML lain yang ditentukan melalui mekanisme Profil UML disebut UML yang dapat dieksekusi, atau xtUML. xtUML adalah bahasa grafis yang diperluas dengan bahasa tindakan objek non-grafis untuk menentukan perilaku objek pada abstraksi tingkat tinggi tetapi dengan presisi yang cukup untuk memungkinkan eksekusi model, simulasi, dan pembuatan kode.

Model xtUML menggunakan diagram dari UML. Dalam beberapa kasus, ia menerapkannya pada konsep yang berbeda atau memiliki versinya sendiri. Kumpulan diagram inti adalah Bagan domain: Ini pada dasarnya adalah diagram paket UML di mana paket-paket tersebut mewakili area fungsionalitas terpisah, yang disebut domain, dalam suatu sistem. Bagan domain mengidentifikasi domain dan hubungan di antara domain tersebut.

- ※ **Diagram kelas**: Diagram ini digunakan untuk mempartisi domain menjadi entitas waktu desain yang mewakili konsep dasar dalam domain dan hubungan di antara mereka. Entitas runtime terkait adalah objek yang dibuat dari kelas.
- ※ **Diagram bagan keadaan**: Diagram ini adalah versi sederhana dari diagram bagan keadaan UML. Dalam xtUML, mesin status yang dijelaskan oleh diagram diagram status xtUML mendefinisikan perilaku setiap kelas.
- ※ **Spesifikasi tindakan**: Ini bukan diagram melainkan formulir entri yang melaluinya perilaku objek dapat ditentukan menggunakan bahasa tindakan objek xtUML.

Diagram UML lainnya seperti diagram kasus penggunaan, urutan, kolaborasi, dan aktivitas juga dapat digunakan kembali dengan cara yang mudah.

Bahasa tindakan objek xtUML mendefinisikan perilaku dengan cukup detail untuk memungkinkan eksekusi. Bahasa ini adalah bahasa tindakan objek (OAL) yang sesuai dengan OMG. Ini berbeda dari bahasa pemrograman konvensional seperti C++ atau Java karena menggambarkan perilaku pada tingkat yang lebih tinggi, mengabstraksi bahasa pemrograman tertentu dan keputusan rinci tentang bagaimana perangkat lunak diatur. Bahasa tindakan objek tidak membuat keputusan pengkodean; itu tidak membuat pernyataan tentang struktur penugasan, distribusi, kelas, atau enkapsulasi. Sebaliknya, bahasa tindakan objek hanya menjelaskan data dan perilaku, memungkinkan model xtUML diterapkan di berbagai lingkungan perangkat lunak tanpa perubahan.

Secara konseptual, setiap objek yang ditentukan dalam model xtUML dieksekusi secara bersamaan dan asinkron terhadap objek lainnya. Setiap objek sedang menjalankan suatu prosedur atau menunggu sesuatu untuk memicu eksekusinya. Urutan peristiwa ditentukan per objek; tidak ada konsep waktu global atau universal. Semua sinkronisasi yang diperlukan antar objek harus dimodelkan secara eksplisit.

Tugas memetakan struktur konseptual ini ke platform target tertentu diberikan kepada kompiler model. Kompiler model tidak hanya menghasilkan kode yang dapat dieksekusi dalam bahasa pemrograman yang dipilih, tetapi juga menyediakan pemetaan runtime ke runtime target lingkungan dengan cara yang melestarikan semantik. Pada prinsipnya, dan bergantung

[50] Friedenthal S, Moore A, Steiner R. A Practical Guide to SysML: The Systems Modeling Language. Burlington, MA: Elsevier; 2008.

pada ketersediaan kompiler model yang sesuai, suatu model dapat dikompilasi menjadi implementasi yang dirancang untuk dieksekusi pada sistem terdistribusi menggunakan CORBA, pada sistem tertanam berukuran kecil menggunakan C dan ditargetkan untuk silikon "telanjang" (tanpa sistem operasi), atau pada sistem kompleks yang menggunakan C++ atau Java yang ditargetkan untuk platform multiprosesor yang menggunakan sistem operasi real-time seperti INTEGRITY atau sistem operasi tujuan umum seperti Linux atau Windows tertanam.

xtUML adalah perwujudan metode desain perangkat lunak Shlaer-Mellor yang menjadi dasar platform Mentor Graphics BridgePoint. Pengantar xtUML yang sangat baik dapat ditemukan di UML yang Dapat Dieksekusi, Landasan untuk Arsitektur Berbasis Model.[51]

3.8.4 Jenis Platform MDD

Penerapan praktis MDD memerlukan penggunaan platform MDD: seperangkat alat pengembangan dan kerangka waktu proses yang mencakup hal-hal berikut:

1. Alat pengeditan untuk membuat dan mengedit model.
2. Memeriksa alat untuk memeriksa kelengkapan model dan memastikan model tersebut terbentuk dengan baik dan bermakna.
3. Alat untuk memverifikasi perilaku yang dirancang, selama fase proyek spesifikasi dan desain; misalnya, dengan mensimulasikan eksekusi runtime, metode verifikasi formal otomatis, atau keduanya.
4. Generator kode untuk menghasilkan kode sumber yang siap kompiler, berkualitas baik, dan efisien sesuai dengan perilaku fungsional dalam model. Sebagian besar platform MDD menyediakan generator kode untuk C dan C++. Bahasa lain seperti Java dan Ada juga tersedia dari beberapa vendor.
5. Alat untuk meningkatkan efisiensi pengujian; misalnya, mengotomatiskan pembuatan kasus pengujian, pelaksanaan pengujian, dan analisis hasil.
6. Kerangka runtime yang memungkinkan kode yang dihasilkan berjalan dengan baik di berbagai lingkungan eksekusi yang berbeda.

Perangkat lunak tertanam menjalankan beragam fungsi di berbagai domain teknik. Skala dan kompleksitas perangkat lunak berkisar dari sensor sederhana, antarmuka I/O, kontrol gerak, pemrosesan gambar, dan kontrol mesin pengawasan pada masing-masing mesin di satu ujung skala hingga kontrol pengawasan seluruh pabrik di sisi lain.

Agar MDD berhasil diterapkan, platform MDD yang digunakan harus sesuai tidak hanya dengan skala dan kompleksitas sistem yang sedang dikembangkan tetapi juga dengan domain teknik sistem. Masalah yang muncul dalam sistem yang kompleks adalah banyaknya domain teknik yang ditemui, dan tidak ada satu platform MDD yang mencakup semuanya.

Untuk mengilustrasikan hal ini, pada bagian berikut, kami menggunakan sistem pemindaian patologi digital modern berkecepatan tinggi yang dibuat oleh produsen peralatan kesehatan dan pencitraan medis terkemuka sebagai contoh untuk menunjukkan bagaimana bagian-bagian berbeda dari sistem yang kompleks memerlukan pendekatan atau platform yang berbeda.

---

[51] Mellor SJ, Balcer MJ. Executable UML, A Foundation for Model-Driven Architecture. Reading, MA: Addison-Wesley; 2002

3.8.5 Studi Kasus: Pemindai Patologi Digital

Tujuan pemindai patologi digital dalam studi kasus ini adalah untuk membuat gambar berwarna berdefinisi tinggi dan beresolusi tinggi dari sampel jaringan manusia dan menyimpannya dalam sistem pengarsipan dan komunikasi gambar standar (PACS) untuk kemudian ditinjau oleh spesialis klinis (lihat Gambar 3.6). Kualitas gambar setidaknya harus sama dengan mikroskop optik terbaik dan harus sesuai untuk pemeriksaan diagnostik berbantuan komputer selanjutnya. Sampel jaringan disiapkan untuk diperiksa dan disajikan ke perangkat pada slide mikroskop kaca; setiap slide dapat berisi beberapa sampel. Slide ditempatkan di rak, dan perangkat ini memiliki dua lengan robot dengan pegangan untuk memindahkan slide antara rak slide dan bagian pemindaian perangkat. Selama pemindaian, slide dipindahkan pada tingkat presisi di bawah tiga kamera pemindai jalur tetap serta optik dan sumber penerangan terkait. Panggung bergerak dalam tiga sumbu; dua sumbu horizontal digunakan untuk memindahkan sampel di bawah kamera selama pemindaian, dan sumbu vertikal digunakan untuk menyesuaikan ketinggian sampel di bawah kamera guna memastikan fokus terbaik untuk setiap titik pada gambar. Ketika setiap gambar selesai, gambar tersebut dikirim melalui koneksi jaringan ke sistem PACS untuk penyimpanan dan analisis selanjutnya. Karena gambar yang dihasilkan sangat besar, perangkat dapat mencapai target throughputnya hanya dengan melakukan tindakan penanganan slide, pemindaian, pemrosesan gambar, dan transmisi gambar secara paralel.

Gambar 3.6 Pemindai patologi digital.

Perangkat lunak tertanam yang mengendalikan mesin tersebut meliputi

1. Perangkat lunak kontrol gerak loop tertutup untuk mengendalikan lengan robot dan tahap presisi; misalnya, ini mencakup perangkat lunak yang diperlukan untuk pengontrol proporsional-integral-derivatif (PID) motor stepper.
2. Perangkat lunak kalibrasi untuk robot, panggung, dan akuisisi gambar.
3. Perangkat lunak pengolah gambar untuk mengaktifkan ekstraksi fitur dan pemfokusan real-time berkelanjutan.
4. Perangkat lunak komunikasi yang menyediakan akses jarak jauh ke server gambar dan menerapkan protokol berlapis-lapis yang ditentukan oleh standar PACS.
5. Perangkat lunak kontrol mesin pengawas untuk menyatukan semuanya dan mengawasi perilaku robot secara bersamaan, pemindaian gambar, pemrosesan gambar, transmisi data, dan pelaksanaan perintah operator.
6. Antarmuka pengguna grafis (GUI) modern untuk operator.

Arsitektur perangkat lunak dari sistem sampel dibagi menjadi tiga lapisan utama sebagai berikut:

1. Kontrol Gerak dan Algoritma
2. Pengawasan Pengendalian Mesin
3. Antarmuka Pengguna Grafis (GUI)

Lapisan Kontrol Gerakan dan Algoritma terdiri dari perangkat lunak yang mengimplementasikan algoritme kontrol loop tertutup untuk pengontrol gerakan bersama dengan algoritme pemrosesan gambar dan fokus otomatis serta beberapa kode antarmuka abstraksi perangkat keras yang memungkinkan akses ke antarmuka dan sinyal perangkat keras.

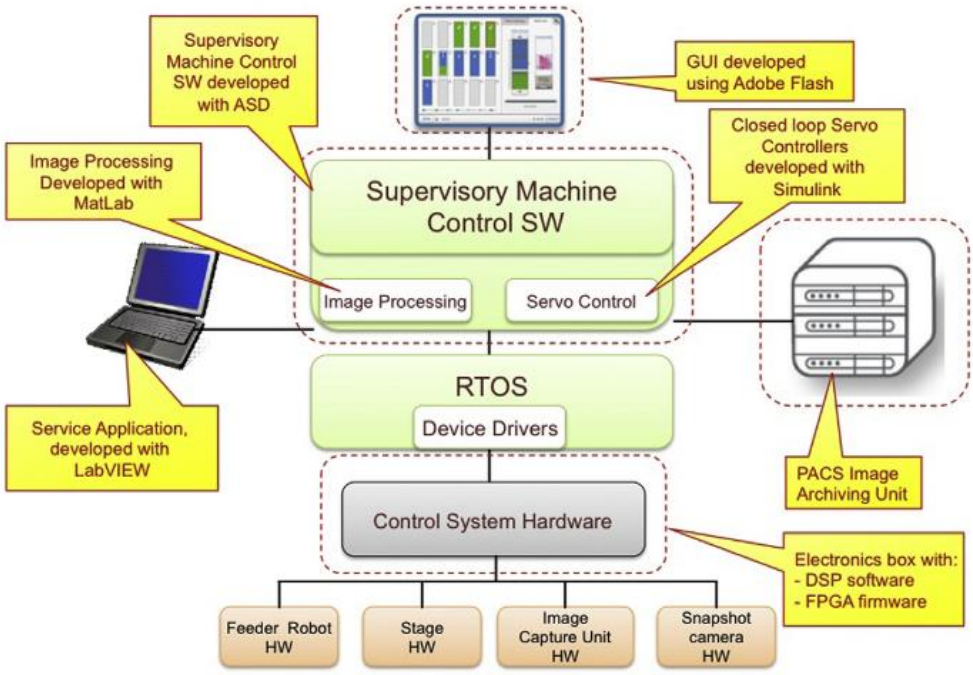
Lapisan Kontrol Mesin Pengawas adalah bagian inti dari mesin. Dibangun pada lapisan Kontrol Gerak dan Algoritma, perangkat lunak ini secara progresif menyadari abstraksi yang diperlukan yang menjauh dari domain implementasi dan pemrograman menuju domain pengguna akhir dan aplikasi dan mengimplementasikan sistem yang lengkap namun “tanpa kepala”. Semua fungsionalitas hadir, termasuk semua yang diperlukan untuk mendukung GUI bagi operator, tetapi tidak termasuk GUI itu sendiri.

Dalam sistem sampel, lapisan GUI diimplementasikan dalam ruang proses terpisah untuk ketahanan dan untuk memfasilitasi pengembangannya dengan mensubkontrakkannya ke pihak ketiga yang memiliki keahlian desain antarmuka manusia khusus. Ini diimplementasikan menggunakan Adobe Flash dan digabungkan ke lapisan Kontrol Mesin Pengawas melalui CORBA *Object Request Broker* (ORB) yang siap pakai.

Seperti halnya sistem yang kompleks, berbagai teknologi diperlukan untuk mewujudkan mesin yang lengkap, dan perangkat lunaknya merupakan campuran komponen yang dibuat khusus dan komponen siap pakai yang dipasok oleh banyak pemasok. Pendekatan MDD yang dipilih harus mampu mendukung keragaman tersebut. Tidak ada platform MDD tunggal yang cocok untuk semua aspek dari semua jenis desain dan implementasi perangkat lunak tertanam dan mampu mendukung beragam teknologi yang diperlukan dan campuran perangkat lunak yang dibuat khusus, siap pakai, dan lama.

Manfaat yang mungkin diperoleh dari MDD dalam praktiknya bergantung pada kesesuaian antara platform MDD yang dipilih dan sifat perangkat lunak yang dikembangkan. Bersiaplah untuk memadupadankan berbagai platform untuk pengembangan sistem tertanam yang kompleks. Campuran bahasa pemodelan dan lingkungan pengembangan lainnya yang digunakan untuk membangun pemindai patologi digital ditunjukkan pada Gambar 3.7. Setiap komponen dan pilihan sistem pemodelan dijelaskan lebih lanjut pada bagian berikut.

Selain itu, platform MDD yang berbeda memerlukan keahlian yang sangat berbeda untuk menggunakannya, dan keterampilan ini mungkin tidak ada di tim pengembangan Anda, sehingga memerlukan pelatihan ekstensif atau penggunaan spesialis luar yang mahal. Ketahuilah bahwa beberapa vendor platform MDD mempunyai pasokan keterampilan konsultasi ini sebagai bagian penting dari model bisnis mereka, dan beberapa di antaranya adalah perusahaan jasa yang menyamar sebagai perusahaan produk.



Gambar 3.7 Subsistem pemindai patologi digital dan lingkungan pengembangan/pemodelan terkait.

Pengembangan Algoritma

Pemindai patologi digital menerima beberapa sampel jaringan per slide, dan setiap sampel harus dipindai dan disimpan secara terpisah dalam gambar resolusi tinggi. Sebelum pemindaian resolusi tinggi dimulai, sampel jaringan individual harus ditempatkan sesuai wilayah yang diinginkan pada slide dan batasnya diidentifikasi. Hal ini dilakukan dengan mengambil gambar beresolusi rendah dari seluruh slide dan menerapkan algoritma pemrosesan gambar untuk mengidentifikasi wilayah yang berisi sampel.

Pengembangan algoritma pemrosesan gambar ini dilakukan dengan menggunakan lingkungan pemodelan yang dirancang untuk pengembangan algoritma. Biasanya, platform tersebut menyediakan

1. Bahasa pemodelan generasi keempat (biasanya) berbasis teks yang dirancang untuk memodelkan proses matematika dan algoritma semudah mungkin. Misalnya, dalam bahasa MATLAB dari The MathWorks Inc., matriks adalah objek orde pertama, dan serangkaian fungsi dan operator matematika bekerja langsung pada matriks tersebut. Sebuah gambar direpresentasikan sebagai matriks nilai piksel yang berisi data gambar.
2. Eksekusi model interpretatif dikombinasikan dengan cara mudah memvisualisasikan data numerik dan gambar dalam berbagai bentuk untuk memberikan umpan balik langsung selama pengembangan algoritma.
3. Berbagai fasilitas untuk mengimpor nilai data dan berbagai format data dan gambar yang umum digunakan.
4. Kemampuan untuk mencocokkan presisi komputasi dalam model dengan presisi perangkat keras target dan fasilitas untuk mengendalikan perambatan kesalahan numerik.
5. Generator kode mampu menghasilkan kode sumber yang efisien yang dapat dikompilasi untuk mesin target. Eksekusi interpretatif sangat ideal untuk

pengembangan algoritma; efisiensi waktu proses algoritme dalam produk akhir memerlukan pembuatan kode yang dioptimalkan.

Dua contoh platform pengembangan algoritmik yang terkenal adalah platform MATLAB dari The MathWorks Inc. dan platform LabVIEW dari National Instruments Corp. Ada penawaran dari vendor lain dan juga beberapa tersedia sebagai open source.

Kontrol Gerak dan Pemodelan Sistem Dinamis

Pemindai patologi digital mencakup dua lengan robot dengan pegangan untuk memindahkan slide masuk dan keluar dari perangkat dan tahap presisi tiga sumbu untuk memindahkan slide di bawah kamera pemindaian saluran tetap selama pemindaian. Algoritme kontrol gerak yang diperlukan untuk kontrol loop tertutup dirancang menggunakan platform pemodelan sistem dinamis yang khusus untuk tujuan ini. Biasanya, platform tersebut meliputi

1. Antarmuka pengguna grafis untuk pemodelan visual dengan serangkaian blok pemodelan standar yang dapat diperluas. Model biasanya dikembangkan dalam bentuk diagram blok aliran data menggunakan editor grafis interaktif.
2. Pemodelan khusus domain yang ditujukan untuk insinyur kontrol yang mengembangkan kontrol gerak atau aplikasi kontrol loop tertutup lainnya.
3. Eksekusi model interpretatif dikombinasikan dengan cara mudah memvisualisasikan data numerik dan gambar dalam berbagai bentuk untuk memberikan umpan balik langsung selama pengembangan algoritma.
4. Perpustakaan perangkat lunak driver untuk berinteraksi dengan instrumen umum dan perangkat I/O.
5. Generator kode mampu menghasilkan kode sumber yang efisien. Ini digunakan untuk simulasi berkecepatan tinggi yang efisien selama pengembangan algoritma dan untuk menghasilkan versi yang dapat diterapkan pada perangkat keras target dalam produk akhir.

Dua contoh terkenal dari platform tersebut adalah Simulink dari The MathWorks Inc. dan LabVIEW dari National Instruments Corp.

Perangkat Lunak Lapisan Kontrol Mesin Pengawas

Perangkat lunak kontrol mesin pengawas biasanya merupakan tempat di mana platform MDD yang lebih umum seperti IBM Rational Rhapsody, Mentor Graphics BridgePoint, dan Verum Software Technologies ASD akan diterapkan. Mereka digunakan untuk mempartisi perangkat lunak ke dalam berbagai subsistem, komponen, dan komponen; menentukan antarmuka komponen; dan menentukan perilaku fungsional dan desain detail komponen.

Perangkat lunak kontrol mesin pengawas dalam sistem sampel terdiri dari 350.000 pernyataan C++ yang dapat dieksekusi. Dibandingkan dengan sistem pencitraan medis yang kompleks, mesin manufaktur, dan peralatan telekomunikasi, ini bukanlah aplikasi yang sangat besar. Menurut VDC,[52] jumlah rata-rata perangkat lunak dalam sistem tertanam yang kompleks kini melebihi satu juta pernyataan bahasa C yang dapat dieksekusi.

Pemindai patologi dirancang untuk memenuhi target keluaran yang ketat; memang, inilah salah satu keunggulan unik produk ini dibandingkan kompetitornya. Untuk mencapai kinerja ini, pemindai dirancang dari awal hingga tumpang tindih dan menjalankan sebanyak

[52] VDC Research. 2011 Embedded Engineer Survey; 2011.

mungkin tahapan operasi yang berbeda secara bersamaan. Oleh karena itu, desain perangkat lunak kendali mesin pengawas memiliki karakteristik sebagai berikut:

- ※ **Event didorong**: Sistem harus menangani semua kejadian yang masuk dalam setiap urutan yang mungkin terjadi.
- ※ **Reaktif:** Sistem harus selalu bereaksi terhadap kejadian yang masuk, termasuk yang berasal dari operator mesin. Tidak ada periode "mati" ketika peristiwa yang masuk diabaikan atau hilang.
- ※ **Asinkron dan bersamaan:** Untuk memenuhi target throughput sistem, perangkat lunak kontrol mesin pengawas harus menjadwalkan sebanyak mungkin fase pemrosesan dan pergerakan mekanis secara paralel.

Semua karakteristik ini menghadirkan tantangan berat bagi perancang dan verifikasi berbasis pengujian.

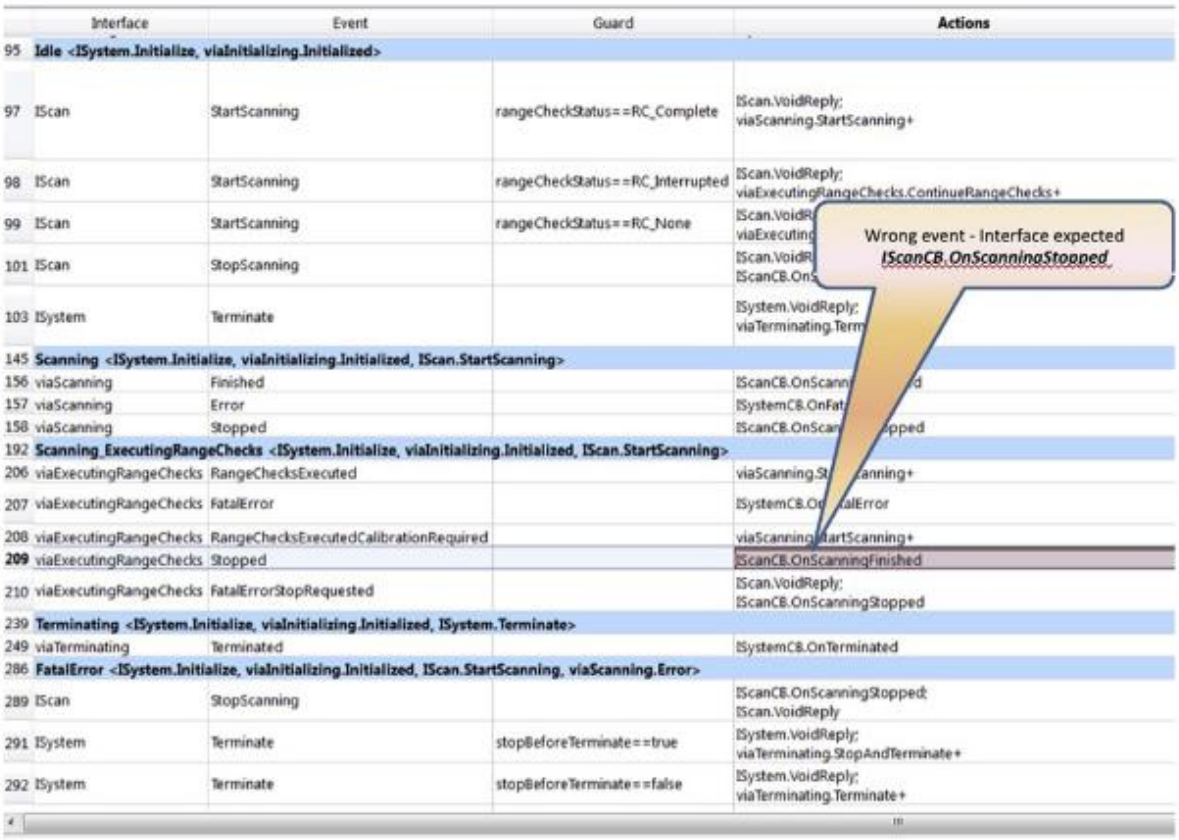
Tren modern dalam desain chip menyediakan peningkatan kapasitas pemrosesan melalui solusi multi-core dibandingkan solusi single-core dengan peningkatan kecepatan clock. Platform multi-inti menghasilkan konkurensi sejati dengan beberapa thread yang menjalankan aliran instruksi secara bersamaan. Sistem sampel dibangun pada platform multi-inti dan dirancang dari awal untuk memanfaatkannya secara maksimal.

Desain serentak adalah yang paling sulit dibuat dan paling sulit diverifikasi melalui simulasi dan pengujian. Dalam praktiknya, jumlah jalur eksekusi yang disisipkan dari masing-masing thread dan tugas sangatlah besar, dan tidak ada jumlah pengujian yang layak secara ekonomi yang dapat memberikan banyak pernyataan mengenai kebenaran perangkat lunak atau persentase jalur eksekusi (dibandingkan dengan pernyataan yang dapat dieksekusi) sebenarnya. diuji. Perbedaan kinerja eksekusi runtime antara platform yang disimulasikan dan platform target dapat mengubah rangkaian kemungkinan jalur eksekusi yang disisipkan. Oleh karena itu, kesalahan dalam jalur eksekusi yang disisipkan yang hanya terjadi pada sistem target mungkin tidak terdeteksi pada platform yang disimulasikan. Efek serupa terjadi ketika menambahkan instrumentasi pengujian ke perangkat lunak yang diuji; kesalahan hilang ketika pernyataan debug ditambahkan dalam upaya menemukannya.

Situasi inilah yang membuat verifikasi formal lebih bermanfaat dibandingkan dengan pendekatan verifikasi berbasis simulasi atau pengujian saja. Perangkat lunak kontrol mesin pengawas pemindai patologi dikembangkan sepenuhnya menggunakan platform ASD dari Verum Software Technologies. Platform ini dirancang dari awal untuk menggabungkan verifikasi formal sistem skala industri. Model ASD dapat dieksekusi dan diverifikasi secara formal, dan inilah alasan utama memilih ASD. Verifikasi formal tidak mengalami kekurangan dalam simulasi dan pengujian yang dijelaskan sebelumnya; semua jalur eksekusi yang mungkin disisipkan dihitung dan dievaluasi untuk memverifikasi desain perangkat lunak.

Gambar 3.8 menunjukkan spesifikasi berbasis urutan (SBS), seperti yang ditunjukkan dalam lingkungan pengembangan ASD, dari sebagian perangkat lunak pengawas: pengontrol jalur kaca, bagian pemindai yang mengontrol penanganan slide kaca masuk, keluar, dan di dalam mesin. Seperti yang ditunjukkan pada Gambar 3.8, langkah verifikasi gagal karena model antarmuka (tidak ditampilkan) memerlukan desain untuk mengirimkan peristiwa pemberitahuan OnScanningStopped pada saat ini dalam pelaksanaannya. Perancang secara tidak sengaja menentukan acara OnScanningFinished sebagai gantinya. Verifikasi formal

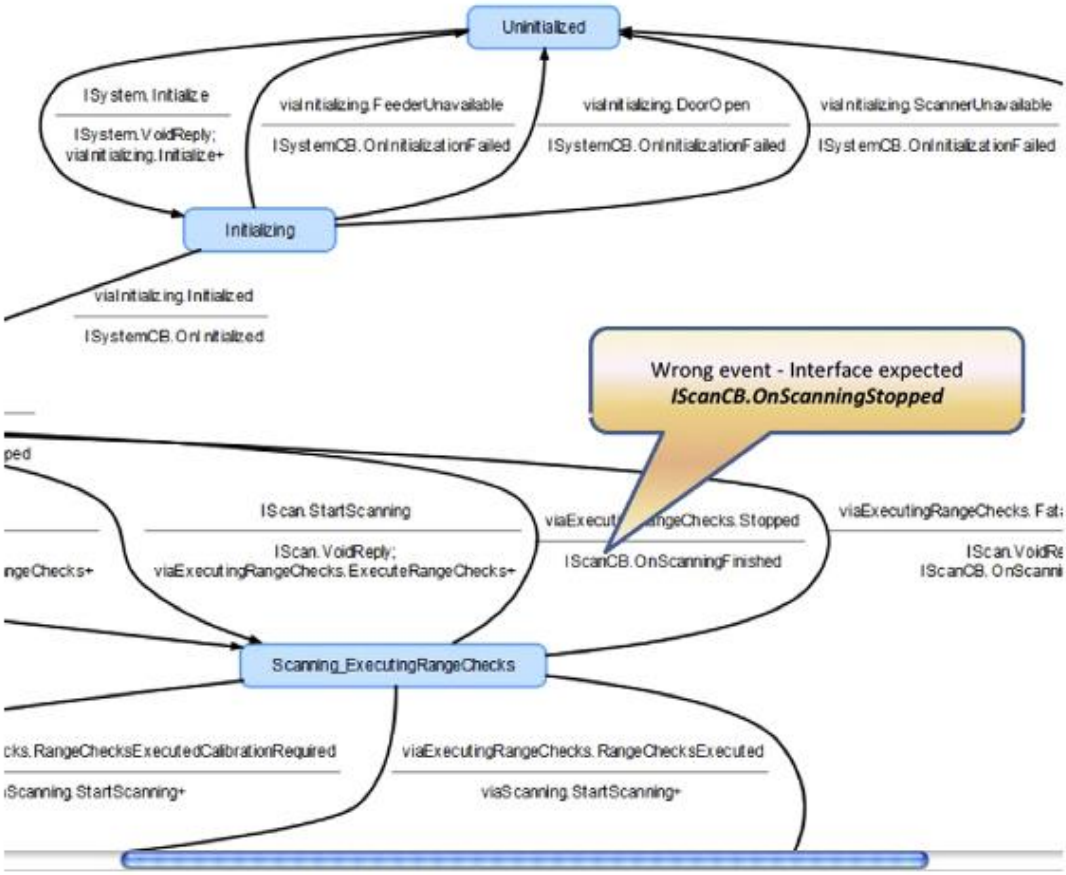
mendeteksi ketidaksesuaian antara perilaku yang diperlukan yang ditentukan dalam model antarmuka dan perilaku yang dirancang yang ditentukan dalam model desain.



	Interface	Event	Guard	Actions
95	**Idle <ISystem.Initialize, viaInitializing.Initialized>**			
97	IScan	StartScanning	rangeCheckStatus==RC_Complete	IScan.VoidReply; viaScanning.StartScanning+
98	IScan	StartScanning	rangeCheckStatus==RC_Interrupted	IScan.VoidReply; viaExecutingRangeChecks.ContinueRangeChecks+
99	IScan	StartScanning	rangeCheckStatus==RC_None	IScan.VoidR viaExecuting
101	IScan	StopScanning		IScan.VoidR IScanCB.Ons
103	ISystem	Terminate		ISystem.VoidReply; viaTerminating.Term
145	**Scanning <ISystem.Initialize, viaInitializing.Initialized, IScan.StartScanning>**			
156	viaScanning	Finished		IScanCB.OnScanni d
157	viaScanning	Error		ISystemCB.OnFat
158	viaScanning	Stopped		IScanCB.OnScan pped
192	**Scanning_ExecutingRangeChecks <ISystem.Initialize, viaInitializing.Initialized, IScan.StartScanning>**			
206	viaExecutingRangeChecks	RangeChecksExecuted		viaScanning.St anning+
207	viaExecutingRangeChecks	FatalError		ISystemCB.O alError
208	viaExecutingRangeChecks	RangeChecksExecutedCalibrationRequired		viaScanning tartScanning+
209	viaExecutingRangeChecks	Stopped		IScanCB.OnScanningFinished
210	viaExecutingRangeChecks	FatalErrorStopRequested		IScan.VoidReply; IScanCB.OnScanningStopped
239	**Terminating <ISystem.Initialize, viaInitializing.Initialized, ISystem.Terminate>**			
249	viaTerminating	Terminated		ISystemCB.OnTerminated
286	**FatalError <ISystem.Initialize, viaInitializing.Initialized, IScan.StartScanning, viaScanning.Error>**			
289	IScan	StopScanning		IScanCB.OnScanningStopped; IScan.VoidReply
291	ISystem	Terminate	stopBeforeTerminate==true	ISystem.VoidReply; viaTerminating.StopAndTerminate+
292	ISystem	Terminate	stopBeforeTerminate==false	ISystem.VoidReply; viaTerminating.Terminate+

Gambar 3.8 Spesifikasi berbasis urutan ASD.

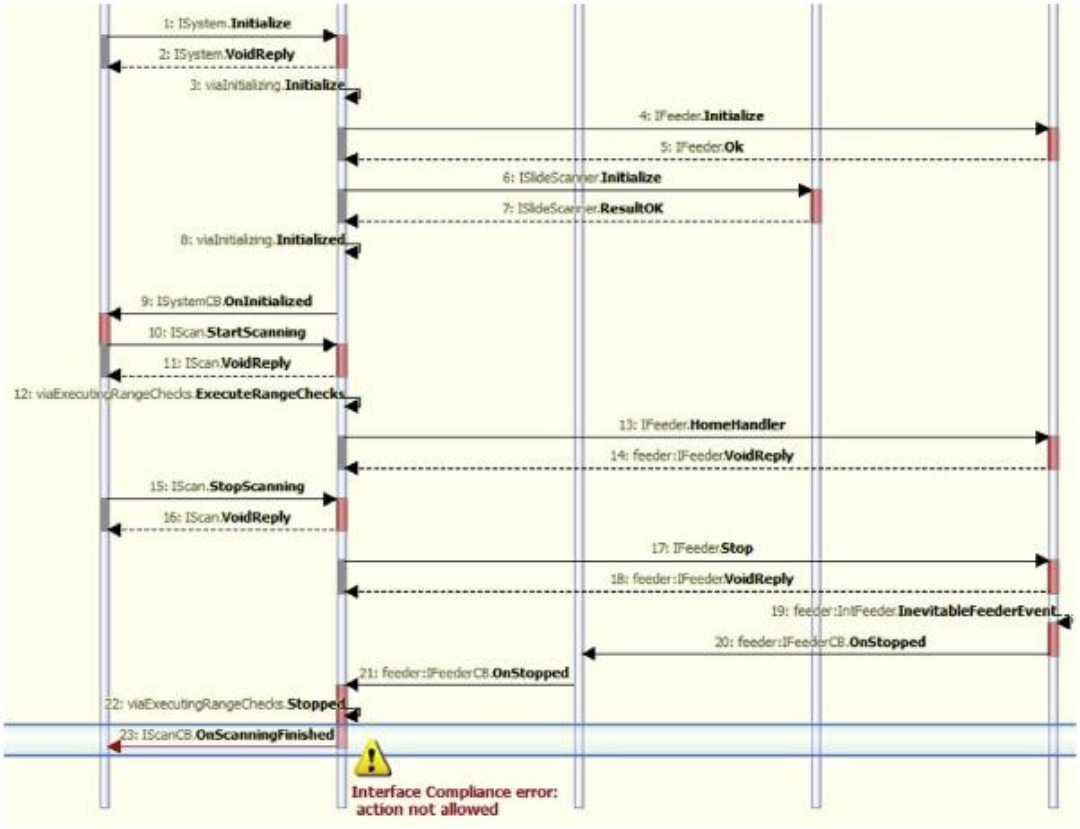
SBS menunjukkan cara pengguna ASD menentukan logika desain dalam bentuk aturan yang memetakan urutan respons terhadap setiap pemicu masukan. Diagram alir keadaan untuk pengontrol jalur kaca ditunjukkan pada Gambar 3.9. Juga ditampilkan acara OnScanningFinished yang salah.



Gambar 3.9 Diagram alir keadaan (parsial) untuk pengontrol jalur kaca.

Diagram alur keadaan ini secara otomatis dihasilkan oleh ASD dari input SBS oleh pengguna. Aliran keadaan adalah tampilan visual yang mudah digunakan untuk memahami transisi keadaan yang disebabkan oleh peristiwa. Busur yang mewakili transisi ilegal ditampilkan dengan jelas di dekat bagian tengah Gambar 3.9.

Ketika analisis formal mendeteksi kesalahan, ASD menampilkan urutan kejadian terpendek yang menyebabkan kegagalan. Diagram urutan yang menjelaskan peristiwa OnScanningFinished ilegal ditunjukkan pada Gambar 3.10.



Gambar 3.10 Diagram urutan menunjukkan bagaimana implementasi desain tidak sesuai dengan spesifikasi antarmuka.

Untuk memahami kemampuan pembuatan kode ASD, mari kita lihat fragmen kode berikut, yang secara otomatis menghasilkan kode C untuk model yang dikoreksi, sesuai dengan metode transisi yang Dihentikan untuk status ExecutingRangeChecks. Peristiwa ini menyebabkan transisi ke kondisi Idle. Jelas dari kode ini bahwa tidak ada transisi negara lain yang sah dari negara ExecutingRangeChecks.

```
static void viaExecutingRangeChecks_Stopped(GlassPathController*
self)
{
ASDDBG_ENTRYTRACE();
switch (self->impl.asd_data.machine){
case GLASSPATHCONTROLLER_MACHINE:
switch (self->impl.asd_data.state){
case GLASSPATHCONTROLLER_SCANNING_EXECUTINGRANGECHECKS_STATE: {
/* Rulecase line number: 209
* The client interrupted the range check by a StopScanning()
request .* /
self->impl.asd_data.state = GLASSPATHCONTROLLER_IDLE_STATE;
IScanCB_OnScanningStopped(self->IScanCB_intf);
self->impl.asd_data.predicate.rangeCheckStatus =
GLASSPATHCONTROLLER_RC_Interrupted_PRED;
}
```

```
break;
default:
asd_illegal(ASD_ILLEGAL_MSG);
break;
}
break:
default:
asd_illegal(ASD_ILLEGAL_MSG);
break;
}
ASDDBG_EXITTRACE();
}
```

3.8.6 Memilih Platform MDD

Contoh Platform MDD Komersial - Platform Desain Perangkat Lunak Analitik

Vendor *Analytical Software Design* (ASD) menggambarkan produk tersebut sebagai platform *Software Design Automation* (SDA) dan bukan platform MDD. Alasannya adalah bahwa ASD disusun dengan verifikasi formal praktis sebagai intinya. Ini adalah kemampuan untuk memverifikasi desain perangkat lunak skala industri secara formal dan membuatnya tersedia bagi semua pengembang perangkat lunak tanpa memerlukan pengetahuan khusus tentang teknik verifikasi formal yang diklaim oleh vendor sebagai fitur pembedanya dan dipertimbangkan dalam kategori produk berbeda dari platform MDD yang tidak memilikinya. kemampuan. Istilah SDA berasal dari platform EDA yang digunakan untuk desain perangkat keras dengan alasan bahwa ASD banyak menggunakan teknik verifikasi formal dengan cara yang analog dengan penggunaan teknik verifikasi formal oleh platform EDA untuk desain logika perangkat keras.

ASD adalah teknologi berbasis komponen untuk membangun sistem skala industri yang benar dan lengkap dari komponen yang diverifikasi secara formal. ASD menggunakan bentuk tabel non-grafis dari bahasa spesifikasi berbasis urutan di mana respons sistem ditentukan untuk setiap kemungkinan urutan peristiwa masukan.[53] ASD:Suite menyediakan hal berikut:

1. Editor Model yang digunakan untuk membuat model komponen.
2. Alat Perbandingan Model untuk membandingkan versi model yang berbeda dan menunjukkan perbedaannya.
3. Verifikasi formal desain yang sepenuhnya otomatis terhadap spesifikasi.
4. Pembuatan kode di MISRA C, C++, dan Java.
5. Dukungan untuk berbagai macam platform runtime, mulai dari silikon “telanjang” hingga sistem multi-prosesor dan multi-core yang kompleks dengan sistem operasi real-time atau tujuan umum.

ASD:Suite sendiri memiliki arsitektur klien/server; alat Editor Model dan Perbandingan Model dipasang di desktop pengguna sementara pemverifikasi model dan pembuat kode disediakan sebagai layanan yang dihosting. Verifikasi formal otomatis sangat intensif sumber daya, dan layanan yang dihosting memungkinkan pengguna mengakses kemampuan ini tanpa

[53] Prowell SJ, Poore JH. Specification, Foundations of Sequence-Based Software. IEEE Transactions on Software Engineering; May 2003:417e25.

berinvestasi pada sistem perangkat keras yang kuat. Keuntungan lain dari layanan yang dihosting adalah keamanan pengembangan: karena pembuat kode diamankan di pusat data Verum, pengembang tidak perlu khawatir tentang perlindungan terhadap jenis serangan kuda Trojan internal yang dijelaskan sebelumnya dalam bab ini. ASD mengakui perbedaan mendasar antara antarmuka dan implementasi dan menggunakan dua jenis model yang berbeda: model antarmuka dan model desain.

Model antarmuka ASD tidak hanya menangkap sintaks antarmuka komponen, seperti operasi dan metode, tipe data parameter, dan sebagainya, namun juga perilaku yang terlihat secara eksternal dalam bentuk spesifikasi berbasis urutan. Perilaku yang terlihat secara eksternal inilah yang dapat dianggap benar oleh komponen yang menggunakan antarmuka untuk setiap implementasi yang sesuai, namun tetap bebas dari semua perilaku internal dan detail implementasi. Model antarmuka ASD dapat dibuat sepenuhnya independen dari bahasa target.

Sebaliknya, model desain ASD adalah spesifikasi perilaku internal yang menentukan implementasinya. Semua kode sumber perilaku dihasilkan dari model desain; model antarmuka digunakan untuk menghasilkan kode sumber yang mewakili antarmuka eksternal komponen. Ketika bahasa targetnya adalah C++, misalnya, model antarmuka digunakan untuk menghasilkan deklarasi kelas abstrak, dan model desain digunakan untuk menghasilkan implementasi kelas. Model desain ASD juga tidak bergantung pada bahasa pemrograman target. Semua bahasa yang didukung dapat dihasilkan dari model ASD apa pun.

Sebelumnya, kita telah membahas kesulitan menerapkan verifikasi formal pada desain perangkat lunak. Secara khusus, verifikasi formal memerlukan keterampilan khusus yang kurang dimiliki sebagian besar organisasi pembangunan, dan desain sistem modern seringkali terlalu rumit untuk penerapan teknik formal.

ASD mengatasi masalah ini dengan dua cara: pertama, dengan memasukkan perilaku yang terlihat secara eksternal sebagai bagian dari spesifikasi antarmuka komponen, dan kedua, dengan menggunakan teknik yang disebut verifikasi komposisi di mana sistem diverifikasi komponen demi komponen sedemikian rupa untuk memastikan bahwa semua properti yang diverifikasi pada tingkat komponen berlaku ketika komponen membentuk sistem yang lengkap.

Model desain komponen ASD merujuk pada sejumlah model antarmuka. Salah satunya, model antarmuka yang diimplementasikan, adalah spesifikasi dari semua perilaku fungsional yang terlihat secara eksternal yang harus diimplementasikan oleh desain. Yang lainnya, model antarmuka bekas, adalah model antarmuka dari komponen lain yang digunakan oleh desain. Bersama-sama, model antarmuka yang direferensikan ini membentuk spesifikasi fungsional yang akan digunakan untuk memverifikasi kepatuhan desain secara formal.

Masalah kegunaan dan ekonomi dalam memperoleh kumpulan properti yang diperlukan untuk verifikasi desain telah diselesaikan karena pengguna ASD tidak perlu menentukannya secara manual; properti ini dihasilkan secara otomatis oleh platform ASD dari kumpulan model antarmuka yang direferensikan. Pengguna ASD tidak memerlukan pengetahuan khusus untuk dapat memverifikasi desain perangkat lunak secara formal.

Masalah skalabilitas dalam menerapkan verifikasi formal untuk menyelesaikan sistem yang terdiri dari desain model baru, kode lama, dan komponen perangkat lunak siap pakai juga ditangani. Desain baru yang dimodelkan dengan ASD memiliki model antarmuka ASD dan model desain ASD, seperti yang disebutkan sebelumnya. Kode lama atau komponen siap pakai tidak akan memiliki model desain ASD; sebaliknya, pengguna harus menganalisis spesifikasi konvensional komponen dan kemudian membuat model antarmuka ASD yang mewakili komponen tersebut.

Verifikasi ASD sepenuhnya otomatis. Sebuah komponen diverifikasi dengan mengirimkan model desain ASD-nya untuk verifikasi bersama dengan kumpulan model antarmuka ASD yang direferensikan. Platform ASD menerjemahkan model ASD ke dalam model matematika dalam aljabar proses yang disebut *Communicating Sequential Processes* (CSP).[54] Model CSP yang dihasilkan diperiksa secara mendalam oleh mesin pemeriksa model keadaan terbatas yang berjalan di server farm vendor.

ASD saat ini mendukung dua model eksekusi: yaitu model eksekusi standar (SEM) dan model eksekusi single-threaded (STM). Dalam kedua model eksekusi, setiap komponen ASD berperilaku sebagai Hoare Monitor dan secara otomatis menyediakan saling pengecualian dan sinkronisasi yang diperlukan.[55] Dalam model eksekusi standar, semua komponen dijalankan secara bersamaan dan asinkron. Namun, ASD menyadari bahwa sebagian besar interaksi antar komponen dalam desain praktis bersifat sinkron, yang pada akhirnya diimplementasikan pada lingkungan target melalui panggilan prosedur sinkron. Daripada mengharuskan pengguna untuk memodelkan sinkronisasi secara eksplisit untuk kasus paling umum ini, ASD melakukannya secara otomatis. Pengguna hanya perlu memodelkan secara eksplisit sinkronisasi yang diperlukan untuk mengoordinasikan interaksi asinkron antara komponen yang dijalankan secara bersamaan.

Sebaliknya, model eksekusi single-thread mengasumsikan hanya satu thread kontrol melalui seluruh tumpukan komponen single-thread. Selain itu, hal ini harus dijamin oleh lingkungan, dan kode yang dihasilkan tidak menyertakan mekanisme untuk memastikan saling pengecualian pada waktu proses. Model ini dirancang untuk aplikasi sederhana dan tertanam dalam yang dijalankan pada platform target dengan sumber daya yang sangat terbatas. Model ini juga banyak digunakan dalam domain telekomunikasi ketika ribuan atau jutaan komponen diperlukan untuk mendukung panggilan aktif melalui node switching.

Komponen dari kedua model yang dapat dieksekusi dapat digabungkan dalam satu subsistem; ini sangat berguna ketika berinteraksi dengan kerangka GUI. Biasanya, kerangka kerja GUI modern merupakan sebuah rangkaian tunggal dan digerakkan oleh apa yang disebut dengan pompa pesan: kerangka tersebut mengharuskan tindakan yang dipanggil secara sinkron oleh kerangka kerja untuk menyelesaikan pemrosesannya pada rangkaian panggilan. Mampu menggabungkan komponen dengan model eksekusi yang berbeda memungkinkan, misalnya, satu set komponen berulir untuk berinteraksi langsung dengan kerangka GUI sambil memanggil komponen model standar untuk melaksanakan tindakan yang diminta melalui GUI baik secara sinkron atau asinkron, seperti yang diminta oleh desain.

---

[54] Roscoe AW. Understanding Concurrent Systems. London: Springer Verlag London Ltd.; 2010.

[55] Hoare CA. Monitors: An Operating System Structuring Concept. Communications of the ACM; October 1974::547e8.

IBM Rasional Rhapsody

Rhapsody adalah platform pemodelan visual berbasis UML/SysML yang ditujukan untuk pengembang perangkat lunak tertanam. Ini tersedia dalam versi Eclipse dan Microsoft Windows. Rhapsody ditujukan untuk siklus hidup pengembangan yang lengkap, mulai dari persyaratan hingga penerapan. Rhapsody tersedia dalam beberapa edisi, masing-masing ditujukan untuk komunitas pengguna tertentu. Tergantung pada edisi yang dipilih, fitur Rhapsody meliputi

1. Persyaratan terintegrasi dan lingkungan pemodelan menggunakan diagram SysML atau UML standar industri.
2. Ketertelusuran dan analisis siklus hidup penuh mulai dari persyaratan hingga desain. Persyaratan dapat diimpor dari sumber eksternal seperti DOORS, IBM Rational Requisite Pro, Microsoft Word, dan Excel. Persyaratan juga dapat dikelola langsung dari dalam Rhapsody dan sepenuhnya dapat ditelusuri ke kasus penggunaan dan komponen perangkat lunak. Ini menyediakan
   - Analisis pemeriksaan model statis untuk meningkatkan konsistensi desain.
   - Dukungan untuk perbedaan grafis dan kemampuan penggabungan.
   - Visualisasi arsitektur dan desain dengan UML standar industri.
   - Pembuatan bingkai kode C, C++, atau Java.
   - Pembuatan kode perilaku penuh untuk C, C++, Java, dan Ada.

Dokumentasi dapat dihasilkan dari model, sehingga mengurangi upaya kepatuhan yang diperlukan untuk perangkat lunak penting keselamatan. Model Rhapsody dapat dieksekusi, dan verifikasi dilakukan melalui simulasi dan pengujian. Rhapsody tidak memberikan verifikasi formal atas kebenaran desain.

Grafik Mentor BridgePoint

Mentor Graphics BridgePoint adalah platform pemodelan visual berbasis UML lainnya yang ditujukan untuk pengembang perangkat lunak tertanam dengan penekanan khusus pada sistem waktu nyata. BridgePoint didasarkan pada metode Shlaer-Mellor dan menggunakan UML yang dapat dieksekusi (xtUML) sebagai bahasa pemodelannya. BridgePoint ditujukan untuk semua fase pengembangan, mulai dari arsitektur hingga penerapan.

Selain elemen inti bahasa xtUML yang disebutkan sebelumnya (bagan domain, diagram kelas, diagram mesin status, dan spesifikasi tindakan), BridgePoint juga mendukung diagram kasus penggunaan, diagram urutan, diagram komunikasi, dan diagram aktivitas. Fiturnya meliputi

1. Analisis pemeriksaan model statis, yang membantu konsistensi desain selama konstruksi model. Saat elemen model berubah, spesifikasi tindakan secara otomatis diperbarui sesuai kebutuhan untuk menjaga konsistensi model.
2. Kemampuan memvisualisasikan arsitektur dan desain dengan UML standar industri.
3. Pembuatan kode perilaku penuh untuk C dan C++.
4. Kompiler model untuk menerjemahkan xtUML ke C atau C++.

Model BridgePoint adalah model platform-independen (PIM) yang dapat langsung dieksekusi di xtUML tanpa memerlukan pembuatan kode atau terjemahan dari xtUML. Eksekusi langsung mendukung verifikasi selama tahap desain.

Rangkaian SCADE

SCADE (*Safety Critical Application Development Environment*) adalah bahasa dan seperangkat alat yang secara khusus ditujukan untuk aplikasi yang kritis terhadap keselamatan. SCADE dihasilkan dari upaya bersama oleh Airbus, Schneider Electric, dan Verilog (mantan perusahaan rintisan Perancis, bukan bahasa deskripsi perangkat keras dengan nama yang sama). SCADE didasarkan pada teori bahasa sinkron untuk aplikasi real-time deterministik dan khususnya pada bahasa Lustre dan Esterel.[56][57]

SCADE mengimplementasikan model eksekusi berbasis siklus yang sinkron, model pengambilan sampel/penggerak umum dari teknik kontrol, dengan mengeksekusi loop kontinu bergantian antara membaca sensor input dan mengeksekusi tindakan aplikasi. Pada setiap siklus, sensor masukan dibaca, dan keluaran siklus dihitung dan diumpankan kembali ke lingkungan. SCADE digunakan untuk mengembangkan logika aplikasi dalam loop kontrol berkelanjutan ini.

SCADE Suite dari Esterel Technologies adalah anggota keluarga produk yang menangani semua fase pengembangan perangkat lunak untuk perangkat lunak kontrol. Fitur-fiturnya antara lain

1. Pemodelan visual aliran data dan aliran kontrol menggunakan kombinasi diagram blok aliran data dan diagram mesin keadaan.
2. Pemodelan visual antarmuka GUI.
3. Generator kode C yang menghasilkan kode tapak kecil yang memenuhi sejumlah standar keselamatan, termasuk DO-178B, IEC 61508, EN 50128, dan IEC 60880.
4. Pembuat kode Ada.
5. Adaptor yang dapat disesuaikan untuk memungkinkan kode yang dihasilkan berjalan di lingkungan target apa pun, mulai dari silikon telanjang hingga yang memiliki RTOS onboard. INTEGRITY dari Green Hills Software, VxWorks dari Wind River, dan PikeOS dari Sysgo termasuk di antara yang didukung langsung oleh suite ini.
6. Verifikasi berdasarkan simulasi.
7. Verifikasi formal terhadap perilaku fungsional dan properti waktu.

Anggota lain dari keluarga SCADE termasuk Sistem SCADE, untuk desain arsitektur sistem kritis berbasis model, dan SCADE Display, yang menyediakan lingkungan pemodelan visual untuk mengembangkan tampilan penting dan antarmuka grafis manusia/mesin untuk sistem yang kritis terhadap keselamatan dan kritis terhadap misi.

Pemverifikasi waktu SCADE secara statis memeriksa kode yang dikompilasi dan ditautkan dari suatu subsistem atau sistem yang lengkap. Pengguna menentukan informasi tentang CPU target seperti kecepatan jam, perilaku cache, dan struktur saluran. Selain itu, alat ini mengharuskan batas atas jumlah iterasi setiap loop diketahui dan ditentukan. Informasi loop ini dapat diekstraksi secara otomatis dari kode yang dihasilkan SCADE Suite tetapi harus ditentukan secara manual untuk kode lama atau modul kode pihak ketiga.

Pendekatan yang diambil oleh SCADE Suite dimungkinkan karena perilaku sistem yang dihasilkannya bersifat sekuensial dan deterministik. Pengembang harus menyelidiki apakah

[56] Halbwachs N, Caspi P, Raymond P, Pilaud D. The Synchronous Dataflow Programming Language Lustre. Proceedings of the IEEE September 1991;79(9):1305e20.

[57] Berry G, Gonthier G. The Esterel Synchronous Programming Language, Design, Semantics and Implementation. Science of Computer Programming; February 1992:87e152.

pendekatan ini cukup berskala dan layak untuk aplikasi tertentu. Karena fokusnya pada keselamatan dan aplikasi sistem tertanam yang sangat penting, vendor telah melakukan prakualifikasi pada platform dan, khususnya, generator kodenya untuk mematuhi sejumlah standar keselamatan, termasuk IEC 61508 dan standar turunannya ISO 26262, EN/ISO 13849, dan IEC 60324.

Pilihan Platform MDD Bersifat Strategis

Mengadopsi pendekatan MDD dalam pengembangan perangkat lunak adalah keputusan yang mempunyai banyak konsekuensi, namun tidak semuanya terlihat jelas. Beberapa pertimbangan terpenting dibahas pada bagian berikut.

Di dunia EDA, pemilihan platform EDA dalam praktiknya ternyata merupakan keputusan strategis. Bukan hal yang aneh bagi pengguna untuk berpindah vendor dan platform karena biaya transisi sangat tinggi. Biaya tersebut mencakup investasi besar dalam melatih kembali pengguna dan memindahkan kekayaan intelektual yang ada ke dalam lingkungan pemodelan baru.

Meskipun beberapa platform MDD mengklaim sebaliknya, kenyataan yang sama juga berlaku untuk MDD. Seiring berjalannya waktu, sebagian besar kekayaan intelektual perangkat lunak organisasi akan berbentuk model khusus untuk satu platform MDD dan juga satu vendor.

> *Meskipun model portabel yang dapat diimpor secara mulus dengan pelestarian semantik penuh antar platform vendor merupakan hal yang diinginkan, kemampuan ini belum ada dalam praktiknya; setiap platform MDD memiliki banyak asumsi yang tidak disebutkan mengenai arti sebenarnya dan interpretasi modelnya, dan asumsi ini dimasukkan ke dalam generator kode, simulator, generator kasus uji, verifikator formal, dan platform runtime pendukung.*

Bahkan untuk platform MDD yang menerapkan standar UML dan/atau SysML, masalahnya tetap ada. Semua bahasa berbasis UML dirancang untuk dapat diperluas, dengan banyak titik variasi semantik. Setiap vendor platform MDD kemungkinan besar telah membuat ekstensi semantik khusus vendor.

Keterampilan

Insinyur perangkat lunak akan memerlukan pelatihan dalam penggunaan platform MDD, dan jumlah pelatihan bervariasi antar platform. Meskipun merupakan pertimbangan penting, pelatihan bukanlah hal yang paling penting ketika mempertimbangkan keterampilan yang dibutuhkan.

Seperti disebutkan sebelumnya, desain berbasis model pada dasarnya adalah tentang bekerja pada tingkat abstraksi yang lebih tinggi dibandingkan biasanya atau mungkin dilakukan dengan praktik pengembangan perangkat lunak konvensional. Akibatnya, pengembangan perangkat lunak berbasis model menghargai arsitektur perangkat lunak dan keterampilan desain tingkat tinggi di atas keterampilan pemrograman. Banyak programmer yang berpengalaman dan berketerampilan tinggi merasa kesulitan untuk mengembangkan arsitektur dan desain perangkat lunak pada tingkat abstraksi yang diperlukan untuk pemodelan yang sukses. Semakin besar dan kompleks sistem yang dirancang, semakin serius permasalahannya.

Oleh karena itu penting untuk menilai tingkat keterampilan ini dalam organisasi yang ada sebagai bagian dari proses pengambilan keputusan. Menjembatani kesenjangan antara

keahlian yang dibutuhkan dan keterampilan yang tersedia memerlukan pelatihan yang diikuti dengan pendampingan dan pembinaan. MDD terkadang diadopsi oleh organisasi sebagai upaya untuk mengatasi kekurangan dalam keterampilan desain, namun dorongan ini jarang menghasilkan kesuksesan. Alih-alih mengatasi defisit keterampilan, hal ini hanya berfungsi untuk menyorotinya.

Dampak terhadap Alur Kerja yang Ada

Pengalaman menunjukkan MDD kompatibel dengan metode pengembangan yang bertahap dan tangkas.

> *Dibandingkan dengan metode konvensional, MDD memungkinkan verifikasi dimulai lebih awal, segera setelah model selesai; verifikasi tidak bergantung pada kode tulisan tangan atau kasus uji. Namun, untuk memanfaatkan peluang ini, organisasi harus menetapkan persyaratan jauh lebih awal dalam siklus hidup pengembangan dan dengan tingkat presisi yang lebih tinggi dibandingkan dengan pengembangan perangkat lunak konvensional.*

Beberapa platform MDD, seperti Rhapsody, BridgePoint, dan ASD, ditujukan untuk pengembangan sistem perangkat lunak yang lengkap dan kompleks serta menerapkan pendekatan “end-to-end” untuk semua fase siklus hidup pengembangan perangkat lunak, mulai dari persyaratan hingga pemeliharaan setelahnya. penyebaran. Arsitektur mapan yang mempartisi sistem dengan rapi menjadi komponen-komponen yang koheren merupakan prasyarat untuk membuat model fungsional yang sukses. Banyak platform MDD mengharuskan proses ini diikuti, dan, terutama bagi organisasi yang tingkat kematangannya lebih rendah (misalnya CMMI level dua atau lebih rendah), dampak dari proses ini bisa sangat parah.

Ketika mempertimbangkan MDD sebagai cara kerja dan memilih platform MDD, pengembang harus memahami apa yang dibutuhkan oleh platform MDD yang dipertimbangkan dari suatu organisasi dan bagaimana penggunaannya akan mengubah cara organisasi beroperasi saat ini.

Berinteraksi dengan Kode Lama dan Perangkat Lunak Siap Pakai

Kebanyakan sistem yang kompleks berisi kombinasi komponen perangkat lunak yang baru dikembangkan, digunakan kembali dari kode lama, dan dipasok oleh pihak ketiga. Kode lama dan komponen pihak ketiga sering kali tidak dikembangkan menggunakan platform MDD, dan tidak ada model komponen tersebut. Selain itu, seperti dalam studi kasus pemindai patologi, akan terdapat lapisan perangkat lunak tulisan tangan yang mengimplementasikan antarmuka abstraksi perangkat keras, driver perangkat, dan sistem operasi yang mendasarinya.

Ketika kode yang dirancang dan dihasilkan MDD dicampur dengan kode tulisan tangan, ada dua masalah utama yang perlu dipertimbangkan: pertama, bagaimana model MDD dapat diverifikasi ketika eksekusi runtime yang benar bergantung pada perilaku dalam kode tulisan tangan yang tidak dimodelkan? Kedua, bagaimana kode yang dihasilkan model berinteraksi dengan kode tulisan tangan saat runtime?

Masalah pertama adalah masalah besar; sebagian besar manfaat MDD berasal dari kemampuan memverifikasi model desain tanpa memerlukan platform target (sehingga mengurangi 40% uang pengembangan yang dihabiskan untuk pengujian dan integrasi). Agar perekonomian ini dapat terwujud, perilaku fungsional yang terlihat secara eksternal dari kode

yang tidak dimodelkan harus ditangkap dan dispesifikasikan dalam beberapa cara, baik verifikasinya dilakukan secara informal melalui simulasi atau melalui verifikasi formal.

Platform MDD mengatasi masalah ini secara berbeda. Beberapa platform mengizinkan komponen yang tidak dimodelkan untuk dipanggil langsung dari model selama verifikasi. Pendekatan umum lainnya adalah menyediakan bahasa di mana perilaku fungsional kode yang tidak dimodelkan dapat ditentukan pada abstraksi tingkat tinggi. Misalnya, platform ASD mengatasi masalah ini dengan mewajibkan model antarmuka ASD yang mewakili perilaku fungsional kode yang tidak dimodelkan.

Terlepas dari pendekatan yang diambil, masalah memastikan kesesuaian model ini dengan kode tulisan tangan tetap ada pada pengguna. Verifikasi berdasarkan model hanya valid sejauh kode lama atau komponen pihak ketiga berperilaku sesuai dengan modelnya. Masalah kedua menyangkut konvensi yang diasumsikan oleh platform MDD ketika menghasilkan kode dan asumsi yang dibuat platform tentang model memori dan proses eksekusi runtime serta struktur thread. Dalam banyak kasus, banyak upaya yang dapat dilakukan untuk membungkus kode tulisan tangan untuk mentransfer kontrol dan data antara kode yang dihasilkan dan kode tulisan tangan.

Pembuatan Kode

Pembuatan kode belum mendapat penerimaan luas di dunia perangkat lunak tertanam. Namun, seperti disebutkan sebelumnya, sebagian besar nilai MDD akan hilang kecuali kode dihasilkan langsung dari model. Vendor platform MDD bertanggung jawab untuk memastikan bahwa kode yang dihasilkan dijalankan persis seperti yang dimodelkan. Jika kode ditulis tangan menggunakan model terverifikasi sebagai spesifikasi, atau kode yang dihasilkan dimodifikasi, maka semua taruhan dibatalkan. Setelah memverifikasi model, pengembang harus memverifikasi ulang kode tulisan tangan atau yang dimodifikasi dengan pengujian konvensional, sehingga menghilangkan keunggulan MDD.

Ketika sebuah organisasi memilih platform MDD, karakteristik efisiensi, portabilitas, dan kepatuhan dari kode yang dihasilkan sangatlah penting, terutama dalam domain tertanam. Pengembang dan pengelola harus memeriksa kode sampel yang dihasilkan dari model realistis agar memenuhi setidaknya hal berikut:

(1) **Jejak memori**: Apakah kode yang dihasilkan cukup kecil untuk dimasukkan ke dalam sistem target? Apakah jejak memori terdistribusi dengan benar antara memori ROM dan RAM (jika ada)? Dalam kasus sistem yang sangat terbatas sumber dayanya, permasalahan ini dapat menjadi penghalang. Model alokasi memori: Beberapa lingkungan target, seperti lingkungan yang sesuai dengan standar MISRA yang dibahas sebelumnya dalam bab ini, mengharuskan semua sumber daya dialokasikan sebelum eksekusi dimulai dan melarang alokasi memori dinamis; akibatnya, rekursi dan padanan bahasa tertentu malloc() dan free() dilarang. Apakah platform MDD menyediakan opsi pembuatan kode untuk mematuhi subset bahasa ini (jika berlaku)?

(2) **Penggabungan kode tulisan tangan/kode lama**: Beberapa pembuat kode menggunakan konvensi panggilan kepemilikan dalam kode yang dihasilkan. Sejauh mana konvensi ini kompatibel dengan kode lama sistem? Dan jika tidak kompatibel, apa solusinya agar kode yang dihasilkan kompatibel? Jika pembungkus diperlukan, apakah ini sepenuhnya ditulis tangan (menimbulkan upaya dan pemeliharaan ekstra

serta potensi risiko injeksi kesalahan), atau apakah pembuat kode memberikan bantuan dalam proses ini?

(3) **Model eksekusi threading dan runtime**: Sebagian besar platform MDD memiliki model eksekusi dan threading tertentu yang tidak dapat dimodifikasi oleh pengembang. Misalnya, beberapa platform memiliki model penyampaian pesan yang ketat untuk menggambarkan interaksi antar komponen dan mengasumsikan konkurensi lengkap, dengan satu thread dialokasikan ke setiap komponen. Platform ini menyediakan sarana yang digunakan untuk memetakan thread yang dihasilkan ke thread sistem operasi aktual saat runtime. Ada beberapa domain di mana model threading ini tidak berfungsi, misalnya, dalam domain telekomunikasi di mana terdapat sejumlah besar komponen instance dan mungkin sumber daya sistem tidak cukup untuk mendukung jumlah thread ini. Perancang perlu memahami model eksekusi platform kandidat MDD dan menentukan apakah model tersebut memadai untuk lingkungan target.

Kecepatan Eksekusi Runtime

Kecuali jika target tertanam secara mendalam atau kinerjanya dibatasi sehingga pengembang harus menghitung siklus secara teratur, kecepatan eksekusi adalah salah satu masalah potensial yang tidak dimiliki oleh sebagian besar kode yang dihasilkan. Pengalaman menunjukkan bahwa di semua lingkungan target, kecuali lingkungan target yang kinerjanya paling terbatas, kinerja kode yang dihasilkan secara otomatis sudah memadai.

Untuk aplikasi yang kinerjanya penting, beberapa platform menawarkan fasilitas untuk memverifikasi properti pengaturan waktu. Misalnya, SCADE Suite memiliki fitur untuk menghitung dan memverifikasi waktu eksekusi kasus terburuk untuk aplikasi berurutan dengan memeriksa secara statis executable yang dikompilasi dan dihubungkan dari suatu subsistem atau sistem yang lengkap.

Persyaratan Sistem Runtime

Aplikasi yang berbeda memiliki persyaratan platform target yang berbeda-beda yang harus didukung oleh platform MDD yang dipilih. Misalnya:

1. Jika sistem target adalah silikon “telanjang” yang tertanam dalam (tanpa sistem operasi) dengan sumber daya terbatas, apakah kode yang dihasilkan akan dieksekusi di lingkungan target “di luar kotak”, atau akankah penjadwal tugas sederhana dan rutinitas layanan interupsi perlu dikembangkan dengan tangan?
2. Jika sistem target memiliki sistem operasi, apakah kode yang dihasilkan mendukungnya? Jika tidak, bagaimana kode yang dihasilkan dapat di-porting ke lingkungan target? Apakah ada lapisan abstraksi sistem operasi yang perlu di-porting, atau haruskah kode yang dihasilkan diubah? Jika ya, apakah ada kit pengembangan yang disediakan oleh vendor platform untuk ini dengan spesifikasi yang jelas dan uji kepatuhan?
3. Jika aplikasi tersebut merupakan aplikasi hard real-time (yaitu, aplikasi yang batasan waktunya merupakan properti kebenaran), apakah kode yang dihasilkan dan model eksekusi waktu proses cukup deterministik sehubungan dengan perilaku pengaturan waktu? Apakah sistem mengizinkan interupsi dilayani dengan cukup cepat?

3.8.7 Menggunakan MDD dalam Sistem Kritis Keselamatan dan Keamanan

Penggunaan platform MDD di bidang keselamatan, keamanan, dan misi penting seperti ruang angkasa, aplikasi militer, transportasi kereta api, dan otomotif secara bertahap meningkat. Standar yang mengatur perangkat lunak di domain tersebut semakin menyadari bahwa pengujian unit dapat dikurangi jika platform MDD digunakan yang menyediakan fasilitas verifikasi yang memadai pada model itu sendiri.

Persyaratan rinci yang mengatur penggunaan alat seperti platform MDD berbeda secara rinci dari satu standar ke standar berikutnya. Secara umum, standar-standar tersebut mengikuti pendekatan umum yang sama:

- Standar ini membedakan antara perangkat lunak tertanam operasional yang dikirimkan sebagai bagian dari produk dan alat yang digunakan untuk membuat perangkat lunak tersebut. Sertifikasi biasanya berlaku untuk perangkat lunak operasional, bukan alatnya. Ketika alat tercakup dalam sertifikasi, persyaratannya sangat berbeda dan, secara umum, tidak terlalu ketat.
- Berbagai standar untuk perangkat lunak yang penting bagi keselamatan dan keamanan, misalnya IEC 61508, EN 50128, ISO 26262, dan ISO 15408 (Kriteria Umum) sering kali mengharuskan pengguna alat untuk melakukan penilaian terhadap alat tersebut untuk mengkategorikannya berdasarkan apakah alat tersebut layak atau tidak. alat itu sendiri dapat menimbulkan kesalahan ke dalam operasional perangkat lunak yang tertanam dan melakukan penilaian terhadap alat tersebut terhadap kriteria yang berlaku dalam standar agar alat tersebut memenuhi syarat untuk penggunaan yang aman dan terjamin.
- Semua pengguna alat pengembangan perangkat lunak harus memenuhi syarat untuk menetapkan bahwa alat tersebut mematuhi standar yang berlaku. Tanggung jawab untuk hal ini berada pada pengguna alat dan bukan pada vendor alat. Minimal, pengguna diharuskan untuk memastikan bahwa analisis operasi berbahaya dilakukan dan Panduan Penggunaan Aman ditulis, yang menetapkan, misalnya, pemeriksaan yang harus dilakukan pengguna pada keluaran yang dihasilkan (misalnya, kode yang dihasilkan) dan fitur alat untuk dihindari karena dianggap berbahaya. Selain itu, kesalahan yang diketahui pada versi platform yang digunakan harus diidentifikasi bersama dengan praktik untuk menghindarinya.
- Pengguna harus menetapkan bahwa praktik dan organisasi pengembangan perangkat lunak vendor platform MDD mematuhi standar yang berlaku. Misalnya, EN 50182 mengharuskan produsen platform MDD memiliki orang yang ditunjuk di luar tim pengembangan yang bertanggung jawab untuk memastikan setiap rilis memenuhi persyaratan dan tetap memenuhi syarat. Orang ini harus melapor kepada manajemen paling senior perusahaan dan harus mempunyai wewenang untuk menghentikan pelepasan. Di beberapa yurisdiksi Uni Eropa, orang ini mempunyai tanggung jawab hukum.

Ketika mempertimbangkan platform MDD untuk digunakan dalam domain keselamatan, keamanan, atau misi penting, organisasi harus mempertimbangkan setidaknya hal-hal berikut:

1. Apakah platform MDD sudah digunakan dalam suatu domain yang diatur oleh standar dan peraturan yang sama atau serupa dengan domain organisasi dan/atau proyek?
2. Apakah vendor platform sudah melakukan penilaian kualifikasi yang diawasi oleh organisasi sertifikasi eksternal yang diakui? Melakukan penilaian yang diperlukan agar platform MDD memenuhi syarat untuk penggunaan yang aman adalah proses yang memakan waktu dan mahal; prakualifikasi oleh vendor MDD secara signifikan mengurangi biaya kualifikasi bagi pengguna. Prakualifikasi juga memberikan bukti bahwa penilaian kualifikasi yang mahal dan memakan waktu kemungkinan besar akan berhasil dan menunjukkan komitmen vendor terhadap, dan pengetahuan tentang, domain penting keselamatan atau keamanan organisasi pengembang.
3. Bentuk verifikasi apa yang ditawarkan platform MDD? Khususnya di bidang-bidang yang kritis terhadap keselamatan dan keamanan, verifikasi formal sangatlah berharga.
4. Sejauh mana verifikasi platform MDD dapat digunakan untuk menghilangkan atau mengurangi jumlah pengujian unit konvensional? Dapatkah platform MDD juga menyederhanakan dan mengurangi pengujian integrasi dan menghasilkan penghematan sehubungan dengan validasi end-to-end?
5. Apakah kode yang dihasilkan sudah sesuai dengan standar dan ketentuan yang berlaku? Misalnya, dalam domain otomotif, kode yang dihasilkan harus sesuai dengan standar MISRA C.
6. Apakah platform membantu dalam menghasilkan dokumentasi perangkat lunak tertanam yang disyaratkan oleh standar yang berlaku?

Karena potensi efisiensi biaya dan waktu pengembangan serta kemampuan untuk mengurangi terjadinya kelemahan desain perangkat lunak, desain berbasis model adalah teknik pengembangan perangkat lunak penting yang patut dipertimbangkan untuk sistem tertanam yang memiliki persyaratan keselamatan, keamanan, dan/atau keandalan yang ketat. Memutuskan platform MDD yang paling tepat untuk komponen tertentu dan untuk keseluruhan sistem mengharuskan para insinyur dan manajer memahami kelebihan dan kekurangan teknis inti dari solusi yang tersedia. Konsultasikan dengan vendor perangkat lunak tertanam Anda untuk saran dan evaluasi lebih lanjut.

BAB 4
KRIPTOGRAFI TERTANAM

4.1 PENDAHULUAN

Bab ini memperkenalkan dan menjelaskan konsep dasar kriptografi digital modern. Banyak buku teks menjelaskan sejarah kriptografi, dimulai pada zaman Romawi dengan penemuan dan penggunaan sandi Caesar, yang terdiri dari substitusi alfabet tunggal berdasarkan sejumlah rahasia pergeseran siklik dalam alfabet, hingga sandi Enigma poli-abjad yang digunakan oleh orang Jerman. selama Perang Dunia II. Semua sandi ini menggunakan rotor dan roda mekanis untuk mengganti karakter secara acak. Sebagian besar sandi, seperti sandi Caesar, memiliki kelemahan mendasar atau relatif mudah dipecah, seperti Enigma. Sejarah sandi-sandi ini begitu menarik dan luas sehingga kami tidak mungkin dapat menjelaskannya secara adil di sini. Bagi pembaca yang tertarik dengan latar belakang yang lebih menyeluruh, kami merekomendasikan buku-buku berikut yang membahas evolusi kriptografi yang menarik. Pembaca yang telah memiliki pemahaman yang baik tentang kriptografi dasar mungkin ingin langsung beralih ke studi kasus di bagian selanjutnya dalam bab ini.

Dalam bab ini, kita mulai dengan prinsip dasar kriptografi, termasuk deskripsi sandi yang aman tanpa syarat: one-time pad (OTP). Dan dengan menggunakan one-time pad sebagai landasan, kami mengembangkan wacana di seluruh cipher kunci simetris modern: keystream dan codebook. Untuk masing-masingnya, kami membahas mode operasi yang berlaku, sinkronisasi kriptografi, dan aspek manajemen kunci yang diperlukan untuk sistem kriptografi yang aman.

Kami kemudian membahas kriptografi kunci publik dan penggunaannya yang luas dalam kriptografi modern untuk menetapkan kunci rahasia secara dinamis antara dua entitas yang berkomunikasi. Kami mengikuti deskripsi mendasar dengan contoh praktis kriptosistem digital, menunjukkan ancaman dan kerentanan serta teknik mitigasi untuk melawan ancaman ini. Meskipun bab ini memberikan gambaran yang baik tentang prinsip dan metode operasi kriptografi simetris (stream dan block cipher) dan asimetris (kunci publik), kami tidak mencoba untuk menjelaskan secara rinci spesifikasi algoritma kriptografi. Spesifikasi rinci untuk algoritma yang tidak terklasifikasi tersedia di domain publik dan dapat diakses dengan mudah di Internet. Untuk kriptografi simetris, bacaan yang disarankan adalah Standar Enkripsi Lanjutan (AES) dari Institut Standar dan Teknologi Nasional (NIST) Publikasi Standar Pemrosesan Informasi Federal (FIPS PUBS) AS:

- NIST FIPS PUB 197, "Standar Enkripsi Lanjutan, 2001," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
- Untuk kriptografi asimetris, algoritma berikut digunakan secara luas saat ini:
- RSA: Lihat PKCS #1: "Standar Kriptografi RSA, Versi 2.1," ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf
- Diffie-Hellman: Lihat PKCS #3: "Standar Perjanjian Kunci Diffie-Hellman," ftp://ftp.rsasecurity.com/pub/pkcs/doc/pkcs-3.doc

- Kurva Elliptic (ECC): Lihat NIST FIPS PUB 186-3, "Digital Signature Standard, 2009," http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf. Juga, lihat Publikasi Khusus NIST 800-56A, "Rekomendasi untuk Skema Pembentukan Kunci Berpasangan Menggunakan Kriptografi Logaritma Diskrit" (Maret, 2007).

Catatan akhir bab ini menyertakan sumber tambahan untuk bacaan lebih lanjut.

Terakhir, kami juga membahas topik yang sering disalahpahami atau diabaikan oleh perancang sistem tertanam: bagaimana merancang dan menggabungkan elemen kriptografi yang aman ke dalam sistem tertanam yang memiliki persyaratan kerahasiaan atau otentikasi yang berlaku. Kami menggambarkan hal ini dengan menggunakan beberapa studi kasus di bagian akhir bab ini.

4.2 PANDUAN KRIPTOGRAFI PEMERINTAH AS

Ada dua sumber utama panduan dan standar kriptografi yang berasal dari pemerintah AS: NIST (disebutkan sebelumnya) dan NSA. Selain FIPS PUBS, NIST juga menulis "publikasi khusus", seperti SP 800-57, yang memberikan rekomendasi untuk manajemen kunci dan dirujuk beberapa kali dalam bab ini dan Bab 5 tentang protokol perlindungan data. NIST mengumpulkan praktik komersial yang baik dari komunitas kriptografi di seluruh dunia dan menyaring praktik ini menjadi standar dan rekomendasi penting, NSA dipercaya untuk melindungi informasi paling sensitif (rahasia) pemerintah AS.

Pengembang sistem tertanam harus memahami panduan kriptografi NSA, meskipun sertifikasi NSA tidak diperlukan untuk sebagian besar sistem tertanam. Sertifikasi kriptografi FIPS 140-2 dan NSA serta latar belakang tambahan tentang strategi kriptografi NSA dibahas nanti dalam bab ini.

4.2.1 NSA Suite B

Untuk melengkapi kebijakan nasional yang ada mengenai penggunaan AES untuk melindungi Informasi Keamanan Nasional (CNSSP-15), [1] NSA mengumumkan kriptografi Suite B pada konferensi RSA tahun 2005.

NSA Suite B mencakup serangkaian algoritme kriptografi tidak terklasifikasi yang menyediakan berbagai layanan kriptografi yang diperlukan dalam kriptografi modern, termasuk enkripsi, hashing, tanda tangan digital, dan pertukaran kunci.

Seluruh rangkaian algoritma kriptografi dimaksudkan untuk melindungi sistem dan informasi keamanan nasional yang bersifat rahasia (Sangat Rahasia dan Rahasia) dan tidak rahasia. Situs web NSA Suite B berisi referensi ke algoritma pilihan.[2] Selain itu, standar Internet Engineering Task Force (IETF), dalam bentuk RFC, mendokumentasikan panduan Suite B sehubungan dengan penggunaan kriptografi di banyak protokol keamanan jaringan populer, seperti sebagai IPsec. Peran Suite B dalam strategi sertifikasi dan persetujuan NSA terkini akan dibahas kemudian dalam bab ini.

Meskipun panduan Suite B disebutkan di seluruh bab ini saat kita membahas kelas fungsi kriptografi yang dapat diterapkan, Tabel 4.1, yang berisi daftar pilihan algoritma dan

[1] Fact Sheet No. 1 for the National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information (CNSSP-15) (June 2003).

[2] NSA Suite B Cryptography. www.nsa.gov/ia/programs/suiteb_cryptography

panjang kunci untuk perlindungan data rahasia dan sangat rahasia, dapat berfungsi sebagai referensi cepat yang berguna.

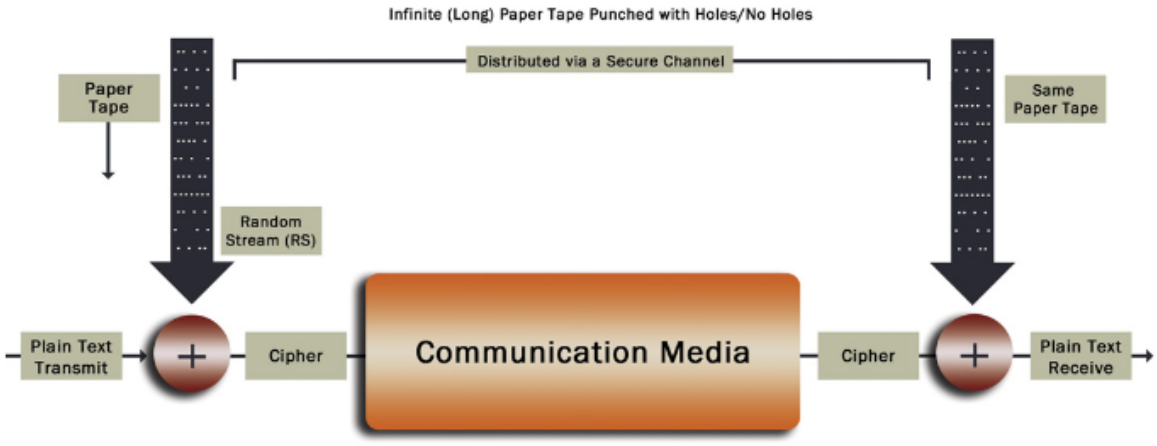
Tabel 4.1: Panduan NSA suite B untuk algoritma dan panjang kunci

Fungsi	Algoritma	Ukuran Kunci Rahasia	Ukuran Kunci Sangat Rahasia
Kunci simetris	AES	128	256
Tanda tangan kunci publik	ECDSA	256	384
Perjanjian utama	ECDH	256	384
Hash	SHA-2	256	384

4.3 PAPAN SEKALI PAKAI

Sepanjang sejarah yang tercatat, kriptografi telah digunakan untuk melindungi informasi. Sistem one-time pad, pertama kali diperkenalkan pada tahun 1917 oleh Gilbert Vernam, yang dianugerahi paten atas penemuan tersebut,[3] adalah skema enkripsi yang sangat sederhana yang dapat diimplementasikan, secara harfiah, dengan beberapa buku catatan. Yang juga sangat penting adalah bukti teoritis formal tentang keamanan OTP oleh Claude Shannon pada tahun 1945 (dideklasifikasi dan diterbitkan pada tahun 1949).[4] Secara khusus, Shannon membuktikan kerahasiaan sempurna dari OTP: musuh, yang menyadap komunikasi terenkripsi (ciphertext) dan memiliki sumber daya tak terbatas untuk mencoba memulihkan pesan asli (plaintext) dari pesan terenkripsi, terbukti tidak mampu membedakan ciphertext dari data acak. Hal ini juga disebut sebagai keamanan tanpa syarat; OTP tidak memberikan informasi tentang pesan aslinya.

Karena keamanannya yang terbukti, tanpa syarat, dan sederhana, one-time pad sering dianggap sebagai dasar dari semua sistem enkripsi yang baik. Karena kriptografi hampir selalu melibatkan trade-off dan kompromi, one-time pad tetap menjadi mercusuar kemurnian yang menyegarkan di dunia kriptografi. Emulasi dan implementasi digital dari sistem one-time pad yang tidak dapat dipecahkan dapat dijelaskan dengan terlebih dahulu mengembangkan biner digital yang setara dengan one-time pad, seperti yang ditunjukkan pada Gambar 4.1.



Gambar 4.1 Setara biner dengan pad satu kali.

[3] Gilbert VS. Secret Signaling System. U.S. Patent 1310719, issued July 22, 1919.
[4] Claude S. Communication Theory of Secrecy Systems. Bell System Technical Journal 28(4): 656e715. 1949.

Dua pihak ingin berkomunikasi dan melindungi komunikasi dari penyadap. Salah satu pihak, sebelum melakukan komunikasi yang aman, mengembangkan pita kertas yang sangat panjang lubang untuk logika atau biner dan tidak ada lubang untuk logika atau nol biner. Urutan pukulan pita kertas ditentukan secara acak dengan pelemparan koin, dengan kepala menandakan logika satu, dan ekor menandakan logika nol. Setiap peristiwa pelemparan koin memiliki kemungkinan 50% untuk menjadi kepala atau ekor, dan dengan demikian pita kertas berisi urutan yang benar-benar acak. Setelah pita kertas panjang ini dilubangi seperti dijelaskan, satu dan hanya satu salinan duplikat dibuat dari pita kertas tersebut. Kami sekarang memiliki dua pita kertas identik yang berisi aliran acak satu dan nol yang sama. Salah satu pita kertas dikirimkan dengan aman (diskusi tentang mekanisme kurir potensial tidak disertakan dalam diskusi ini dan tidak bergantung pada konsep enkripsi inti yang kami jelaskan) ke ujung lain dari tautan komunikasi, dan kemudian kami membuat protokol komunikasi aman berikut.

Pada akhir transmisi, pesan yang akan dienkripsi diinterpretasikan sebagai string yang setara dengan satu dan nol. Misalnya, pesan “halo” dapat dikonversi ke urutan biner enam byte (total 48 bit) di mana setiap byte berisi representasi ASCII digital dari karakter string yang sesuai (diikuti dengan akhiran nol). Kami menyebut string ini transmisi teks biasa (PTT). PTT sekarang dioperasikan dengan fungsi eksklusif - atau (XOR) dengan aliran acak dari pita kertas (Tabel 4.2).

Tabel 4.2: Eksklusif-atau fungsi

PTT	Aliran Acak	Sandi
0	0	0
0	1	1
1	0	1
1	1	0

Tabel 4.3: Urutan pad satu kali: penerimaan teks biasa (PTR) cocok dengan pengiriman teks biasa (PTT) ketika aliran acak sama

Pemancar			Penerima		
PTT	**RS**	**CTT**	**CTR**	**RS**	**PTR**
1	0	1	1	0	1
0	1	1	1	1	0
1	0	1	1	0	1
0	1	1	1	1	0
1	1	0	0	1	1
1	1	0	0	1	1
1	1	0	0	1	1
0	1	1	1	1	0
0	1	1	1	1	0
0	0	0	0	0	0
1	0	1	1	0	1
1	0	1	1	0	1
0	1	1	1	1	0

0	0	0	0	0	0
0	0	0	0	0	0
0	1	1	1	1	0
1	1	0	0	1	1

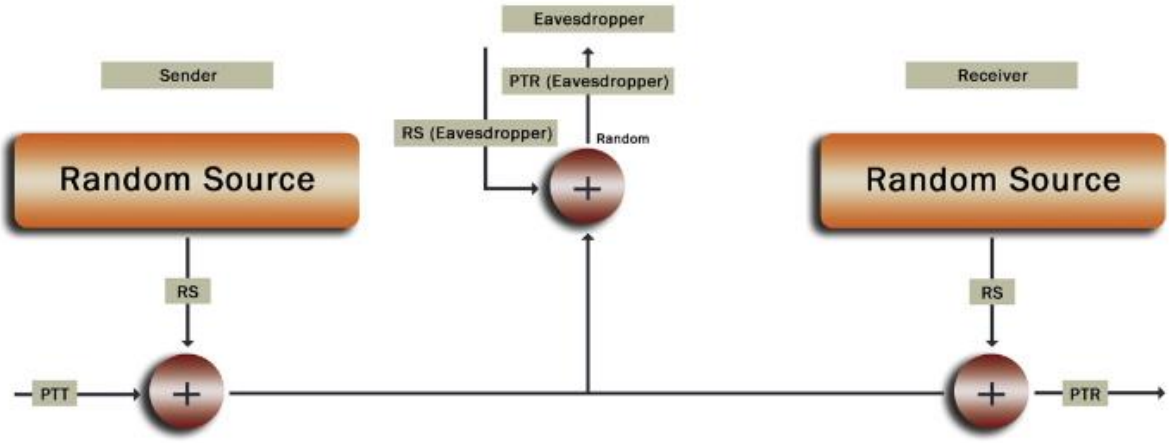
Keluaran XOR yang dioperasikan pada seluruh PTT disebut cipher text transmit (CTT). Untuk setiap bit PTT yang diproses, pita kertas dipindahkan ke posisi lubang berikutnya/tanpa lubang. Bagian pita kertas yang sudah terpakai, saat digerakkan ke depan, akan dibuang secara permanen (tidak boleh digunakan kembali). Kami mengikuti proses ini dengan menerapkan sedikit aliran acak pita kertas ke sedikit PTT hingga seluruh string PTT dienkripsi atau dienkripsi. Kami kemudian mengirim pesan terenkripsi melalui saluran komunikasi. Untuk contoh ini, asumsikan kabel tersebut adalah kabel point-to-point, seperti kabel fiber, yang dihubungkan dari pengirim pesan ke penerima pesan.

Setelah seluruh pesan terkirim, penerima melanjutkan untuk menguraikan atau mendekripsi pesan dengan mengikuti proses yang sama seperti pemancar: untuk setiap bit teks sandi yang diterima (CTR, karena CTT menjadi CTR di sisi penerima), kami XOR RKT dengan aliran acak pita kertas duplikat. Berdasarkan sifat fungsi XOR, prosesnya dapat dibalik, dan dengan demikian kita memperoleh penerimaan teks biasa (PTR), yang cocok dengan PTT. Sebagai contoh, asumsikan PTT dan aliran acak (RS) adalah sebagai berikut:

RT: 10101110001100001
RS: 01011111100010011
Operasi XOR ditunjukkan pada Tabel 4.3.

Seperti yang ditunjukkan pada tabel, PTT cocok dengan PTR, dan kami berhasil mengenkripsi dan mendekripsi pesan yang berisi string satu dan nol. Pesannya bisa sangat panjang (asalkan pita kertasnya sangat panjang). Untuk setiap pesan PTT berikutnya, kami mengikuti proses yang sama, membuang pita bekas sambil menariknya ke depan, hingga pita kertas habis. Untuk pesan selanjutnya, kami memulai kembali seluruh proses, menghasilkan dan mendistribusikan pita kertas acak baru.

Secara intuitif, kita dapat menyatakan bahwa metode penyandian pesan ini aman tanpa syarat. Kami berasumsi bahwa pengirim dan penerima dapat menjaga keamanan fisik masing-masing teks biasa; dengan demikian, penyadap yang memantau saluran komunikasi hanya dapat memperoleh teks sandi (CTTor CTR). Karena penyadap tidak memiliki aliran acak yang dikodekan pada pita kertas, mencoba menebak bit teks biasa apa pun dari teks sandi akan menghasilkan 50% kemungkinan dia mendapatkan bit teks biasa yang salah. Dengan demikian, penyerang tidak dapat membedakan pesan terenkripsi dari data acak (lihat Gambar 4.2). Dengan demikian kita dapat mengklaim bahwa sistem tersebut aman tanpa syarat.



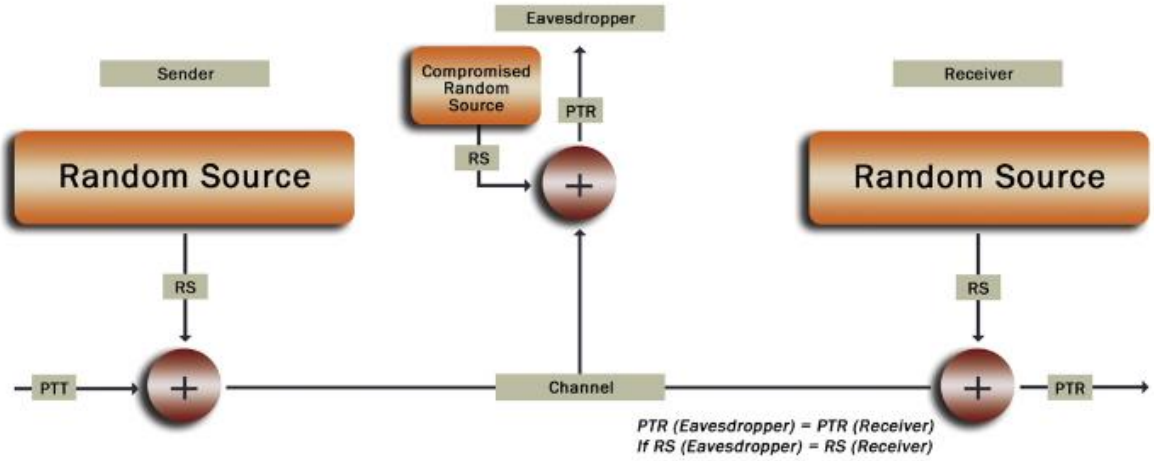
Gambar 4.2 Seorang penyadap tanpa akses ke sumber acak one-time pad asli tidak dapat memulihkan teks biasa.

Dengan teknologi saat ini, sistem one-time pad sangat mudah diterapkan. Daripada menggunakan pita kertas yang panjang, kita dapat menggunakan sepasang perangkat memori duplikat, seperti USB thumb drive, untuk menyimpan urutan acak dan mengirimkan ke kedua ujung saluran komunikasi. Dengan thumb drive (tersedia sejak 2010), hingga 32 miliar byte pesan dapat dikomunikasikan dengan aman menggunakan metode ini.

Meskipun sistem ini sederhana dan mudah diterapkan, sistem ini tidak efisien karena memerlukan aliran acak sepanjang pesan itu sendiri dan menimbulkan tantangan logistik dalam mendistribusikan media secara aman ke semua calon komunikator. Lebih jauh lagi, kompromi terhadap aliran acak yang dikodekan pada media flash menghasilkan kompromi terhadap semua pesan antara pasangan node mana pun yang pernah dikirim menggunakan media tersebut, seperti yang ditunjukkan pada Gambar 4.3.

Lebih jauh lagi, jika satu node dikompromikan dalam sistem pita kertas bersama, pita kertas bersama tersebut harus dicabut dari penggunaannya, sehingga memerlukan regenerasi dan redistribusi urutan acak baru ke semua node tersisa yang tidak dikompromikan. Sangat mudah untuk melihat bagaimana distribusi fisik dan perlindungan kaset kertas bersama dengan cepat menjadi tidak dapat dipertahankan. Secara umum, merupakan ide yang buruk untuk berbagi satu aliran acak jangka panjang di antara sejumlah besar node.

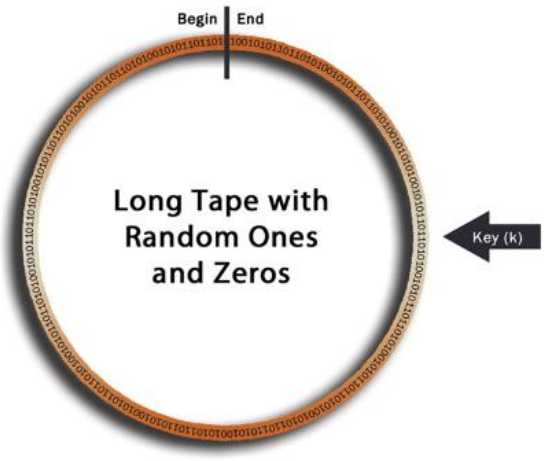
Jika setiap pasang komunikator unik menggunakan pita kertas unik per pasang, maka kompromi pita kertas tunggal hanya akan mempengaruhi satu saluran komunikasi antara sepasang node. Semua pasangan node lainnya dan komunikasinya tidak terpengaruh.



Gambar 4.3 Sumber acak yang dikompromikan memungkinkan penyadap memulihkan semua pesan.

Bagaimana kita dapat mempertahankan properti keamanan tanpa syarat yang kita inginkan sambil menghindari masalah distribusi aliran panjang? Salah satu caranya adalah dengan mendistribusikan aliran acak pendek dengan aman dan menggunakan urutan pendek ini untuk menghasilkan aliran acak bersama yang panjang secara lokal dan deterministik. Hal ini akan meningkatkan efisiensi, dan kita hanya perlu melindungi aliran acak pendek yang didistribusikan ke semua node. Terlebih lagi, jika kita dapat mengimplementasikan sistem tersebut secara efisien baik pada perangkat keras maupun perangkat lunak, maka kita dapat menjamin bahwa sistem tersebut aman.

Untuk menjelaskan konsep penggunaan aliran pendek untuk menghasilkan aliran yang panjang, asumsikan bahwa pita kertas yang ditunjukkan pada Gambar 4.1 tidak terlalu panjang tetapi diikat ujung ke ujung. Pita yang dilingkarkan ini masih berisi aliran acak, namun aliran acak akan berulang setelah perulangan menyelesaikan satu siklus. Kami ingin menerapkan loop terbatas ini sedemikian rupa sehingga pemancar dan penerima menghasilkan aliran acak panjang yang sama menggunakan aliran acak pendek yang sebelumnya didistribusikan dengan aman, seperti yang ditunjukkan pada Gambar 4.4. Urutan masukan disebut kunci, dan aliran panjang yang dihasilkan sekarang disebut aliran kunci acak.

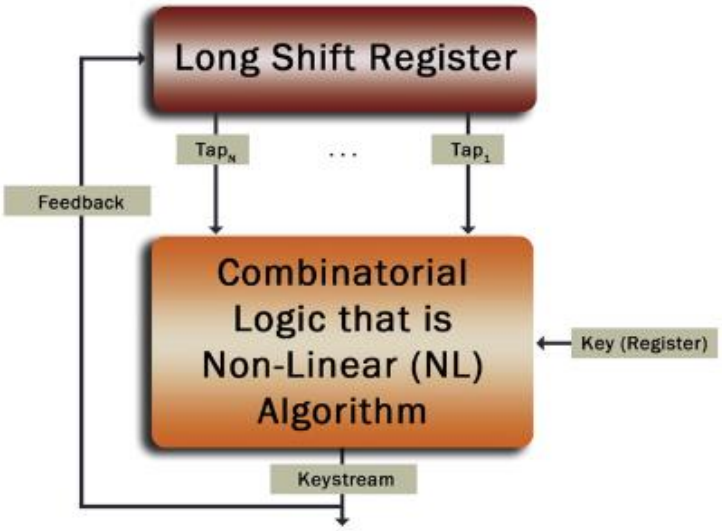


Gambar 4.4 Keystream acak panjang.

Hanya kunci yang didistribusikan dengan aman ke penerima, dan karena kedua titik akhir menggunakan mekanisme perulangan yang sama untuk menghasilkan aliran kunci acak, kami yakin bahwa kedua titik akhir memiliki aliran kunci acak yang sama. Perhatikan bahwa sebelum mengenkripsi pesan apa pun, kita harus menyinkronkan keystream panjang di kedua ujungnya; dengan kata lain, bahkan jika kedua ujung memiliki kunci yang sama, jika salah satu ujung memulai keystream pada titik yang berbeda dalam loop panjang, mereka tidak akan dapat berkomunikasi karena mereka tidak menggunakan urutan bit acak yang sama.

Dalam praktiknya, menduplikasi urutan pendek untuk membuat loop panjang adalah pendekatan yang salah: urutan yang berulang akan mudah dilihat oleh penyerang, sehingga mengalahkan properti keacakan yang diperlukan. Sebaliknya, kita harus menggunakan mekanisme yang mengambil kunci acak pendek dan menghasilkan aliran panjang yang juga acak.

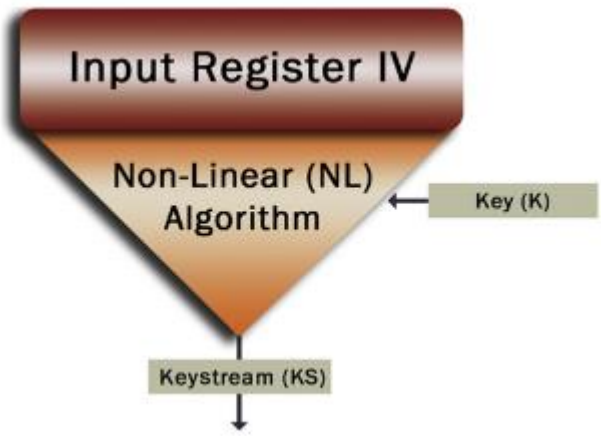
Kita dapat mengimplementasikan keystream ini secara digital yang diperoleh dari kunci yang relatif pendek. Pembuatan keystream ini digerakkan oleh algoritma kriptografi dan dapat diimplementasikan pada perangkat keras, perangkat lunak, atau kombinasinya, bergantung pada kecepatan data enkripsi yang diperlukan dan kecepatan komputer yang menjalankan perangkat lunak tersebut. Untuk aplikasi suara, kecepatan data yang diperlukan mungkin tidak lebih tinggi dari 64 kilobyte per detik, sedangkan enkripsi video mungkin memerlukan gigabit per detik. Potensi implementasi keystream ditunjukkan pada Gambar 4.5. Meskipun implementasi ini tidak layak untuk sistem kriptografi yang aman, kami menggunakannya untuk menyoroti konsep tersebut.



Gambar 4.5 Generator keystream yang disederhanakan.

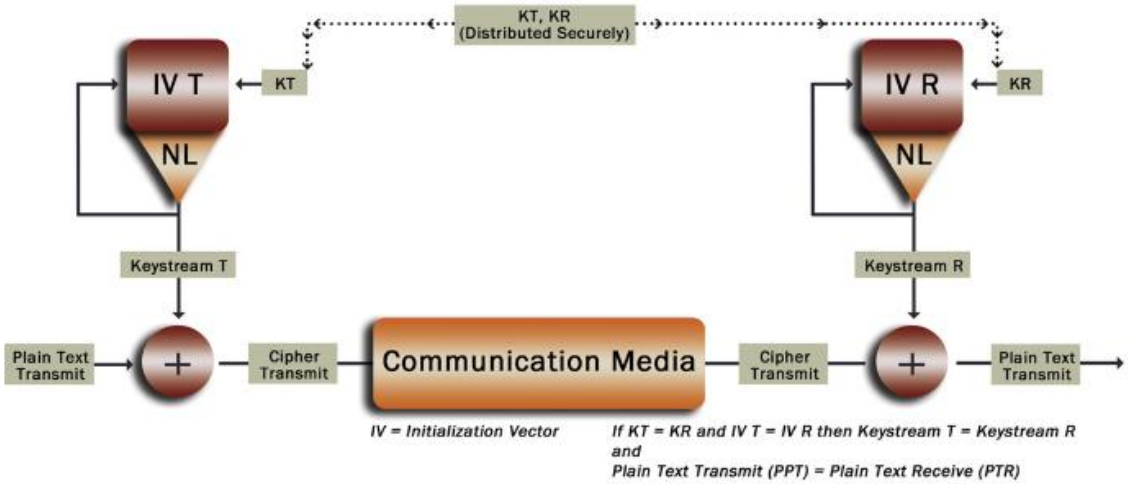
Keystream diimplementasikan menggunakan register geser yang panjang (80 bit atau lebih besar untuk menghasilkan loop yang cukup panjang), yang tapnya (tahap keluaran bit dari register geser) digabungkan dengan elemen kombinatorial non-linier dan kuncinya. Blok kombinatorial ini secara menyeluruh mencampurkan keluaran register geser dengan kunci dan disebut algoritma kriptografi keystream. Kita menyambungkan output keystream ke tahap pertama dari register geser, dan jika kita terus-menerus mencatat konfigurasi ini, kita

memperoleh pita loop panjang digital yang dihubungkan ujung ke ujung, menghasilkan keystream acak yang siklusnya berulang setelah 2 hingga N (2^N) bit, dimana N adalah jumlah bit dalam register geser. Register geser diinisialisasi dengan urutan acak N bit, dan kami menyatakan keadaan awal ini sebagai vektor inisialisasi (IV). Vektor inisialisasi adalah titik awal dari rekaman digital kita. Kami secara simbolis mewakili generator keystream ini seperti yang ditunjukkan pada Gambar 4.6.



Gambar 4.6 Representasi simbolis dari generator keystream.

Kami sekarang telah mengembangkan sistem kriptografi seperti yang ditunjukkan pada Gambar 4.7.



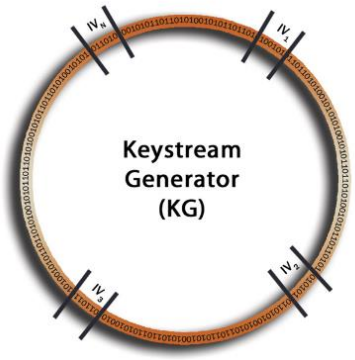
Gambar 4.7 Kriptosistem berbasis keystream.

Gambar 4.7 identik dengan Gambar 4.1, kecuali one-time pad telah diubah menjadi generator keystream, diimplementasikan secara identik pada kedua ujungnya dan digerakkan oleh kunci umum yang sebelumnya didistribusikan dengan aman melalui kurir atau cara lain dan dimuat ke dalam generator keystream di kedua ujungnya. Seperti ditunjukkan pada Gambar 4.7, agar PTT cocok dengan PTR, kita harus memastikan bahwa IV pemancar (IVT) cocok dengan IV penerima (IVR) dan kunci pemancar (KT) cocok dengan kunci penerima (KR).

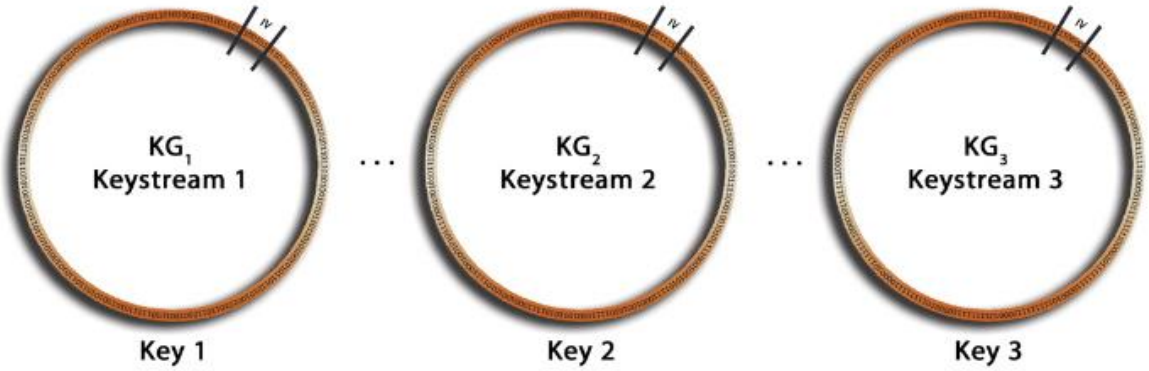
Bagaimana cara menyinkronkan IVT ke IVR? Salah satu caranya adalah dengan mengirimkan infus dari pemancar ke penerima. Penerima sekarang memiliki urutan inisialisasi

yang diperlukan dan mengetahui bahwa teks sandi berikutnya akan dihasilkan menggunakan awal keystream yang dihasilkan oleh IV ini. Seorang penyadap yang menerima IV juga akan dapat menyinkronkan generator kunci jahatnya ke titik awal yang sama dengan pemancar dan penerima yang sah. Namun, PTR penyadap akan bersifat acak karena ia tidak memiliki kunci dan oleh karena itu tidak dapat menghasilkan aliran kunci yang setara dengan pemancar dan penerima. Jika kunci dikompromikan dan penyadap mengetahui algoritma kriptografi (seperti yang sering terjadi), penyadap akan dapat mendekripsi semua pesan dengan menggunakan sinkronisasi IV yang sama. Untuk setiap pesan, pemancar menghasilkan IV baru untuk menghindari penggunaan kembali keystream sebelumnya (lihat Gambar 4.8).

Apa yang terjadi jika kunci baru dimasukkan ke generator kunci kita? Keystream acak baru dihasilkan, dan keystream baru tidak cocok dengan keystream lama, seperti yang ditunjukkan pada Gambar 4.9.



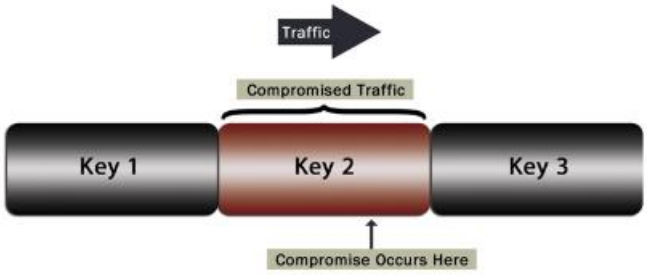
Gambar 4.8 IV baru dihasilkan untuk setiap periode kripto.



Gambar 4.9 Aliran kunci yang berbeda untuk kunci yang berbeda.

Dengan membuat kunci baru secara berkala, kita memiliki rekaman digital yang panjangnya sembarang dan keystream acak yang panjangnya sembarang. Hal ini memungkinkan sistem kripto untuk menangani lalu lintas pesan selama masa pakai peralatan tanpa harus menggunakan kembali keystream acak yang sama. Jika suatu kunci disusupi, semua teks sandi yang terkait dengan kunci tersebut akan disusupi, namun tidak teks sandi yang dienkripsi dengan kunci lainnya.

Masa penggunaan kunci dalam sistem kriptografi disebut periode kripto kunci. Misalkan seorang penyadap menyimpan semua ciphertext dan kemudian melancarkan serangan dengan mencoba semua kunci untuk mendekripsi konten secara mendalam. Dengan asumsi ia berhasil, ia hanya dapat mendekripsi pesan yang dihasilkan selama periode kripto kunci yang diretas (lihat Gambar 4.10). Setiap pesan yang diterima dengan kunci lain akan memerlukan serangan menyeluruh baru yang sangat memakan waktu dan paling buruk tidak dapat dilakukan untuk kunci dengan panjang yang cukup.



Gambar 4.10 Kompromi pada kunci sesi mengakibatkan kompromi lalu lintas hanya dalam periode kripto kunci yang dikompromikan.

Tabel 4.4 Fungsi kerja untuk beberapa cipher

	Aman Kombinasi Standar	Kunci Sandi	Variabel 120-Bit
Kombinasi	10^6	60	2^{120} =1.33 x 10^{36}
Tingkat percobaan	10 menit	1/detik	1/ µdetik
Berarti waktu untuk istirahat	833 jam 34 hari	30 detik	0,72 x 10^{25} hari 2,1 x 10^{22} tahun 2,1 x^{20} abad

One-time pad dan sistem keystream terkait menggunakan kunci simetris: pemancar dan penerima harus menyetujui dan menggunakan kunci yang sama agar dapat berkomunikasi dengan aman. Keamanan sistem terletak pada kuncinya, yang harus tetap dirahasiakan. Setelah kunci dibocorkan, disalin, atau disusupi, siapa pun dapat mendekripsi atau mengenkripsi pesan meskipun algoritme kripto adalah hak milik dan bukan milik publik.

Untuk tujuan menentukan kekuatan kriptografi, algoritme kepemilikan atau rahasia selalu dianggap rusak (dipahami sepenuhnya dan dapat direproduksi oleh penyerang) melalui rekayasa balik atau cara lain.

Terlebih lagi, jika sebagian dari kunci dikompromikan, sistem akan lebih rentan terhadap serangan menyeluruh dengan mengurangi ukuran ruang kunci yang harus dicari. Misalnya, ketika kunci 80-bit biasanya mengharuskan penyerang untuk mencoba 2^{80} kunci yang berbeda, penyerang hanya memerlukan 2^{40} pencarian jika setengah dari bit kunci telah dikompromikan.

Tabel 4.4 menyajikan beberapa contoh waktu yang diperlukan untuk membuka atau memecahkan gembok atau kunci kombinatorial relatif terhadap bit kebebasannya. Lamanya waktu yang diperlukan untuk memecahkan suatu kunci tertentu disebut fungsi kerjanya. Seperti yang ditunjukkan pada tabel, serangan menyeluruh untuk kunci 120-bit membutuhkan waktu yang sangat lama (RLT). Jika kunci tersebut sering diubah, seperti pada setiap pesan,

serangan menyeluruh akan menjadi sia-sia, bahkan dengan daya komputasi yang tidak terbatas.

4.3.1 Sinkronisasi Kriptografi

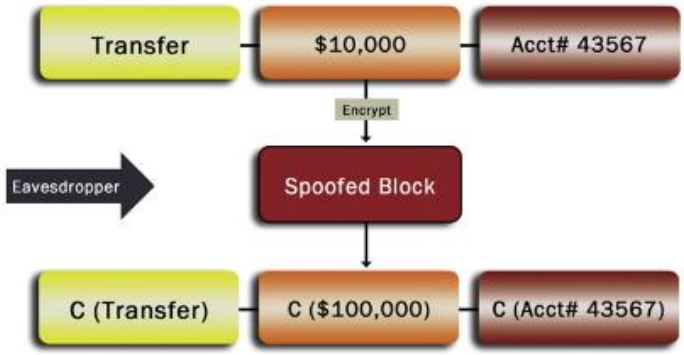
Seperti yang ditunjukkan pada Gambar 4.7, agar kriptosistem simetris berfungsi dengan baik, IV yang digunakan pada titik akhir pengirim dan penerima harus sama untuk urutan teks tersandi yang dihasilkan dengan IV. Biasanya, pengirim hanya mengirimkan IV secara jelas sebelum ciphertext untuk IV tersebut. Untuk sistem digital, sinkronisasi IV dapat terjadi hanya jika jam pengirim dan penerima berada dalam fase dan tersinkronisasi. Teknik sinkronisasi tambahan, termasuk pemulihan jam dari aliran data, harus digunakan ketika pemancar dan penerima terpisah secara fisik. Untuk keperluan pembahasan ini, kita asumsikan bahwa jam pengirim dan penerima sudah tersinkronisasi dan hanya IV yang memerlukan sinkronisasi. Dalam praktiknya, sinkronisasi jam biasanya ditangani oleh perangkat keras secara transparan kepada pengembang sistem atau perangkat lunak.

Seperti dijelaskan sebelumnya, untuk keystream yang panjang, register IV biasanya panjangnya 80 bit atau lebih. 80 bit ini harus diterima dengan sempurna tanpa kesalahan untuk mencapai sinkronisasi kriptografi (yaitu, IVT ¼ IVR). Jika salah satu bit IV yang diterima mengalami kesalahan, terdapat kemungkinan 50% bahwa setiap bit keystream yang dikirimkan akan tidak cocok dengan bit keystream penerima yang terkait. Sinkronisasi kriptografi tidak akan tercapai, dan hasil teks biasa yang diterima akan berupa data acak. Selain itu, titik akhir komunikasi tidak akan mengetahui bahwa sinkronisasi kriptografi telah gagal dan akan terus mengenkripsi dan mendekripsi hingga pemroses teks biasa (misalnya, perangkat lunak atau manusia) menentukan bahwa data yang diterima salah dan mengeluarkan perintah sinkronisasi ulang kriptografi.

Kesalahan bit terjadi secara alami pada beberapa media komunikasi, dan perancang sistem harus mempertimbangkan fakta ini ketika merancang protokol komunikasi untuk memastikan bahwa IV diterima dengan sempurna meskipun terdapat kesalahan komunikasi. Beberapa saluran komunikasi, seperti saluran frekuensi radio (RF), mungkin memiliki tingkat kesalahan bit sebanyak 10%. Perancang sistem harus menggunakan skema pengkodean kesalahan IV yang kuat, yang mungkin mencakup deteksi kesalahan, koreksi kesalahan, atau keduanya, tergantung pada kualitas saluran. Kesalahan pengkodean/penguraian kode dapat dilakukan pada perangkat keras, perangkat lunak, atau kombinasinya. Selain itu, protokol tingkat atas, seperti IPsec, dapat digunakan untuk menyediakan deteksi kesalahan dan perlindungan integritas IV yang dikirimkan.

Untuk keperluan pembahasan berikut, kami berasumsi IV telah diterima dengan sempurna ke dalam generator kunci penerima. Setelah IV dimuat, enkripsi/dekripsi dapat dilakukan meskipun terdapat kesalahan saluran. Seperti yang ditunjukkan pada Gambar 4.7, setelah IVT cocok dengan IVR, keystream yang dihasilkan di kedua ujungnya tidak terpengaruh oleh kesalahan transmisi apa pun dalam ciphertext. Kesalahan dalam ciphertext hanya akan mengakibatkan kesalahan bit yang sesuai dalam plaintext yang diterima; dengan asumsi sinkronisasi jam kuat, sinkronisasi kriptografi dipertahankan. Kita dapat menyatakan bahwa skema ini menunjukkan ekstensi kesalahan bit tunggal, cukup untuk banyak aplikasi kriptografi.

Misalnya, komunikasi suara dapat mentolerir beberapa kesalahan teks biasa, tidak mempengaruhi kejelasan karena tingginya redundansi dalam ucapan.



Gambar 4.11 Serangan spoofing penyadap.

Namun, untuk aplikasi data, kurangnya integritas dalam satu bit PTR pun dapat berdampak buruk pada isi pesan namun tetap terlihat oleh penerima sebagai pesan yang sah. Seorang penyadap yang memantau saluran pada waktu-waktu tertentu mungkin secara aktif memaksakan kesalahan pada saluran sehingga pesan yang diterima diubah. Misalnya, mari kita perhatikan pesan transaksi bank: “transfer 10.000 dolar ke rekening ini.” Seorang penyadap dapat mengubah isi pesan menjadi “mentransfer 100.000 dolar ke rekening ini” (lihat Gambar 4.11), yang bukan merupakan hasil yang diinginkan oleh bank namun sangat memuaskan bagi penyadap jika itu terjadi pada rekeningnya. Bahkan jika penyerang belum memecahkan enkripsi dan hanya dapat memodifikasi ciphertext, jika penyerang melihat pola dalam pesan dan mengetahui di mana jumlah dolar berada, ia dapat membuat tebakan tentang cara memasukkan ciphertext berbahaya. Jenis serangan ini disebut serangan pesan palsu, suatu bentuk serangan man-in-the-middle. Kriptografi dapat memberikan penanggulangan terhadap spoofing pesan dan banyak serangan lainnya dengan menyediakan mode operasi kriptografi khusus.

4.4 MODE KRIPTOGRAFI

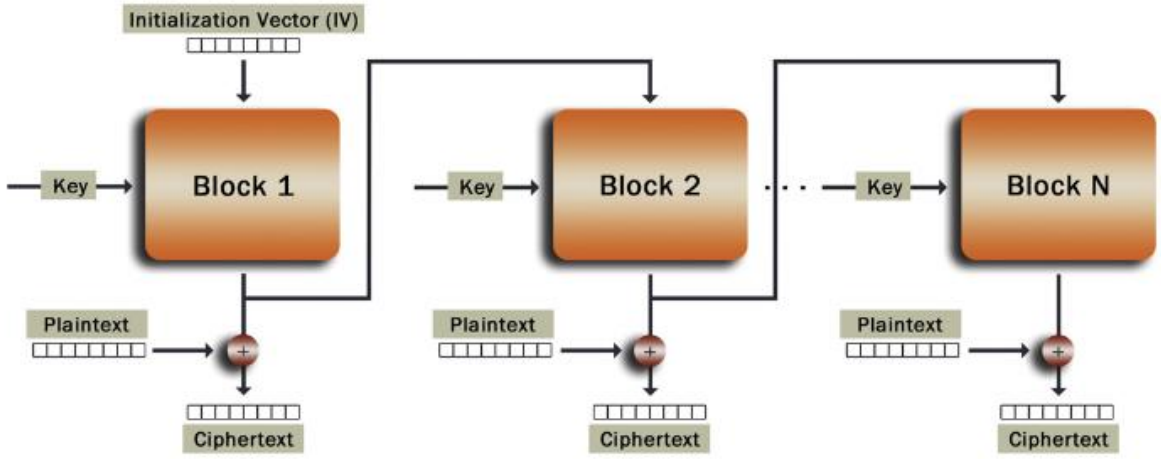
Mode kriptografi memungkinkan penerapan enkripsi pada kasus penggunaan tertentu yang memiliki persyaratan berbeda-beda:

- Beberapa mode mengaktifkan otentikasi asal dan perlindungan data secara bersamaan.
- Beberapa mode telah diciptakan hanya untuk meningkatkan efisiensi waktu eksekusi.
- Beberapa mode menyebarkan kesalahan bit dan dengan demikian menggagalkan serangan spoofing.
- Mode lain mengorbankan anti-spoofing dengan imbalan toleransi kesalahan bit.

4.4.1 Keluaran Umpan Balik

Seperti yang ditunjukkan pada Gambar 4.7, sandi keystream secara internal mengumpankan keluaran keystream kembali ke register IV. Umpan balik internal ini disebut *Output Feedback* (OFB) atau *Key Auto Key* (KAK). Mode kriptografi ini toleran terhadap kesalahan pada saluran tetapi rentan terhadap spoofing.

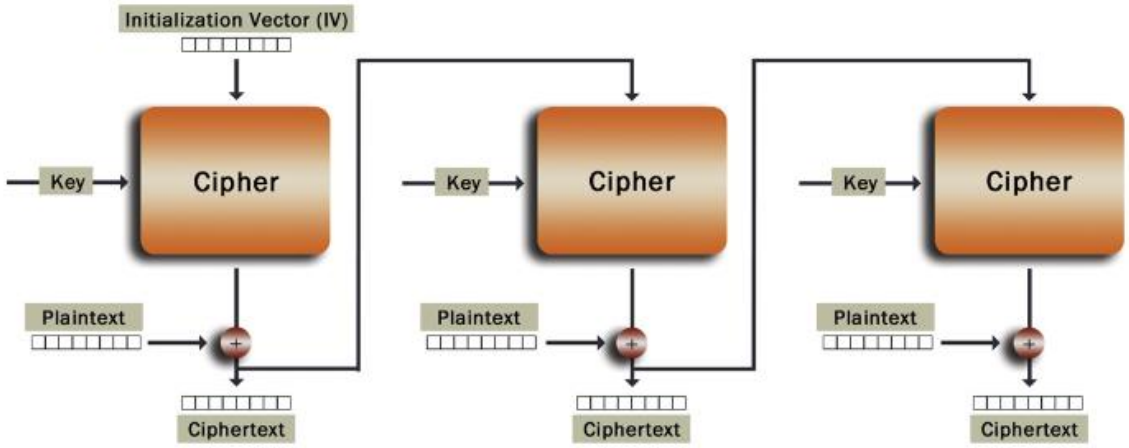
Satu atau lebih umpan balik dapat digunakan tergantung pada implementasinya. Misalnya, fungsi pembuatan kunci dapat disusun untuk meng-XOR setiap byte PTT dengan satu byte keystream dan satu byte umpan balik. Meskipun keluaran dihasilkan dalam blok byte (atau lebih besar), saluran komunikasi sering kali memerlukan serialisasi aliran. Konversi paralel ke serial biasanya ditangani secara transparan oleh antarmuka fisik dan/atau data link. Mode OFB yang diimplementasikan dengan N bit umpan balik ditunjukkan pada Gambar 4.12. Mode OFB menjamin kerahasiaan pesan tetapi tidak memberikan integritas pesan yang baik; seorang penyerang yang mampu memanipulasi saluran dapat mempengaruhi modifikasi setiap bit dalam teks biasa.



Gambar 4.12 Mode umpan balik keluaran (OFB).

4.4.2 Umpan Balik Sandi

Variasi OFB adalah mode Cipher Feedback (CFB) atau Cipher Text Auto Key (CTAK), seperti terlihat pada Gambar 4.13:



Gambar 4.13 Mode umpan balik sandi (CFB).

Tidak seperti OFB, mode CFB memasukkan output cipher ke dalam register IV, bukan ke keystream. Mirip dengan OFB, umpan balik sandi dapat berupa satu atau N bit. Karena keluaran N bit awal dari cipher digunakan sebagai IV, mode CFB tidak memerlukan IV eksplisit

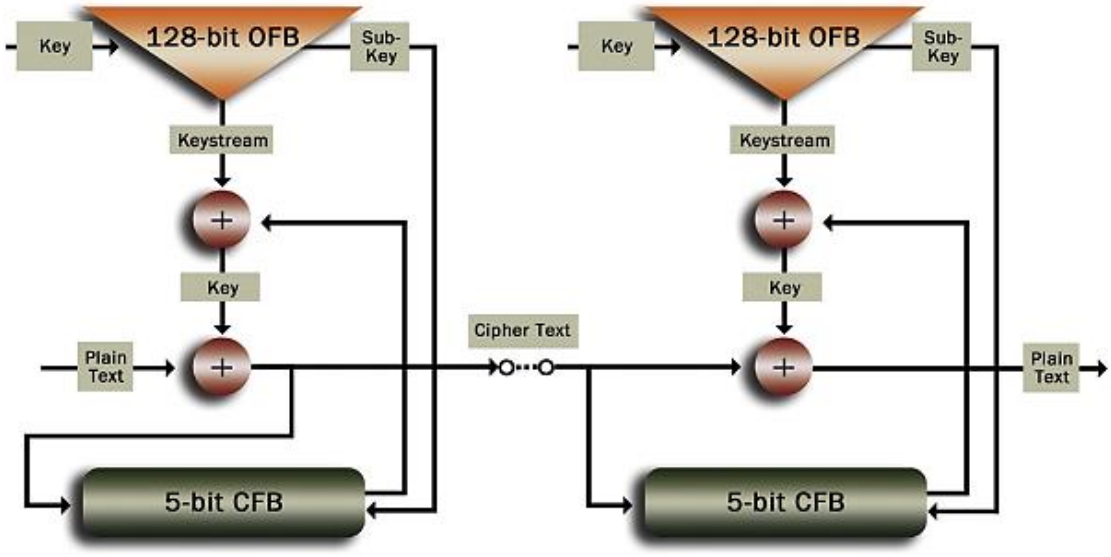
untuk ditransmisikan. Dengan demikian, CFB melakukan sinkronisasi mandiri, dengan aliran kunci pemancar dan penerima cocok segera setelah nilai register IV di kedua titik akhir cocok. Sinkronisasi kriptografi dicapai setelah N bit pemrosesan sandi, dimana N adalah panjang bit register IV.

Dengan CFB, kesalahan satu bit pada saluran meluas ke kesalahan N-bit pada hasil dekripsi. Titik akhir melakukan sinkronisasi ulang sendiri setelah N bit pemrosesan sandi. Pembesaran kesalahan ini memberikan perlindungan anti-spoofing, karena penyadap tidak lagi dapat mempengaruhi penetapan bit individu tertentu dalam teks biasa. Jika penyerang memodifikasi sedikit ciphertext saat transit, N bit plaintext akan rusak, namun rusak secara tidak terduga. Di sisi lain, CFB menunjukkan penurunan efisiensi pada saluran komunikasi yang bising karena kesenjangan sinkronisasi yang lebih panjang. Kurangnya ketahanan ini mungkin cukup untuk mengganggu komunikasi suara.

4.4.3 OFB dengan Perlindungan CFB

Mode hibrid ini menambah manfaat efisiensi OFB dengan perlindungan integritas data. Salah satu metode implementasi menggunakan generator kunci mode OFB sebagai enkripsi utama dan generator kunci CFB yang lebih kecil untuk perlindungan anti-spoofing (lihat Gambar 4.14).

Lima bit ekstensi kesalahan disediakan dalam contoh ini. Meskipun hal ini menggagalkan serangan spoofing sederhana, generator kunci CFB yang lebih besar (misalnya, umpan balik 32 bit atau lebih) memberikan perlindungan yang lebih baik.



Gambar 4.14 OFB dengan CFB anti-spoofing.

4.4.4 Keamanan Arus Lalu Lintas

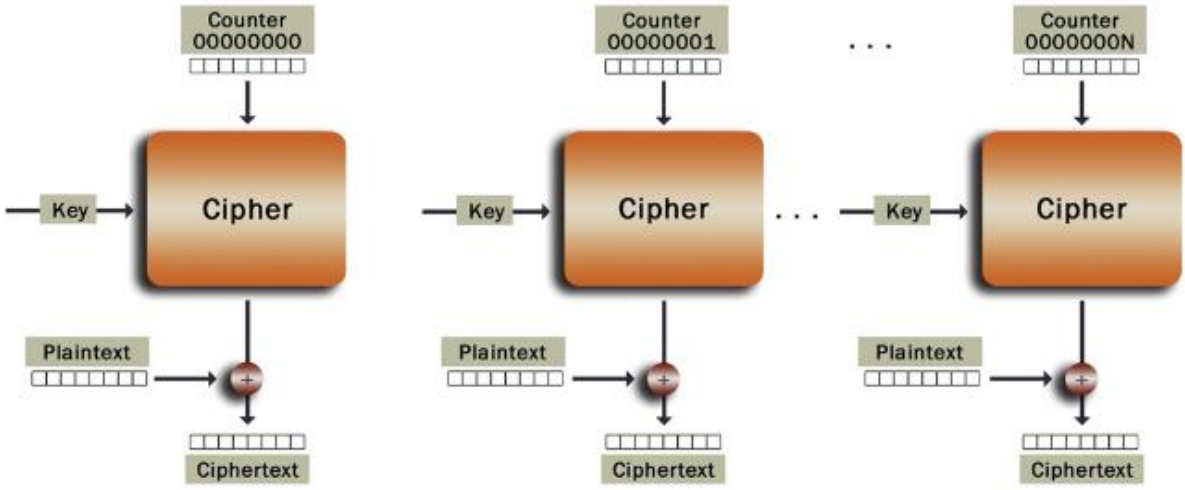
Sifat yang diinginkan dari algoritma dan protokol komunikasi yang aman, keamanan arus lalu lintas (TFS) bertujuan untuk mencegah penyadap menyimpulkan informasi pribadi dari pola komunikasi dan bukan dari data terenkripsi yang dikomunikasikan itu sendiri.

Misalnya, karena protokol *Secure Shell* (SSH) mengirimkan pesan untuk setiap penekanan tombol selama sesi interaktif, para peneliti telah menggunakan waktu antara

penekanan tombol sebagai masukan pada teknik analisis statistik untuk memulihkan kata sandi jauh lebih cepat daripada pencarian menyeluruh.[5] Menghapus struktur atau pengaturan waktu pola komunikasi menggagalkan serangan semacam ini. TFS adalah salah satu bentuk konsep keamanan transmisi yang lebih umum, yang bertujuan untuk melindungi terhadap serangan yang berada di luar cakupan serangan kriptanalitik terhadap data terenkripsi. Contoh lain dari keamanan transmisi adalah penggunaan teknik anti-jamming pada komunikasi frekuensi radio (RF). Kami menyebutkan TFS di sini karena ini berkaitan dengan pilihan mode enkripsi. Secara khusus, penggunaan vektor inisialisasi secara jelas dalam aliran komunikasi memungkinkan penyadap untuk menyimpulkan awal dan akhir pesan atau sesi. Ingatlah hal ini saat kita membahas lebih banyak mode di bagian berikut.

4.4.5 Mode Penghitung

Counter Mode, terkadang disebut Long Cycle Mode (LCM), ditunjukkan pada Gambar 4.15. LCM memiliki sifat keamanan yang mirip dengan OFB; perbedaan utamanya adalah generasi IV: di OFB, IV acak, tetapi IV LCM dihasilkan menggunakan nilai awal yang bertambah setelah setiap periode kripto (untuk sesi yang menggunakan kunci yang sama).



Gambar 4.15 Mode siklus panjang (penghitung).

Mirip dengan OFB, LCM tidak memiliki ekstensi kesalahan dan karenanya rentan terhadap spoofing pesan. Penghitung IV dapat menjadi generator urutan dengan panjang maksimum yang sebenarnya, memberikan manfaat tambahan dari keystream yang sangat panjang dan tidak berulang. Di LCM, penghitung IV dapat didasarkan pada jam waktu nyata. Ketika jam waktu nyata digunakan, sinkronisasi kriptografi dapat didasarkan pada waktu hari (TOD), nilai tanggal yang tidak pernah terulang. Jika TOD disinkronkan pada kedua titik akhir, transmisi IV tidak diperlukan. Oleh karena itu, mode counter menggunakan TOD berguna dalam keamanan arus lalu lintas, seperti yang dijelaskan sebelumnya.

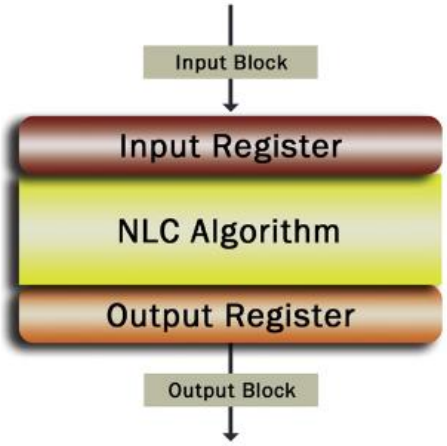
Manfaat lain dari pendekatan ini adalah penerapannya pada penerima siaran seperti set-top box digital. Banyak sistem seperti ini yang tidak memiliki komunikasi dua arah full-duplex. Sebaliknya, satu pemancar menyiarkan informasi ke sejumlah besar penerima satu

[5] Song D, et al. Timing Analysis of Keystrokes and Timing Attacks on SSH. Proceedings of the 10th Conference on USENIX Security Symposium 2001;vol. 10.

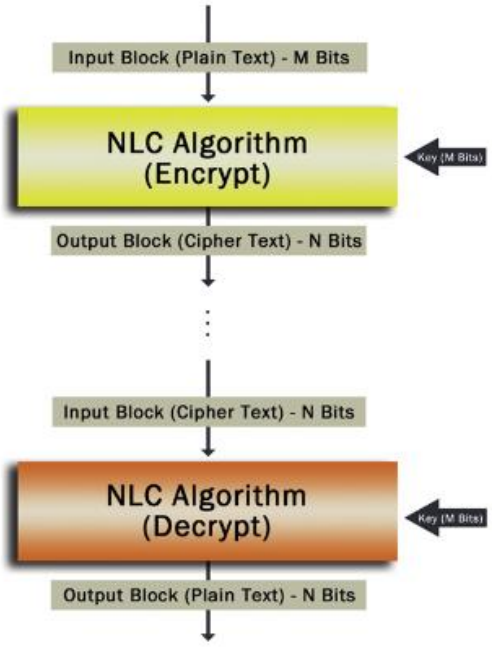
arah. Penggunaan mode penghitung berbasis TOD memungkinkan penerima untuk memasuki aliran komunikasi pada waktu yang berubah-ubah tanpa memerlukan sinkronisasi kriptografi yang eksplisit. Misalnya, penghitung TOD dapat bertambah satu kali sehari. Saat receiver dinyalakan, ia dapat dengan cepat memeriksa jam internal real-time dan mendapatkan penghitung TOD saat ini, menyediakan sinkronisasi instan ke konten siaran terenkripsi.

4.5 BLOK CIPHER

Register IV digunakan sebagai titik awal dari keystream cipher. Misalkan alih-alih menggunakan register IV sebagai titik awal, kita menggunakannya sebagai register masukan untuk teks biasa, seperti yang ditunjukkan pada Gambar 4.16, dan kemudian menerapkan algoritma kriptografi non-linier ke register masukan.



Gambar 4.16 Sandi blok yang digeneralisasi.



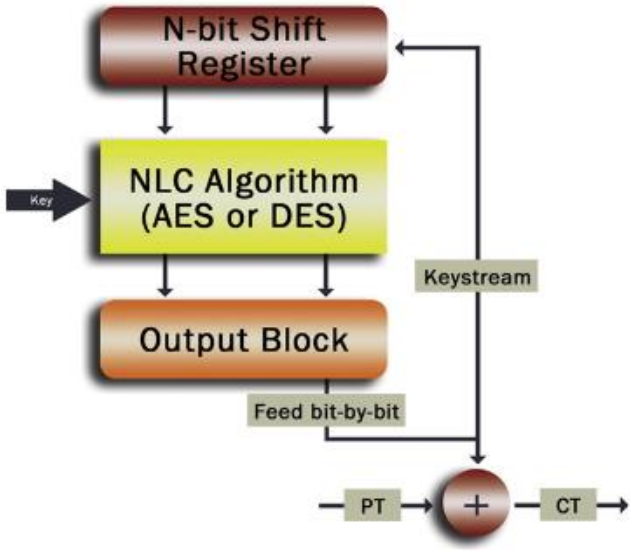
Gambar 4.17 Enkripsi dan dekripsi menggunakan N-bit block cipher.

Konfigurasi ini menghasilkan enkripsi blok teks biasa yang satu bloknya bisa sepanjang 128 bit untuk algoritma modern seperti AES. Block cipher berfungsi mirip dengan keystream

cipher tetapi memiliki sifat yang berbeda. Block cipher berguna untuk penyimpanan terenkripsi data yang tidak aktif, namun juga dapat digunakan untuk mengimplementasikan mekanisme komunikasi terenkripsi bit demi bit seperti keystream. Untuk block cipher, proses enkripsi dan dekripsi ditunjukkan pada Gambar 4.17.

Cipher blok mengubah blok masukan (rangkaian bit masukan dengan panjang tetap) menjadi blok keluaran yang merupakan rangkaian bit keluaran dengan panjang tetap yang sama. Sandi adalah fungsi dari kunci dan beberapa transformasi non-linier. Cipher blok (algoritma kriptografi) dapat menggunakan fungsi non-linear yang sama dengan yang digunakan untuk cipher keystream. Cipher secara menyeluruh mencampurkan bit-bit dari blok masukan sedemikian rupa sehingga setiap bit dari blok keluaran bergantung secara bersama-sama pada setiap bit dari blok masukan dan setiap bit dari kunci. Pencampuran blok masukan dengan kunci ini memberikan perlindungan terhadap pembacaan sandi oleh musuh.

Block cipher harus dirancang sedemikian rupa sehingga pengetahuan tentang sejumlah besar blok input dan output yang cocok tidak cukup untuk menentukan kunci selain melalui pencarian menyeluruh terhadap semua kemungkinan kunci untuk panjang kunci yang dapat diterapkan.



Gambar 4.18 Sandi blok dikonfigurasikan sebagai sandi keystream.

Pencampuran harus memastikan bahwa perubahan kecil pada blok masukan atau perubahan kecil pada kunci menghasilkan perubahan besar pada blok keluaran: tujuannya adalah 50% variasi bit keluaran ketika sandi dijalankan dengan blok masukan atau kunci yang memiliki sedikit berbeda.

Kebanyakan algoritma modern, seperti *Data Encryption Standard* (DES) atau *Advanced Encryption Standard* (AES), dirancang sebagai block cipher tetapi dapat dikonfigurasi sebagai keystream cipher dalam semua mode yang telah dibahas sebelumnya (OFB, CFB, LCM). Gambar 4.18 menggambarkan cipher blok yang dikonfigurasi sebagai cipher keystream.

Block cipher kadang-kadang disebut buku kode elektronik (ECB); nama-nama tersebut digunakan secara bergantian. Cipher ECB selalu menghasilkan blok keluaran yang sama dengan blok masukan tertentu. Dengan demikian, aliran teks biasa yang besar dapat dengan

mudah mengungkap pola dalam teks biasa. Hal ini juga membuat ECB rentan terhadap serangan teks sandi yang disusun ulang: menyusun ulang (atau memasukkan) blok teks sandi ke dalam saluran komunikasi menyebabkan penyusunan ulang teks biasa yang didekripsi sama persis, sehingga berpotensi mengubah pesan untuk menguntungkan penyerang tanpa penerima menyadari adanya gangguan. Semua sandi ECB harus memiliki proses penguraian yang dapat dibalik.

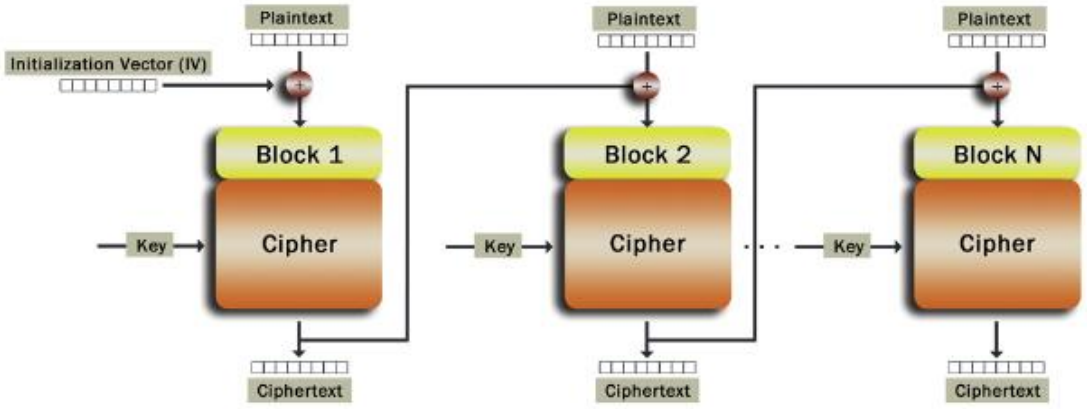
AES adalah cipher blok pilihan di NSA Suite B. Dua ratus lima puluh enam bit kunci digunakan untuk melindungi informasi rahasia, dan kunci 128-bit atau 256-bit digunakan untuk melindungi informasi rahasia. Panduan panjang kunci ini berlaku untuk semua mode enkripsi dan enkripsi yang diautentikasi. Panduan NIST saat ini menyatakan bahwa AES dengan kunci 128-bit akan mencukupi untuk produk tujuan umum hingga tahun 2030.

4.5.1 Mode Cipher Blok Kriptografi Tambahan

Mode ECB berguna untuk enkripsi data atau kunci kecil tetapi tidak digunakan untuk perlindungan informasi massal karena kerentanan blok yang disusun ulang. Mode Cipher Block Chaining (CBC), yang ditunjukkan pada Gambar 4.19, mengatasi serangan penataan ulang ini. Dalam mode CBC, setiap blok masukan (kecuali blok awal) di-XOR ke blok keluaran sebelumnya. Putaran umpan balik ini mencegah serangan penyusunan ulang karena modifikasi blok teks tersandi akan merusak semua blok yang didekripsi berikutnya.

Mode CBC hanya dapat digunakan dengan cipher blok. Kesalahan saluran bit tunggal mengakibatkan kerusakan pada blok teks biasa saat ini dan kesalahan terkait pada bit teks biasa pada blok berikutnya. Penyadap yang aktif, yang mencoba mengubah teks tersandi, akan menyebabkan hilangnya teks biasa yang dipulihkan dalam jumlah besar. Dengan demikian, mode CBC mencegah spoofing pesan. Karena properti ini, mode CBC terkadang digunakan dalam skema otentikasi pesan. Dalam skema seperti itu, kunci rahasia digunakan untuk menghasilkan Kode Otentikasi Pesan (MAC) dari teks biasa.

Jika penerima dapat menghitung MAC yang sama, maka ia yakin bahwa pesan tersebut sampai dengan utuh (integritas data) dan harus berasal dari pemegang kunci rahasia yang sama (keaslian asal). Salah satu skema tersebut adalah algoritma CBC Message Authentication Code (CBC-MAC). Dalam CBC-MAC, blok sandi terakhir dari enkripsi pesan CBC dimasukkan ke dalam sandi CBC sebagai blok masukan tambahan, dienkripsi, dan hasilnya ditambahkan ke keluaran teks sandi. Blok hasil ini adalah MAC. Di penerima, MAC dihasilkan menggunakan kunci yang sama dan diverifikasi untuk mencocokkan bit demi bit dengan MAC yang diterima. Karena blok ciphertext bergantung pada semua blok plaintext sebelumnya, blok terakhir yang cocok (kode autentikasi) dimungkinkan jika dan hanya jika pesan lengkap sampai secara utuh. MAC akan dibahas lebih mendalam pada bab ini.



Gambar 4.19 Mode enkripsi rantai blok sandi (CBC).

Seperti terlihat pada Gambar 4.19, mode CBC menggunakan vektor inisialisasi (IV) sebagai masukan ke fungsi kriptografi. Keamanan mode CBC mengharuskan IV tidak dapat diprediksi (yaitu, penghitung sederhana tidak akan berfungsi), dan oleh karena itu angka yang dihasilkan secara acak sering kali digunakan untuk tujuan ini. IV tidak perlu dirahasiakan dan biasanya dikirimkan langsung ke penerima yang menggunakan IV yang sama untuk proses dekripsi simetris. Pembuatan bilangan acak akan dibahas nanti dalam bab ini.

Untuk enkripsi kunci (atau dikenal sebagai pembungkusan kunci), ECB dengan lebar variabel adalah mode enkripsi yang lebih disukai. Kita telah melihat bahwa untuk ECB, perbedaan satu bit pada blok masukan atau kunci mengakibatkan kerusakan pada semua teks biasa yang terkait (dengan setidaknya 50% variasi pada blok keluaran). Meskipun ECB biasanya digunakan dengan blok 64 atau 128-bit, perluasan ke ukuran blok yang lebih besar (untuk menangani kunci yang lebih besar dari satu blok) dipengaruhi melalui penggunaan ECB dengan lebar variabel, yang pada dasarnya membungkus sebuah blok. ECB dengan lebar tetap yang menunjukkan properti keamanan yang sama dengan ECB dengan lebar tetap. Algoritme pembungkusan kunci NIST AES menggunakan ECB dengan lebar variabel.[6] Sebagai contoh, mari kita pertimbangkan masukan 256-bit yang ingin kita bungkus menggunakan kunci enkripsi kunci 128-bit dan algoritma AES-ECB.

Pendekatan naif akan menerapkan AES-ECB dua kali, satu kali untuk setiap blok masukan secara berurutan. Namun jika diberikan input 256-bit lain yang cocok dengan input sebelumnya, kecuali dengan dua blok yang ditukar, maka hasil 256-bit juga akan cocok dengan hasil sebelumnya kecuali pertukaran blok yang sama. Dengan demikian, enkripsi blok demi blok mengungkapkan pola dalam data masukan. Sebaliknya, spesifikasi bungkus kunci NIST memastikan bahwa masukan 256-bit mengikuti properti yang sama dengan ECB blok tunggal: setiap perubahan bit tunggal pada masukan akan menghasilkan hasil 256-bit yang sama sekali berbeda, seolah-olah ukuran blok AES sebenarnya 256 bit.

Ada banyak mode enkripsi lain yang digunakan saat ini, namun diskusi sebelumnya mencakup mode dasar dan bagaimana mode tersebut melindungi terhadap serangan umum seperti spoofing pesan dan penyusunan ulang. Bagian berikut membahas kelas mode yang menggabungkan kerahasiaan (enkripsi) dan perlindungan integritas.

---

[6] AES Key Wrap Specification. http://csrc.nist.gov/groups/ST/toolkit/documents/kms/AES_key_wrap.pdf

4.6 ENKRIPSI YANG DIAUTENTIKASI

Enkripsi yang diautentikasi bisa lebih efisien secara komputasi daripada menjalankan algoritma enkripsi dan autentikasi independen. Ada banyak algoritma enkripsi yang diautentikasi; di bagian ini kita membahas dua mode yang lebih populer: CCM dan GCM.

4.6.1 CCM

Mode CCM (Penghitung dengan CBC-MAC), yang digunakan dalam standar nirkabel ZigBee, merupakan kombinasi mode penghitung untuk enkripsi dan CBC-MAC untuk otentikasi. CCM juga diamanatkan seperti pada 802.11i (juga dikenal sebagai WPA2), skema otentikasi modern untuk Wi-Fi. Standar 802.11i dibahas di Bab 5. Dalam CCM, mode penghitung digunakan untuk mengubah seluruh teks biasa menjadi teks tersandi. Kemudian CBC-MAC digunakan untuk menghitung intisari seluruh ciphertext. CCM didefinisikan dalam RFC 3610,[7] dan didefinisikan secara kompatibel dalam Publikasi Khusus NIST 800-38C.[8]

4.6.2 Mode Penghitung Galois

Galois Counter Mode (GCM) adalah sandi enkripsi terotentikasi yang direkomendasikan Suite B. AES-GCM direferensikan dalam beberapa dokumen panduan Suite B, termasuk RFC 4869 (IPsec) dan RFC 5430 (TLS). AES-GCM diinginkan karena dapat diimplementasikan secara efisien pada perangkat keras. Seperti CBC, GCM memerlukan vektor inisialisasi. Namun, tidak seperti CBC, GCM IV tidak dapat diprediksi, meskipun harus unik untuk setiap pemanggilan dengan kunci tertentu.

Di sisi negatifnya, implementasi perangkat lunak GCM yang mengoptimalkan kinerja sering kali memerlukan sejumlah besar RAM dalam bentuk tabel yang dihitung secara dinamis. Untuk perangkat tertanam dengan sumber daya terbatas dan tidak memiliki perangkat keras AES-GCM, overhead jejak mungkin menjadi penghalang. Jika tidak, penggunaan AES-GCM untuk enkripsi yang diautentikasi adalah pilihan yang sangat baik, karena kemudahan penggunaannya dan perlakuan yang lebih disukai oleh pemerintah AS. Pada chipset *Intel Architecture* (IA) modern, set instruksi Intel AES-NI dapat digunakan untuk mengimplementasikan AES-GCM yang dioptimalkan tanpa perangkat keras kriptografi tambahan apa pun.[9]

4.7 KRIPTOGRAFI KUNCI PUBLIK

Martin Hellman dan Whitfield Diffie pertama kali menerbitkan sistem kriptografi kunci publik pada tahun 1976. Beberapa orang sezaman dengan Hellman dan Diffie juga mengklaim penemuan kriptografi kunci publik pada tahun 1970an. Meskipun konsep yang diperkenalkan bersifat revolusioner dan menggambarkan serangkaian layanan keamanan yang tidak dapat disediakan oleh kriptografi simetris, kriptografi kunci publik tidak digunakan secara luas hingga awal tahun 2000an ketika era Internet benar-benar meledak. Pertumbuhan Internet dan transaksi e-commerce yang terkait kini memerlukan layanan keamanan digital tambahan, yang dimungkinkan oleh kriptografi kunci publik. Layanan keamanan ini mencakup integritas

[7] Counter with CBC-MAC (CCM). Network Working Group, Request for Comments: 3610 (September 2003).
[8] NIST Special Publication 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality (May 2004).
[9] Vinodh G, et al. Optimized Galois-Counter-Mode Implementation on Intel Architecture Processors (April 2010).

dan otentikasi file multimedia besar, tanda tangan digital, dan non-penyangkalan untuk semua jenis transaksi komersial dan hukum.

Banyak sumber pendidikan menghubungkan penciptaan kriptografi kunci publik untuk memperbaiki kelemahan praktis yang melekat dalam kriptografi kunci simetris: masalah distribusi kuncinya, kadang juga disebut masalah bootstrapping. Agar dua titik akhir dalam jaringan besar dapat berkomunikasi menggunakan kriptografi simetris, operator jaringan harus terlebih dahulu mendistribusikan kunci simetris bersama antara kedua pihak. Distribusi ini dapat dilakukan melalui pertemuan pribadi atau melalui layanan kurir yang dapat dipercaya. Pemeriksaan langsung memberikan jaminan bahwa hanya komunikator resmi yang menggunakan kunci tersebut. Dalam jaringan global, distribusi seperti ini jelas tidak praktis. Dengan kriptografi kunci publik, kunci simetris diganti dengan pasangan kunci asimetris yang hanya salah satu kuncinya, yaitu kunci privat, yang harus dirahasiakan oleh salah satu pihak yang berkomunikasi; pihak lain menggunakan kunci berpasangan, kunci publik. Karena kunci kedua ini tidak perlu dirahasiakan, masalah distribusi seharusnya sudah terpecahkan: satu pihak dapat membuat pasangan kunci dan mengirimkan separuh publiknya melalui jaringan ke pihak lain yang perlu berkomunikasi dengannya. Pemegang kunci privat adalah satu-satunya pihak yang dapat mendekripsi informasi yang dikirim oleh pemegang kunci publik yang dipasangkan.

Sayangnya, dalam sebagian besar penggunaan kriptografi kunci publik yang realistis, masalah distribusi kunci tidak lebih sederhana daripada masalah pembagian kunci simetris karena kebutuhan untuk mendistribusikan dan melakukan instalasi terpercaya dari sertifikat otoritas umum.

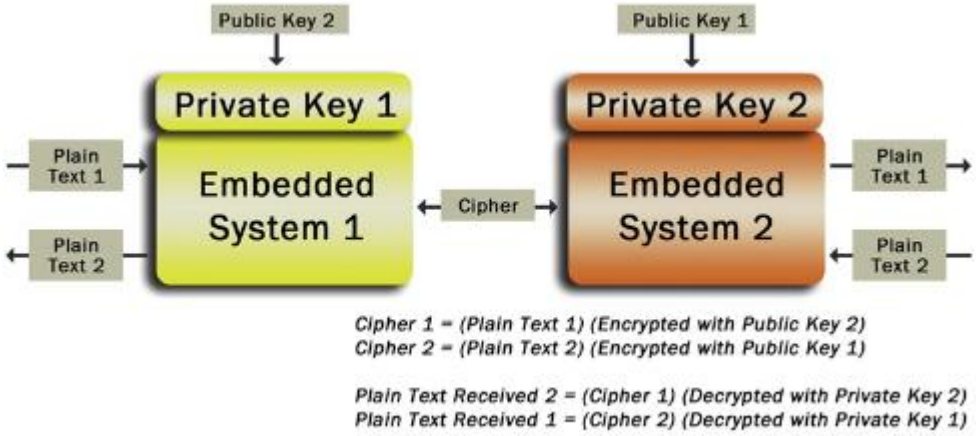
Masalahnya adalah kemampuan untuk mengenkripsi data dengan sukses di satu sisi dan mendekripsinya dengan sukses di sisi lain tidaklah cukup. Setidaknya salah satu titik akhir biasanya memerlukan bukti bahwa suatu pesan datang dari pihak tertentu yang “dikenal baik”. Misalnya, sebelum seorang pembelanja mengirim informasi kartu kredit melalui World Wide Web ke pedagang, ia menginginkan jaminan bahwa ia memang mengirimkan data pribadi ini ke pedagang dan bukan entitas jahat yang menyamar sebagai pedagang. Penjual mengirimkan kunci publiknya untuk digunakan dalam enkripsi, namun bagaimana pembeli ini mengetahui bahwa pedagang (dan hanya pedagang) yang benar-benar memegang bagian pribadi dari kunci publik tersebut? Seorang perantara yang menguping jaringan dapat mengganti kunci publik pedagang dengan kunci publiknya sendiri, dan kemudian informasi kartu kredit dengan mudah dicuri oleh penyusup. Jadi, di sebagian besar aplikasi, kunci publik harus diautentikasi sebelum digunakan.

Mekanisme umum untuk autentikasi kunci publik, yang akan kami jelaskan nanti, memerlukan pengesahan kunci publik oleh otoritas yang dipercaya oleh kedua titik akhir. Dalam sistem seperti itu, pembeli harus memiliki salinan kunci publik otoritas tepercaya yang digunakan untuk memvalidasi kunci publik pedagang. Dengan kata lain, masalah pendistribusian kunci simetris secara fisik telah digantikan dengan masalah pendistribusian kunci publik otoritas secara fisik! Masalah bootstrap masih tetap signifikan. Namun demikian, kriptografi kunci publik, seperti yang akan segera kita bahas, sangat penting dalam banyak skema perlindungan karena sifatnya yang asimetris.

Sekali lagi, dalam kriptografi kunci publik, seperti ditunjukkan pada Gambar 4.20, pengirim dan penerima pesan memiliki dua kunci yang berbeda, namun berpasangan: kunci enkripsi, yang dipublikasikan; dan kunci dekripsi, yang hanya diketahui oleh penerima. Kami menyebut kunci enkripsi sebagai kunci publik (PK); dan kunci berpasangannya, kunci penerima pribadi (PRK). Setiap pemancar yang ingin mengirim pesan ke penerima tertentu mengenkripsi pesan tersebut dengan PK dan mengirimkan ciphertext melalui saluran yang tidak aman; hanya penerima yang dituju, yang memiliki pengetahuan eksklusif tentang PRK, yang dapat menguraikan pesan tersebut.

Matematika kriptografi kunci publik sedemikian rupa sehingga pengetahuan musuh tentang PK atau PRK tidak cukup untuk menentukan pasangan kunci lainnya. Selain itu, kriptografi kunci publik memungkinkan penggunaan kunci dekripsi pribadi untuk menyandikan pesan dan kunci enkripsi publik untuk menguraikannya. Properti ini sangat berguna untuk otentikasi asal: siapa pun yang memegang PK dapat mendekripsi (memverifikasi) pesan tersebut, dan dekripsi yang berhasil membuktikan bahwa pesan tersebut berasal dari pemegang PRK pribadi yang unik.

Matematika kriptografi kunci publik memerlukan pengetahuan tentang teori bilangan dasar, aritmatika modular, dan aritmatika kurva elips. Meskipun kami tidak membahas matematika ini secara rinci dalam buku ini, sangat penting untuk memahami konsep-konsep kunci publik dan bagaimana konsep-konsep tersebut diperoleh dan diimplementasikan. Konsep-konsep ini dijelaskan pada bagian berikut.



Gambar 4.20 Enkripsi/dekripsi kunci publik.

Kriptografi kunci publik didasarkan pada fungsi matematika khusus yang relatif mudah dihitung tetapi sangat sulit untuk dibalik. Oleh karena itu, fungsi khusus ini disebut fungsi satu arah. Beberapa contoh fungsi matematika khusus tersebut adalah

- **Memfaktorkan hasil kali dua bilangan prima besar**: Sangat mudah untuk menghitung hasil kali dua bilangan prima, namun tidak ada metode yang diketahui untuk memfaktorkan bilangan prima selain dengan mencoba semua kemungkinan kombinasi untuk mendapatkan bilangan prima itu sendiri; algoritma RSA (dinamai Rivest-Shamir-Adleman) menggunakan metode ini. Keuntungan utama RSA adalah kesederhanaannya, yang mendorong implementasi yang baik.

- **Algoritme diskrit**: Mudah untuk menaikkan suatu bilangan menjadi eksponen besar namun sulit menghitung logaritma diskrit; algoritma Diffie-Hellman menggunakan metode ini dan juga elegan dalam kesederhanaannya.
- **Kurva elips**: Menghitung perkalian titik skalar sangatlah mudah tetapi sangat sulit mendapatkan inversnya; kriptografi kurva elips (ECC) menggunakan metode ini.

Untuk fungsi matematika khusus ini, inversi hanya mungkin terjadi jika fungsi tersebut mengandung fungsi pintu jebakan, istilah yang diciptakan oleh Diffie dan Hellman. Pintu jebakan bertindak sebagai kunci untuk membuka transformasi satu arah yang tidak dapat dibalikkan. Pintu jebakan ini harus dibangun dan diformulasikan dengan baik ketika pasangan kunci publik (PK dan PRK) dikembangkan. Misalnya, mudah untuk mengalikan dua bilangan prima yang sangat besar (masing-masing terdiri dari 500 hingga 2.000 atau lebih digit biner), namun memfaktorkan hasil kali kedua bilangan prima ini secara komputasi tidak mungkin dilakukan tanpa pintu jebakan.

4.7.1 RSA

Mari kita pertimbangkan P dan Q, dua bilangan prima 500 digit, dan N ¼ P * Q. N disebut modulus dalam algoritma kunci publik RSA. Mengingat N dan rahasia P dan Q yang diungkapkan secara publik, diperlukan waktu sekitar lima miliar tahun untuk memfaktorkan P dan Q pada komputer yang dapat menjalankan dua miliar instruksi per detik.

Namun, jika metode yang lebih baik untuk memfaktorkan modulus RSA ditemukan, maka waktu tersebut dapat dikurangi secara drastis. Tentu saja, peningkatan kecepatan komputer secara konsisten telah meningkatkan ukuran kunci yang dapat dipulihkan dengan faktorisasi brute force. Misalnya, saat tulisan ini dibuat, RSA-768 (RSA memerlukan pemfaktoran modulus 768-bit untuk memecahkannya) tidak lagi aman.[10] Meskipun memfaktorkan modulus 768-bit mungkin tampak mengesankan di tahun 2010-an, kemungkinan besar Anda, para pembaca, adalah tertawa kecil melihat prestasi ini jika Anda membaca di tahun 2020-an. Laju peningkatan faktorisasi ukuran modulus antara tahun 1991 dan 2010 ditunjukkan pada Gambar 4.21. Sebagian besar sistem yang digunakan pada tahun 2012 menggunakan 1.024 digit atau lebih besar kunci RSA. Namun, para kriptografer percaya bahwa modulus 1.024-bit sudah mendekati (atau pada) akhir masa pakainya sebagai pilihan yang aman. NIST merekomendasikan penggunaan modul 2.048-bit dan mengantisipasi pilihan ini akan tetap aman setidaknya hingga tahun 2030.[11]

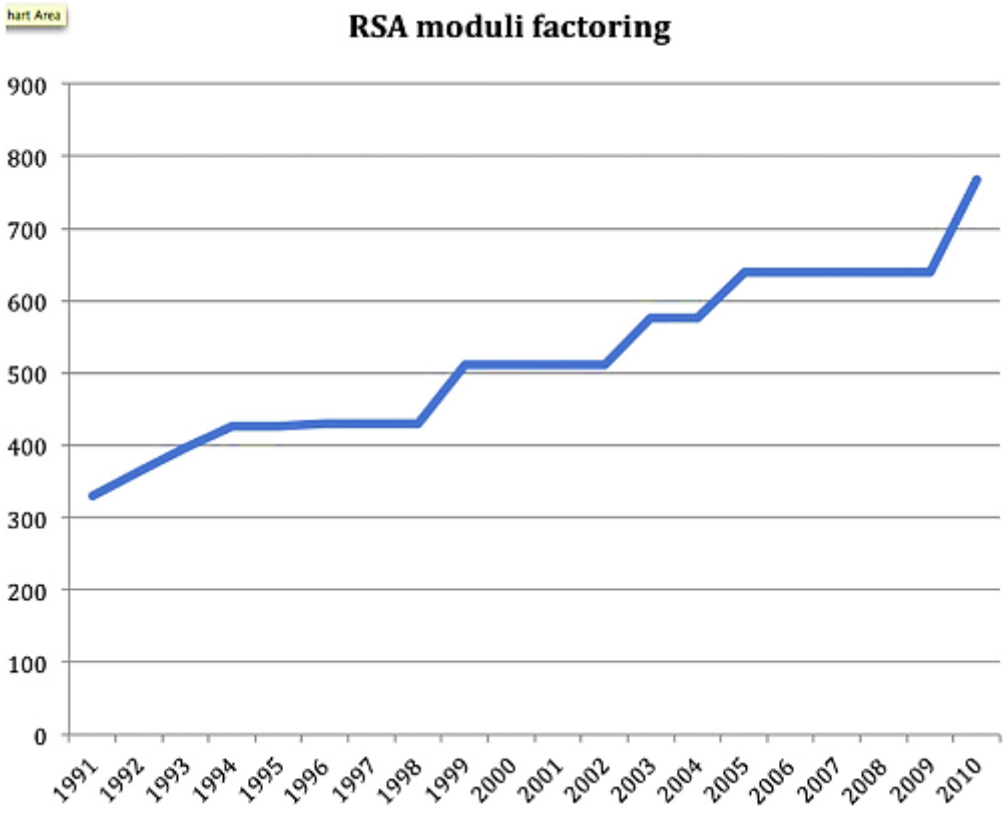
4.7.2 Kekuatan Kunci Setara

Komunitas kriptografi secara tradisional memperkirakan kekuatan kriptografi yang sesuai dengan panjang kunci kunci publik dan algoritma lainnya sebagai ukuran kekuatan bit dalam algoritma kunci simetris, seperti AES.

---

[10] Kleinjung, Thorsten, et al. Factorization of a 768-Bit RSA Modulus, version 1.4 (February 18, 2010), http:// eprint.iacr.org/2010/006.pdf

[11] NIST Special Publication 800-57. Recommendation for Key ManagementdPart 1: General (Revised) (March 2007).



Gambar 4.21 Faktorisasi modulus RSA dari waktu ke waktu.

Misalnya, kunci RSA 3.072-bit diperkirakan memiliki kekuatan setara dengan AES 128-bit.

Publikasi khusus NIST 800-57 memberikan perkiraan panjang kunci untuk algoritma tertentu berdasarkan lamanya waktu perlindungan data. Kuanta ini mengantisipasi bahwa daya komputasi dan/atau peningkatan teknik cracking akan membuat panjang kunci generasi sebelumnya tidak memadai. Kekuatan kunci setara NIST untuk kriptosistem kunci simetris dan kunci publik umum dirangkum dalam Tabel 4.5.

Tabel 4.5: Kekuatan kunci setara NIST

Sedikit Keamanan	Simetris	RSA/DH	ECC
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

4.7.3 Konstruksi Pintu Trap

Kami mengilustrasikan proses konstruksi pintu jebakan menggunakan algoritma RSA. Dua sifat aritmatika modular yang penting adalah sebagai berikut:

```
Given primes P, Q and N = P*Q and W =(P-1)*(Q-1):
X^Y (mod N) = X^(Y(mod W))
((X^Y (mod N)) ^Z)(mod N)=X^(Y*Z)(mod N)
```

Agar enkripsi dan dekripsi dapat berlangsung, pintu jebakan dibuat dengan memilih bilangan bulat acak, E, antara 3 dan W yang tidak memiliki faktor persekutuan dengan W. Sekarang kita mencari bilangan bulat, D, yang merupakan kebalikan dari E modulo W. Dalam notasi modulo, invers ini dinyatakan sebagai berikut:

$$(\boldsymbol{D^{*}E})(\boldsymbol{mod\ W}) = \mathbf{1}$$

Bilangan bulat E dan N bersifat publik; W dibuang setelah E dan D dihitung; semua kuantitas lainnya dirahasiakan.

Dengan adanya parameter ini, operasi enkripsi RSA didefinisikan sebagai berikut:

$$\boldsymbol{CTT = PTT^{\wedge}E\ (mod\ N)}$$

Operasi dekripsi adalah sebagai berikut:

$$\boldsymbol{PTR = CTR^{\wedge}D\ (mod\ N)}\ dimana\ \boldsymbol{CTR = CTT}$$

Aritmatika modular dan pilihan E dan D membuktikan bahwa PTR==PTT sebagai berikut:

PTR = CTR^D(mod N)
= (PTT^E(mod N)) ^D)(mod N)
= PTT^(E*D)(mod N)
= PTT^((E*D)(mod W))
= PTT^1
= PTT

Mengingat E yang dipilih, D adalah pintu jebakan yang memungkinkan sandi yang didekripsi cocok dengan teks biasa. Contoh berikut mendemonstrasikan matematika dengan bilangan sederhana:

Let P = 3,Q =11,N = P*Q =33,W =(P-1)*(Q-1)=2*10=20

Pilih secara acak suatu bilangan E yang relatif prima (tidak ada faktor persekutuan) terhadap W = 20. Faktor W adalah 1, 2, 4, 5, 10, dan 20. Oleh karena itu, pilihan untuk E adalah 3, 7, 9, 11, 13 , 17, dan 19. Kita pilih E = 7 secara acak. Dalam aritmatika modulo, kita menghitung D sehingga (E*D) (mod W) = 1. Jadi, D = 3. Misalkan PTT=2.

Encryption : PTT^E (mod N) = 2^7 (mod 33)= 128 (mod 33) = 29 (CTT)
Decryption : CTR^D (mod N) = 29^3 (mod 33)=23,389 (mod 33) = 2 (PTT)

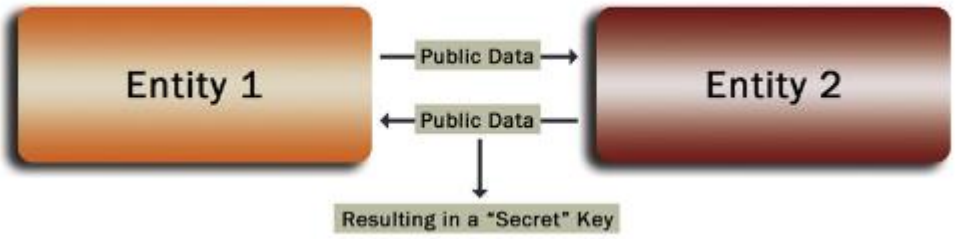
Karena dalam praktiknya, bilangan bulat yang sangat besar, yang sering disebut bignum oleh praktisi kriptografi, digunakan untuk bagian publik dan privat dari sebuah kunci,

kriptografi kunci publik merupakan proses komputasi yang intensif. Misalnya, untuk RSA-2048, modulus N yang digunakan selama enkripsi adalah bignum 2.048-bit. Secara umum, algoritma kunci publik setidaknya 1.000 kali lebih lambat dibandingkan algoritma simetris yang menggunakan kunci dengan kekuatan yang relatif sama.

Karena dalam praktiknya, bilangan bulat yang sangat besar, yang sering disebut bignum oleh praktisi kriptografi, digunakan untuk bagian publik dan privat dari sebuah kunci, kriptografi kunci publik merupakan proses komputasi yang intensif. Misalnya, untuk RSA-2048, modulus N yang digunakan selama enkripsi adalah bignum 2.048-bit. Secara umum, algoritma kunci publik setidaknya 1.000 kali lebih lambat dibandingkan algoritma simetris yang menggunakan kunci dengan kekuatan yang relatif sama. Kunci sementara ini disebut kunci sesi dan dibuat melalui pertukaran kunci publik, atau perjanjian kunci, antara dua pihak. Para pihak kemudian menggunakan kunci sesi rahasia untuk melindungi kerahasiaan lalu lintas yang dikirim selama sesi komunikasi.

4.8 PERJANJIAN UTAMA

Seperti yang ditunjukkan pada Gambar 4.22, kriptografi yang diterapkan dengan benar menyediakan skema untuk mengembangkan kunci rahasia yang dibagikan antara dua entitas yang bekerja sama.



Gambar 4.22 Pembuatan kunci rahasia bersama.

Awalnya, kedua pihak, yang kami sebut Alice dan Bob, tidak berbagi rahasia kriptografi yang sama. Baik Alice dan Bob menghasilkan pasangan kunci publik sementara dan mengirimkan bagian publik satu sama lain. Baik Alice dan Bob, menggunakan pasangan kunci mereka sendiri dan separuh kunci publik yang baru diterima, mengembangkan dua rahasia bersama yang berbeda serta gabungan rahasia umum. Algoritma yang digunakan untuk mengembangkan rahasia bersama disebut algoritma perjanjian kunci. Rahasia ini digunakan untuk mendapatkan kunci sesi satu kali, yang dirahasiakan hanya untuk Alice dan Bob. Kunci sesi digunakan sebagai kunci simetris untuk enkripsi keystream atau buku kode.

Algoritma Diffie-Hellman adalah algoritma perjanjian kunci yang paling umum. Untuk mengilustrasikan konsep perjanjian kunci menggunakan Diffie-Hellman, asumsikan Alice membuat pasangan kunci publik di mana E1 adalah kunci publik, dan D1 adalah kunci privat. Demikian pula Bob menciptakan E2 dan D2. Parameter Public Diffie-Hellman G dan P diketahui sebelumnya oleh Bob dan Alice (misalnya, protokol perjanjian kunci keseluruhan mungkin mencakup transmisi G dan P antara Alice dan Bob). E1 dan E2 dihitung sebagai berikut:

Alice : E1 = G^D1 (mod P); Alice sends E1 to Bob
Bob : E2 = G^D2 (mod P); Bob Sends E2 to Alice

Baik Alice dan Bob kemudian menghitung rahasia bersama sebagai berikut:

ALICE: S1 = E2^D1 (mod P)
= ((G^D2 9mod P)) ^D1)(mod P)
= G^(D2*D1))(mod P)
Bob: S2 = E1^D1(mod P)
= ((G^D2(mod P)) ^D1) (mod P)
= (G^(D1*D2))(mod P)
Thus:S1 = S2

Rahasia bersama yang dihasilkan S==S1==S2 dihitung secara independen dalam perangkat komputasi masing-masing Alice dan Bob dan dirahasiakan. Karena S bergantung pada bagian privat dari pasangan kunci ephemeral Alice dan Bob, tidak ada entitas lain yang mampu menghitung rahasia bersama yang sama. Hasil S1 dan S2 dipersingkat dengan cara tertentu dan digunakan sebagai kunci sesi rahasia untuk kriptografi simetris.

Diffie-Hellman adalah algoritma yang penting karena memungkinkan kerahasiaan maju yang sempurna dicapai oleh protokol keamanan jaringan: kunci sesi tidak dapat dikompromikan bahkan jika kunci sesi lain atau salah satu kunci pribadi jangka panjang yang digunakan untuk otentikasi asal dikompromikan. Karena Diffie-Hellman sendiri tidak menyediakan otentikasi asal apa pun, ini dianggap sebagai algoritma perjanjian anonim.

Fungsionalitas tanda tangan digital harus ditambahkan ke protokol yang menggunakan Diffie-Hellman agar dapat menggunakan perjanjian kuncinya untuk komunikasi peer-to-peer yang aman.

4.8.1 Serangan Man-in-the-Middle terhadap Diffie-Hellman

Skema perjanjian kunci klasik Diffie-Hellman mempunyai kerentanan serangan man-in-the-middle. Salah satu serangan tersebut ditunjukkan pada Gambar 4.23.

Gambar 4.23 Serangan man-in-the-middle terhadap Diffie-Hellman.

Karena kunci publik sementara Alice dan Bob dipublikasikan, musuh, Mallory, yang merupakan penyadap aktif, dapat mencegat mereka dan memperkenalkan kunci publiknya sendiri untuk didistribusikan kepada Alice dan Bob. Alice menghitung kunci sesi bersama, AMK, yang dapat dihitung oleh Mallory, dan Bob menghitung kunci bersama yang berbeda,

BMK, yang juga dapat dihitung oleh Mallory. Lalu lintas yang berasal dari Alice didekripsi oleh Mallory menggunakan AMK dan dienkripsi ulang dengan BMK dan dikirim ke Bob.

Demikian pula, lalu lintas dari Bob didekripsi menggunakan BMK dan dienkripsi ulang menggunakan AMK. Hasilnya, Mallory sekarang dapat membaca semua lalu lintas antara Alice dan Bob tanpa sepengetahuan mereka tentang penipuan tersebut. Untuk mencegah serangan ini, kita harus memastikan bahwa kunci publik sementara Diffie-Hellman benar-benar dihasilkan oleh Alice dan Bob dan bukan oleh musuh. Kami mengatakan bahwa kunci publik harus diautentikasi.

4.9 OTENTIKASI KUNCI PUBLIK

Ada banyak metode modern untuk otentikasi kunci publik, termasuk;

- **Menerbitkan semua kunci publik ke server kunci tepercaya**: Basis data server kunci pada dasarnya adalah daftar putih klien yang telah diautentikasi sebelumnya. Semua klien dapat berkomunikasi dengan server kunci pada saluran yang aman (terenkripsi dan diautentikasi) menggunakan kunci publik yang telah ditukarkan sebelumnya. Jika Alice ingin mengirim pesan terautentikasi kepada Bob, maka Alice menandatangani pesan tersebut dan mengirimkannya ke server kunci. Bob meminta server kunci untuk memvalidasi pesan tersebut. Server kunci melakukan otentikasi menggunakan kunci publik Alice dan kemudian menginformasikan Bob melalui saluran aman antara Bob dan server. Dengan pendekatan ini, setiap titik akhir harus memiliki kunci publik yang telah ditentukan sebelumnya dan diautentikasi hanya untuk server kunci, bukan untuk semua titik akhir lainnya. Kelemahannya adalah semua pesan harus melewati server kunci tepercaya. Kelemahan lainnya adalah risiko terhadap ketersediaan: jika server mati, tidak ada pesan yang dapat diautentikasi.
- **Mengirimkan kunci publik menggunakan saluran aman yang telah ditentukan sebelumnya dan diautentikasi**: Alih-alih memasukkan daftar putih, server kunci pada awalnya tidak memiliki kunci publik. Setiap klien harus mendaftarkan kunci publiknya secara dinamis menggunakan koneksi aman yang dimilikinya dengan server kunci.
- **Menanamkan kunci publik dalam amplop digital yang ditandatangani oleh pihak ketiga yang saling percaya**: Kunci publik digabungkan dengan data identitas/atribut pengguna dan hasilnya ditandatangani secara kriptografis oleh pihak ketiga menggunakan kunci tanda tangan rahasia pihak ketiga; amplop bertanda tangan ini dikenal sebagai sertifikat kunci publik. Sertifikat mengikat identitas pengguna ini ke kunci publiknya. Oleh karena itu, verifikasi tanda tangan menggunakan kunci publik ini memberikan jaminan bahwa data yang ditandatangani berasal dari identitas terkait. Setiap klien harus sudah menginstal kunci publik, yang disebut sertifikat akar, dari otoritas tanda tangan pihak ketiga tepercaya, yang disebut otoritas sertifikasi (CA).
- **Memiliki entitas yang saling percaya (web of trust) yang menjamin kunci publik**: Alih-alih menggunakan otoritas sertifikat tunggal untuk sejumlah besar klien, otorisasi dilakukan secara terdesentralisasi. Setiap titik akhir dapat menjamin kunci publik dengan menandatanganinya, dan beberapa titik akhir dapat menjamin kunci yang sama. Keputusan kebijakan dibuat secara lokal mengenai apakah akan mempercayai

kunci publik, misalnya berdasarkan jumlah tanda tangan. Pretty Good Privacy (PGP) adalah contoh sistem kriptografi yang menggunakan konsep web of trust.

Pendekatan paling umum yang digunakan untuk distribusi kunci publik tepercaya adalah dengan menggunakan sertifikat yang ditandatangani oleh otoritas sertifikasi (CA) tepercaya; penggunaan CA biasanya memerlukan infrastruktur kunci publik (PKI) yang digunakan untuk mengelola siklus hidup sertifikat di semua titik akhir yang berbagi CA.

Badan Keamanan Nasional Amerika Serikat (NSA) mengoperasikan PKI untuk mengamankan informasi rahasia. NSA menghasilkan pasangan kunci publik untuk semua entitas dan menerbitkan kunci privat dan sertifikat kunci publik terkait kepada pengguna terdaftar melalui proses distribusi yang aman. Pemerintahan berdaulat lainnya menggunakan pembangkitan kunci terpusat yang serupa.

Otoritas sertifikat komersial seperti VeriSign menggunakan pendekatan berbeda terhadap PKI: biasanya tanda tangan publik CA komersial tertanam di browser atau penyimpanan lokal lainnya pada perangkat titik akhir. Titik akhir menghasilkan pasangan kunci publiknya sendiri dan dimiliki CA menandatangani sertifikat yang dihasilkan titik akhir yang berisi identitas dan kunci publik titik akhir yang terverifikasi CA. Proses pendaftaran/penandatanganan titik akhir dapat menggunakan email atau metode komunikasi lainnya antara titik akhir dan CA. CA tidak pernah melihat kunci rahasia titik akhir.

4.9.1 Jenis Sertifikat

X.509 adalah format standar yang paling umum untuk sertifikat kunci publik. RFC 5280 mendefinisikan standar terbaru, X.509v3.[12] Untuk sistem tertanam yang harus berkomunikasi secara aman dengan sistem lain yang tidak dikontrol langsung oleh vendor tertanam, sertifikat X.509 dan CA pihak ketiga akan umum digunakan. Panduan NSA Suite B untuk penggunaan sertifikat kunci publik didokumentasikan dalam RFC 5759.[13]

NSA Suite B menetapkan penggunaan sertifikat X.509 v3 menggunakan kriptografi kurva elips (ECC) berbasis tanda tangan digital.

Pendekatan Sertifikat Kustom untuk Sistem Tertanam

Untuk sistem tertanam yang rekan-rekannya dapat dikontrol secara lebih langsung (tidak perlu beroperasi dengan titik akhir yang sewenang-wenang), format sertifikat lain dapat digunakan. Secara khusus, beberapa sistem tertanam dengan memori terbatas mungkin menganggap persyaratan penyimpanan untuk kumpulan minimum sertifikat X.509, yang masing-masing mungkin berukuran sekitar 1 KB, menjadi penghalang. Beberapa sistem tertanam dengan sumber daya terbatas telah mengadopsi sertifikat implisit, di mana kunci publik direkonstruksi dari sertifikat tersebut. Sertifikat implisit berukuran sama dengan kunci publik. Dalam kasus kriptografi kurva elips, hal ini menghasilkan sertifikat yang jauh lebih kecil dibandingkan sertifikat X.509 tradisional.

Sistem sertifikat implisit kurva elips yang terkenal adalah Elliptic Curve Qu-Vanstone (ECQV).[14] Pengguna sertifikat implisit menghitung kunci publik dari identitas terkait kunci publik dan kunci publik otoritas sertifikat (yang biasanya sudah diinstal sebelumnya) dan

[12] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, Internet Engineering Task Force, Request for Comments: 5280 (May 2008).

[13] Suite B Certificate and Certificate Revocation List (CRL) Profile, Internet Engineering Task Force, Request for Comments: 5759 January 2010.

[14] SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV), Version 0.97, Certicom Research (March 2011).

kemudian menggunakannya dalam beberapa operasi kunci publik (misalnya, Elliptic Curve Digital Signature Algorithm, atau ECDSA). Operasi berhasil hanya jika kunci publik valid; oleh karena itu, kami menyebut pendekatan ini tersertifikasi secara implisit.

Pada saat tulisan ini dibuat, divisi Certicom di RIM memiliki paten ECQV yang belum habis masa berlakunya, sehingga penggunaannya menjadi penghalang bagi sebagian besar pengembang sistem tertanam.

Dalam beberapa kasus, perancang sistem tertanam dapat memperoleh manfaat dari vendor kriptografi yang dapat menyediakan sertifikat khusus yang sumber dayanya dioptimalkan. Misalnya, sertifikat yang dioptimalkan mungkin menyertakan format informasi identifikasi biner, kunci publik, dan tanda tangan yang ditentukan secara statis. Format seperti itu juga akan jauh lebih kecil dibandingkan sertifikat X.509 pada umumnya. Perancang sistem tertanam harus berkonsultasi dengan pemasok perangkat lunak/kriptografi sistem mereka untuk memahami pilihan yang tersedia.

4.10 KRIPTOGRAFI KURVA ELIPS

Terlepas dari keamanan dan keberadaan RSA di mana-mana, NSA Suite B menetapkan kriptografi kurva elips (ECC) untuk fungsionalitas kunci publik. Alasannya adalah efisiensi. Menurut NIST SP 800-57, kunci ECC 224-bit yang digunakan untuk tanda tangan digital memiliki kekuatan yang setara dengan RSA-2048.

Meskipun perbedaan kinerja antara algoritma RSA dan kurva eliptik dengan ukuran kunci kontemporer masing-masing tidaklah dramatis, ECC memberikan efisiensi kekuatan keamanan per bit yang lebih baik dan oleh karena itu dianggap oleh NSA dan sebagian besar kriptografer sebagai pilihan jangka panjang yang unggul.

Misalnya, setelah tahun 2030, rekomendasi NIST untuk panjang kunci ECC adalah 256 bit, hanya 14% lebih tinggi dari panduan sebelum tahun 2030. Namun rekomendasi RSA (3072) 50% lebih tinggi dibandingkan panduan sebelum tahun 2030.

ECC menggunakan matematika yang lebih rumit daripada RSA, sehingga menghasilkan implementasi yang lebih kompleks, yang merupakan alasan lain mengapa beberapa orang masih memilih RSA. Alasan lain kurangnya adopsi ECC secara cepat adalah ketakutan akan paten ECC RIM (yang diperoleh melalui akuisisi Certicom).

Meskipun beberapa opsi implementasi untuk ECC dilindungi oleh paten, semua vendor perangkat kriptografi besar (serta pengembang open source OpenSSL) telah menciptakan implementasi ECC yang mengikuti spesifikasi publik dan diyakini bebas dari risiko paten.

Namun alasan lain kurangnya adopsi ECC adalah kelembaman warisan: banyak sistem komputer yang diterapkan dan otoritas sertifikasi (bahkan di dalam Departemen Pertahanan AS (DoD), yang kebijakan kriptografinya dikendalikan oleh NSA) diatur untuk menggunakan kunci RSA dan RSA. sertifikat digital berbasis.

Untuk sistem tertanam yang beroperasi pada jaringan menggunakan PKI berbasis RSA lama, ECC tidak dapat digunakan secara praktis sampai infrastruktur ini ditingkatkan untuk menggunakan ECC. Konversi dari RSA ke ECC di Departemen Pertahanan AS sedang berlangsung.

Tanda Tangan Digital Kurva Elips

Algoritma tanda tangan digital untuk ECC disebut Elliptic Curve Digital Signature Algorithm (ECDSA) dan merupakan satu-satunya algoritma tanda tangan digital di NSA Suite B. Saat mengambil keputusan mengenai apakah akan menggunakan ECC atau RSA, perancang sistem tertanam perlu mempertimbangkan lamanya penerapan di lapangan dan apakah persetujuan NSA mungkin diperlukan (dalam hal ini Suite B adalah pilihan yang aman).

Perjanjian Kunci Anonim Kurva Elliptik

Seperti RSA, Diffie-Hellman tidak ada di Suite B. Sebaliknya, Suite B menentukan Elliptic Curve Diffie-Hellman (ECDH), yang menghitung pasangan kunci publik/pribadi sementara berdasarkan kurva elips, bukan eksponensial. Seperti RSA dan ECDSA, preferensi Suite B untuk ECDH terletak pada efisiensi bit relatifnya. Menariknya, algoritma yang berbeda, ECMQV, dulunya ada di Suite B, bukan ECDH. Meskipun ECMQV dikenal lebih efisien kinerjanya dibandingkan ECDH, NSA tidak mampu menyelesaikan beban paten ECMQV secara memuaskan. Meskipun tidak ada persyaratan teknis untuk menggunakannya bersama-sama, secara umum, jika desain tertanam memilih RSA dan memerlukan persetujuan kunci, DH biasanya digunakan. Dan jika suatu desain menggunakan ECDSA, maka ECDH yang akan digunakan.

Untuk semua algoritma kurva elips, panduan NSA Suite B menetapkan kurva spesifik yang juga menyiratkan panjang kunci. Untuk informasi sangat rahasia, diperlukan kunci 384-bit. Untuk rahasia dan di bawahnya, kunci 256-bit atau 384-bit dapat digunakan. Untuk memenuhi rekomendasi NIST, ECDSA dan ECDH memerlukan panjang kunci minimal 224 bit hingga tahun 2030 dan 256 bit setelahnya (hingga revisi panduan berikutnya).

4.11 HASH KRIPTOGRAFI

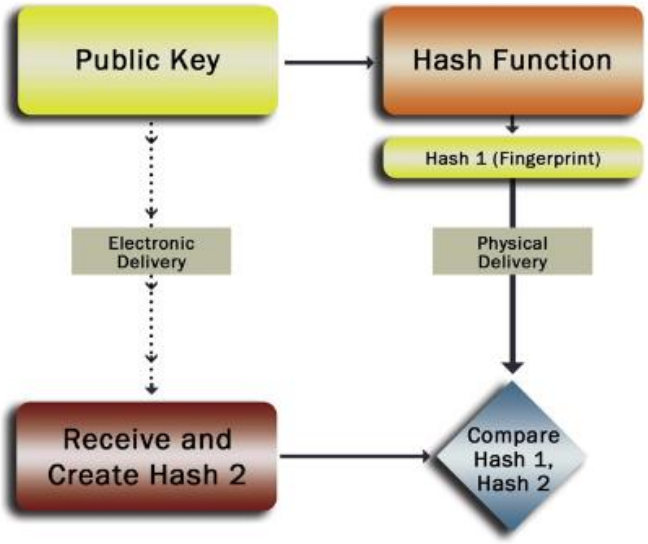
Kunci publik berukuran sangat besar (biasanya ribuan bit), dan distribusi fisik yang aman serta instalasi manualnya tidak praktis. Praktik yang umum dilakukan adalah mendistribusikan kunci publik secara elektronik tetapi secara fisik mendistribusikan sidik jari kunci publik tersebut. Sidik jari ini disebut hash, seperti yang ditunjukkan pada Gambar 4.24:

Hash digunakan untuk memverifikasi keaslian dan integritas kunci publik.

Algoritme hash mengubah aliran bit yang berubah-ubah menjadi pesan dengan panjang tetap, yang disebut nilai hash. Hash kriptografi harus berupa fungsi satu arah yang sebenarnya; dengan kata lain, mengingat nilai hash h yang berubah-ubah, maka secara komputasi tidak mungkin untuk menentukan pesan apa pun yang nilai hashnya cocok dengan h. Hash kriptografi juga harus tahan terhadap benturan; dengan kata lain, mengingat sebuah pesan, m, dan nilai hashnya, h, maka secara komputasi tidak mungkin untuk menemukan pesan lain, selain m, yang juga memiliki nilai hash h. Properti yang terkait dengan resistensi tabrakan adalah keacakan; fungsi hash harus bertindak sebagai fungsi acak namun tetap efisien untuk dihitung. Properti hash kriptografi ini memastikan bahwa penyerang tidak dapat mengubah hash tanpa terdeteksi oleh siapa pun yang memiliki pesan asli dan juga memastikan bahwa penyerang tidak dapat memperoleh informasi apa pun tentang pesan asli yang hanya diberi hash.

Hash kriptografi sendiri paling sering digunakan untuk perlindungan integritas data dan deteksi kerusakan atau kesalahan. Misalnya, pembuat produk perangkat lunak yang kritis

terhadap keamanan harus membuat hash kriptografi dari gambar perangkat lunak yang membentuk produk dan kemudian mendistribusikan hash tersebut dengan aman kepada pengguna akhir perangkat lunak. Pengguna akhir dapat menghasilkan hash pada pengiriman perangkat lunak dan membandingkan hash tersebut dengan nilai yang diketahui baik untuk memastikan bahwa perangkat lunak belum dimodifikasi sebelum diterima. Verifikasi hash yang sama dapat dilakukan pada saat boot pada produk yang dimasukkan untuk memastikan bahwa isi memori yang digunakan untuk menyimpan perangkat lunak tidak rusak.



Gambar 4.24 Menggunakan hash untuk mengurangi ukuran kunci publik yang dikirimkan.

Algoritma Hash Aman

Algoritma Hash Aman (SHA)-1 telah menjadi algoritma hash yang populer secara historis yang menghasilkan intisari 160-bit. SHA-1 telah mengalami penemuan kelemahan tabrakan. Pada bulan Februari 2005, para peneliti melaporkan penyempitan ruang pencarian tabrakan menjadi 2^69 komputasi, jauh di bawah kekuatan brute force sebesar 2^80 komputasi.[15] Pada bulan Agustus di tahun yang sama, peningkatan pada serangan tersebut mengurangi kompleksitas waktu menjadi 2^ 63.[16] Sejak saat itu, beberapa peneliti mengklaim dapat mengurangi kompleksitas lebih lanjut, sekitar 2^57 komputasi. Karena sifat kriptanalitiknya yang bermanfaat, SHA-1 harus diturunkan hanya ke tanggung jawab dasar pemeriksaan integritas data.

Pengganti SHA-1 saat ini adalah SHA-2, sebuah keluarga algoritma hash dengan pilihan panjang bit hasil yang digunakan untuk lebih memenuhi syarat nama algoritma: SHA-224, SHA-256, SHA-384, dan SHA-512. Perhatikan bahwa SHA-1 menghasilkan hasil 160-bit. NIST sedang dalam tahap akhir pemilihan SHA-3 melalui kompetisi algoritma publik. Pemenangnya harus diumumkan pada tahun 2012. Kelompok algoritma SHA-1 dan SHA-2 ditentukan dalam FIPS 180-2.[17]

[15] Xiaoyun W, Lisa Y, Hongbo Y. Finding Collisions in the Full SHA-1 (February 2005). Proceedings of Crypto Core.
[16] Xiaoyun W, Andrew CY, Frances Y. Cryptanalysis on SHA-1, Presented by Adi Shamir at the Rump Session of CRYPTO 2005.
[17] Secure Hash Standard. Federal Information Processing Publication 180-2 (August 1, 2002).

NSA Suite B menetapkan penggunaan SHA-384 untuk informasi sangat rahasia dan penggunaan SHA-256 atau SHA-384 untuk informasi rahasia dan di bawahnya. Sebaliknya, panduan NIST menyatakan bahwa SHA-224 cukup untuk produk keperluan umum hingga tahun 2030.

MMO

Meskipun varian SHA adalah satu-satunya algoritme hash yang disetujui NIST dan FIPS, varian tersebut bukanlah satu-satunya algoritme yang populer. Misalnya, algoritma Matyas-Meyer-Oseas, yang biasanya digunakan dengan AES (AES-MMO), adalah algoritma menarik yang digunakan di ZigBee. Hal yang menyenangkan tentang MMO adalah ia dijalankan dengan cepat dan dengan overhead memori yang dapat diabaikan ketika prosesor yang tertanam sudah memiliki mesin offload untuk algoritma enkripsi yang mendasarinya. Contoh implementasi kode C berikut menunjukkan kesederhanaan MMO:

```
// This is the Matyas-Meyer-Oseas hash algorithm applied to the AES block
// cipher. The algorithm is standardized in ISO/IEC 10118-2.
// This implementation uses AES-ECB-128 in order to match the keysize
// with the AES block size (avoiding a transformation of intermediate blocks
// to a different key size for AES). The initialization vector must also
// be of length equal to the AES block size. Finally, this implementation
// requires that the input plaintext be of length equal to a multiple of
// the AES block size (no padding is performed).
// The output tag is the contents of the final output block, of size equal to
// the AES block size.
void AesMmo(const uint32_t *iv, const uint32_t *pt, uint32_t len,
uint32_t *tag)
{
  uint32_t i      = 0
  blk *key  = =(blk *)tag;
  const blk *ptb  =(blk *)pt;
  assert((len % AES_BLOCK_SIZE) ==0);
  for (*key -* (blk *)iv: i < len / AES_BLOCK_SIZE: i++) {
  AesEcbEncrypt(&ptb[i], key);
  XORBLK(key,&ptb[i], key); // 128-bit XOR (ptb ^ key)
}
}
```

4.12 KODE OTENTIKASI PESAN

Kode Otentikasi Pesan, atau MAC, memberikan perlindungan integritas dan otentikasi asal, dan karenanya dikaitkan dengan kunci pribadi, seperti algoritma enkripsi simetris. MAC mengambil kunci pribadi dan pesan ini sebagai masukan, dan mirip dengan fungsi hash, menghasilkan intisari (kode otentikasi). Penerima pesan dan kode autentikasinya menjalankan algoritma yang sama dengan kunci pribadi yang dibagikan sebelumnya. Jika kodenya sama, ini berarti pesan tersebut tidak hanya sampai secara utuh, namun juga pesan tersebut harus di-hash menggunakan kunci pribadi yang sama.

Dalam protokol keamanan jaringan seperti IPsec, MAC dapat digunakan untuk memberikan perlindungan integritas dan otentikasi asal pesan yang dikirim antara dua rekan yang sebelumnya telah membuat kunci pribadi bersama. Meskipun ada banyak algoritma MAC (CBC-MAC dijelaskan sebelumnya dalam bab ini), algoritma keyed-hash, HMAC, adalah

yang paling umum. Meskipun HMAC yang menggunakan hash SHA-1, yang disebut sebagai HMAC-SHA-1, tidak mengalami kelemahan tabrakan yang ditemukan di SHA-1, varian HMAC SHA-2 (misalnya, HMAC-SHA-256) diperlukan oleh Suite B dan merupakan pilihan terbaik untuk sebagian besar desain tertanam modern. HMAC adalah algoritma yang sederhana dan elegan, didefinisikan (secara setara) dalam RFC 2104 dan FIPS 198.[1819]

4.13 PEMBUATAN ANGKA ACAK

Banyak algoritma kriptografi memerlukan penggunaan angka acak. Misalnya, kunci RSA dan ECC harus dibuat dengan nomor acak untuk memastikan keunikannya. Ketika banyak bit diperlukan, algoritma pembangkit bilangan acak semu (PRNG) dapat digunakan, selama algoritma tersebut diunggulkan dengan pembangkit bilangan acak sebenarnya (TRNG), biasanya berasal dari sumber entropi alami di perangkat keras. Agar sup akronim tetap mengalir, istilah lain yang digunakan FIPS untuk PRNG adalah deterministic random bit generator (DRBG).

Hal penting yang penting bagi perancang sistem tertanam adalah persyaratan bahwa mereka menggunakan prosesor tertanam dengan generator bilangan acak yang sebenarnya (TRNG). Tanpa fitur seperti itu pada prosesor, desainer akan kesulitan menemukan sumber entropi untuk TRNG dan membuktikan bahwa pendekatan tersebut valid. Kebanyakan prosesor tertanam dengan mesin keamanan apa pun akan memiliki TRNG. Namun, pembeli harus berhati-hati: beberapa mesin keamanan perangkat keras yang didokumentasikan menghasilkan "angka acak" mungkin sebenarnya mengeluarkan angka pseudo-acak menggunakan algoritma deterministik. Penting untuk memvalidasi bahwa angka acak benar-benar acak. Salah satu cara untuk melakukan hal ini, selain meminta NSA mengevaluasi TRNG, adalah dengan menguji TRNG dengan uji angka acak yang disetujui FIPS atau NIST.[20]

Pembuatan Angka Acak Sejati

Pengacak non-deterministik menggunakan proses yang tidak dapat diprediksi, bukan algoritma tertentu, untuk menghasilkan keluaran. Output dari pengacak non-deterministik tidak dapat ditentukan bahkan jika penyerang memiliki atau menentukan desain pengacak yang lengkap, termasuk informasi keadaan internal pada waktu tertentu. Dari sudut pandang konseptual, pengacak non-deterministik terdiri dari satu atau lebih sumber entropi dan fungsi gabungan untuk menghasilkan keluaran acak yang terdistribusi secara seragam, seperti yang ditunjukkan pada Gambar 4.25.

Seperti yang ditunjukkan, keluaran acak N-bit sama sekali tidak dapat diprediksi; probabilitas menghasilkan bilangan acak tertentu adalah $1/2^n$, dengan n adalah jumlah bit dalam keluaran acak. Probabilitasnya harus tetap pada $1/2^n$ bahkan jika penyerang memiliki pengetahuan lengkap tentang nomor acak sebelumnya yang dihasilkan oleh pengacak dan memiliki pengetahuan lengkap tentang desain dan implementasi pengacak.

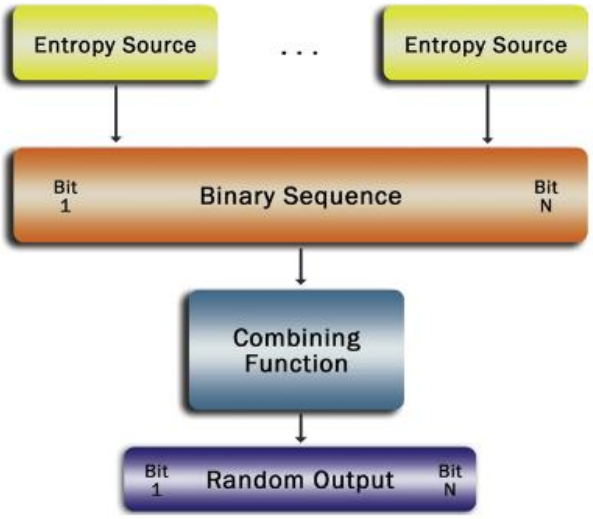
---

[18] HMAC: Keyed-Hashing for Message Authentication, Internet Engineering Task Force, Request for Comments: 2104 (February 1997).

[19] NIST FIPS PUB 198. The Keyed-Hash Message Authentication Code (HMAC) (2002), http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

[20] NIST Special Publication 800-22dRevision 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (April 2010).

Sumber entropi adalah dasar untuk operasi pengacak non-deterministik. Banyak komponen dan proses fisik yang dapat berfungsi sebagai sumber entropi yang dapat diterima. Contohnya termasuk osilator cincin, dioda derau, peluruhan radioaktif, dan derau sinyal bandwidth tinggi pada perangkat elektronik. Sumber entropi yang dapat diimplementasikan menggunakan logika digital sangat berguna karena dapat disintesis dengan array yang dapat diprogram (FPGA) atau khusus aplikasi khusus.



Gambar 4.25 Masukan entropi ke fungsi penggabungan digunakan untuk membuat keluaran acak sebenarnya.

Tanpa fitur seperti itu pada prosesor, desainer akan kesulitan menemukan sumber entropi untuk TRNG dan membuktikan bahwa pendekatan tersebut valid. Kebanyakan prosesor tertanam dengan mesin keamanan apa pun akan memiliki TRNG. Namun, pembeli harus berhati-hati: beberapa mesin keamanan perangkat keras yang didokumentasikan menghasilkan "angka acak" mungkin sebenarnya mengeluarkan angka pseudo-acak menggunakan algoritma deterministik. Penting untuk memvalidasi bahwa angka acak benar-benar acak. Salah satu cara untuk melakukan hal ini, selain meminta NSA mengevaluasi TRNG, adalah dengan menguji TRNG dengan uji angka acak yang disetujui FIPS atau NIST.[21]

Pembuatan Angka Acak Sejati

Pengacak non-deterministik menggunakan proses yang tidak dapat diprediksi, bukan algoritma tertentu, untuk menghasilkan keluaran. Output dari pengacak non-deterministik tidak dapat ditentukan bahkan jika penyerang memiliki atau menentukan desain pengacak yang lengkap, termasuk informasi keadaan internal pada waktu tertentu. Dari sudut pandang konseptual, pengacak non-deterministik terdiri dari satu atau lebih sumber entropi dan fungsi gabungan untuk menghasilkan keluaran acak yang terdistribusi secara seragam, seperti yang ditunjukkan pada Gambar 4.25.

Seperti yang ditunjukkan, keluaran acak N-bit sama sekali tidak dapat diprediksi; probabilitas menghasilkan bilangan acak tertentu adalah $1 / 2^n$, dengan n adalah jumlah bit

[21] NIST Special Publication 800-22dRevision 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (April 2010).

dalam keluaran acak. Probabilitasnya harus tetap pada 1 /2^n bahkan jika penyerang memiliki pengetahuan lengkap tentang nomor acak sebelumnya yang dihasilkan oleh pengacak dan memiliki pengetahuan lengkap tentang desain dan implementasi pengacak.

Sumber entropi adalah dasar untuk operasi pengacak non-deterministik. Banyak komponen dan proses fisik yang dapat berfungsi sebagai sumber entropi yang dapat diterima. Contohnya termasuk osilator cincin, dioda derau, peluruhan radioaktif, dan derau sinyal bandwidth tinggi pada perangkat elektronik. Sumber entropi yang dapat diimplementasikan menggunakan logika digital sangat berguna karena dapat disintesis dengan array yang dapat diprogram (FPGA) atau ASIC sirkuit terintegrasi khusus aplikasi khusus dan tersedia untuk digunakan oleh pengembang sistem tertanam dalam prosesor aplikasi atau co-prosesor tambahan. Meskipun sebagian besar pengembang sistem tertanam yang membaca buku ini dapat dengan mudah memanfaatkan perangkat keras ini dan tidak perlu memahami detail implementasi generator bilangan acak yang sebenarnya, studi kasus berikut disediakan bagi mereka yang ingin menambah pengetahuan dan kedalaman dasar kriptografi mereka.

Studi Kasus: Randomizer berbasis Ring Oscillator

Osilator cincin telah dipelajari dan dimodelkan secara menyeluruh. Sumber entropi osilator cincin adalah variasi penundaan, atau jitter, di seluruh rangkaian, yang menyebabkan keadaan cincin tidak dapat diprediksi. Pembaca yang tidak terbiasa dengan konsep osilator cincin sebaiknya meluangkan waktu sejenak untuk membaca deskripsi yang umum ditemukan, seperti Wikipedia, yang tersedia di Internet.

Gambar 4.26 menyajikan osilator cincin dengan tiga elemen penundaan: D0, D1, dan D2. Setiap elemen penundaan terdiri dari inverter logika sederhana, kecuali yang pertama, D0, yang menggunakan gerbang NAND untuk melipatgandakan saluran pengaktifan eksternal yang menahan osilator cincin dalam kondisi stabil sebelum berosilasi. Saat diaktifkan, osilator cincin memiliki 2N status, dengan N adalah jumlah elemen penundaan dalam cincin. Diagram waktu osilator cincin tiga tahap ini ditunjukkan pada Gambar 4.27.

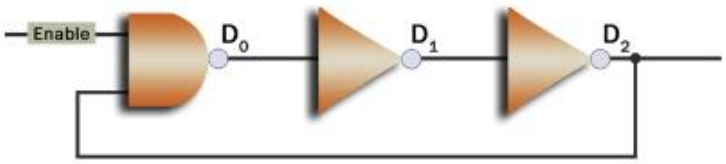
Pada diagram pada Gambar 4.27, Dn mewakili elemen tundaan, P adalah periode, dan d adalah tundaan salah satu elemen. Hanya satu elemen penundaan yang mengubah status pada satu waktu; dengan demikian, osilator cincin menghasilkan keluaran gelombang persegi yang teratur jika dilihat dalam interval pendek. Namun, dalam jangka waktu yang lebih lama, penyimpangan dan jitter menyebabkan osilator cincin menjadi kurang dapat diprediksi, terutama jika jitter lebih besar dari periode osilator. Meskipun tidak dibuktikan di sini, dapat ditunjukkan bahwa keadaan berikutnya dari osilator cincin S1 setelah waktu T, dengan mengabaikan jitter apa pun, berhubungan dengan keadaan awal S0 melalui persamaan berikut:

$$S_1=(S_0+T/d) \bmod (2N)$$

Misalnya, jika d = 5 ns, N = 5, S0 = 0, dan T = 500.000 ns (sampel panjang), maka S1 adalah (0 + 500.000 / 5) mod 10 = 0. Dengan jitter, keadaan selanjutnya adalah

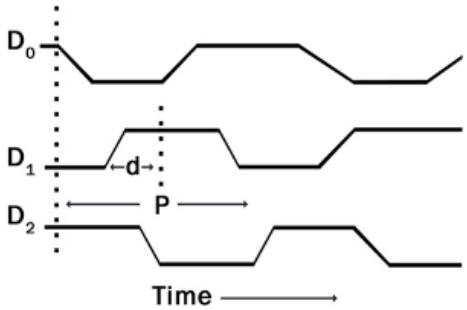
$$S_1=(S_0+T+J) \bmod (2N)$$

Jitter, J, didefinisikan sebagai variasi puncak waktu yang dibutuhkan osilator untuk menyelesaikan N siklus dan bervariasi secara acak dengan noise akibat perubahan suhu dan tegangan di dalam transistor.



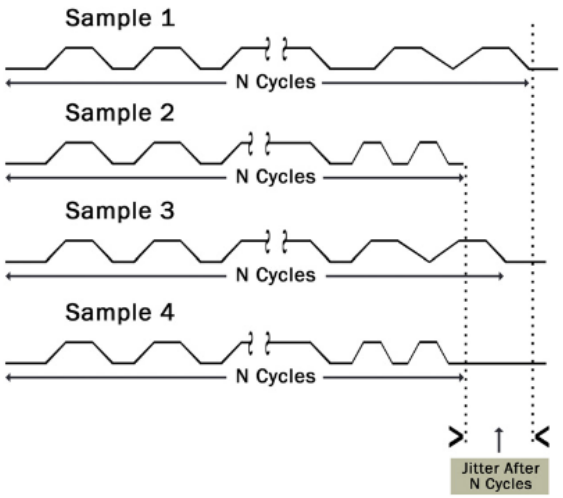
Gambar 4.26 Osilator cincin tiga elemen.

D_0	D_1	D_2
1	0	1
0	0	1
0	1	1
0	1	0
1	1	0
1	0	0
1	0	1



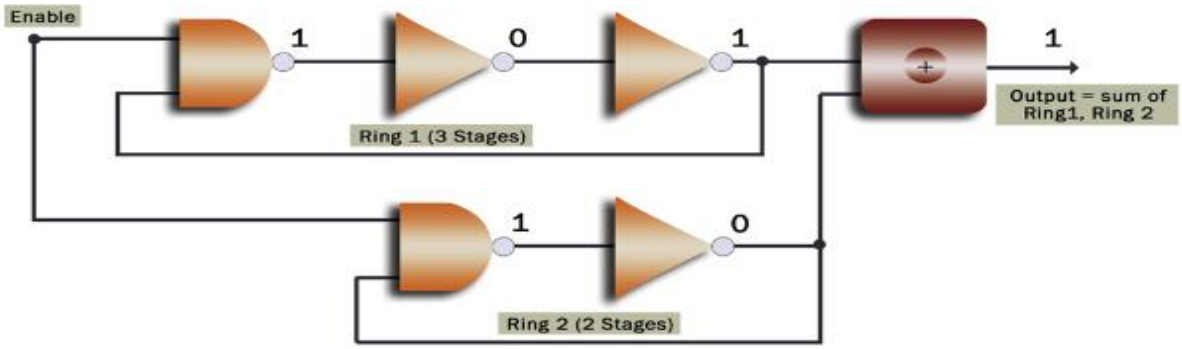
Gambar 4.27 Diagram waktu untuk osilator cincin tiga elemen.

(lihat Gambar 4.28). Persamaan sebelumnya menunjukkan bahwa peningkatan jitter menyebabkan keadaan selanjutnya menjadi tidak dapat diprediksi.



Gambar 4.28 Naik opelet.

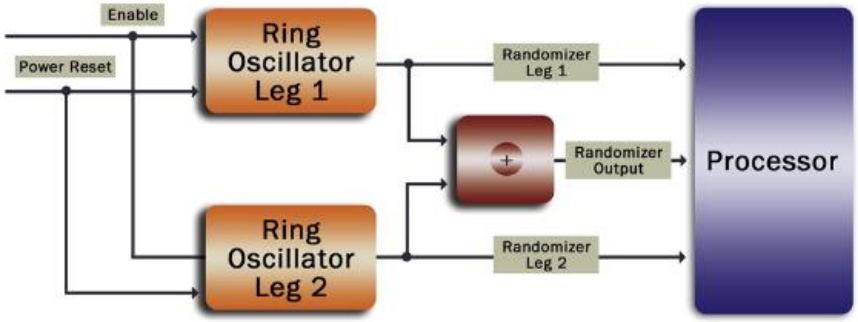
Faktanya, waktu T antar sampel memiliki efek langsung pada jumlah jitter yang terakumulasi antar sampel. Dapat juga ditunjukkan bahwa bergantung pada jitter puncak, J, setelah waktu T, penundaan d, dan periode osilator cincin P, jika J mendekati d dan lebih kecil dari P, keadaan selanjutnya akan memiliki variasi yang kecil. Tetapi jika J lebih besar dari P, keadaan keluaran menunjukkan variasi yang besar. Kita dapat menyimpulkan bahwa osilator cincin tunggal dapat digunakan untuk menghasilkan data acak selama kita mengambil sampelnya setelah jangka waktu yang lama, memastikan bahwa J lebih besar dari P. Pendekatan ini akan menghasilkan laju pengambilan sampel yang sangat lambat. Dalam praktiknya, pengacak menggunakan banyak osilator cincin yang dijumlahkan untuk membentuk keluarannya guna menghindari laju sampel yang panjang. Misalnya, Gambar 4.29 menunjukkan dua osilator cincin sederhana yang terdiri dari dua dan tiga keadaan.



Gambar 4.29 Pengacak dengan dua osilator cincin berbeda.

Seperangkat osilator gabungan, masing-masing dengan jumlah tahapan yang divariasikan secara sengaja, disebut kaki. Selanjutnya, beberapa kaki dapat digabungkan untuk membuat pengacak akhir. Ketika jitter ditambahkan ke masing-masing kaki pengacak dan besarnya jitter acak dan bervariasi antar kaki, dapat dengan mudah diamati bahwa keluaran gabungan menghasilkan lebih banyak osilasi, menghasilkan aliran biner acak ketika diambil sampelnya dari waktu ke waktu.

Untuk menggunakan pengacak sebelumnya untuk data rahasia atau sensitif, pengacak tersebut harus aman dan aman dari kegagalan. Misalnya, jika pengacak yang terdiri dari 20 osilator cincin dengan jumlah elemen penundaan yang bervariasi mengalami kegagalan di beberapa bagian, hal ini dapat menghasilkan keluaran yang bias (tidak lagi acak). Untuk memastikan bahwa keacakan dipertahankan di tengah-tengah banyak kegagalan, desain praktis menggunakan fase pasca-pemrosesan yang mengambil masukan berupa jumlah keluaran dari semua bagian serta keluaran masing-masing bagian, seperti yang ditunjukkan pada Gambar 4.30.

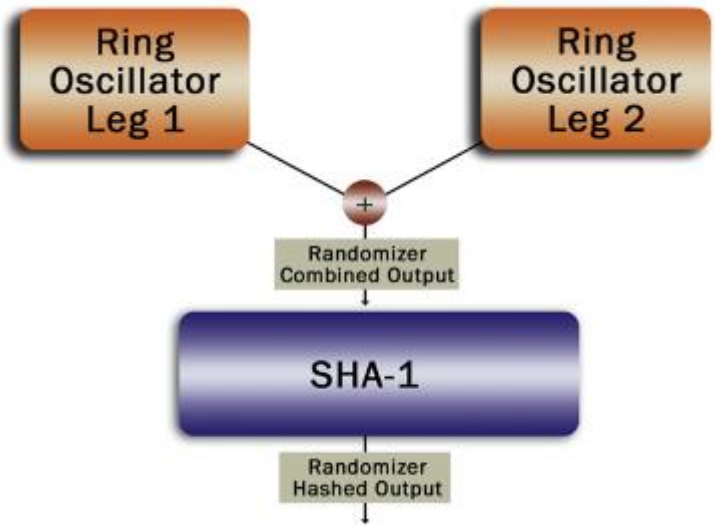


Gambar 4.30 Output osilator redundan.

Setiap kaki memiliki entropi yang cukup untuk menyediakan aliran bit acak; oleh karena itu, bahkan jika salah satu bagian gagal, keluaran gabungan akan menghasilkan aliran acak pada tingkat pengambilan sampel yang tepat. Namun, karena jumlah sampel yang cukup pendek sering kali diperlukan, setiap bagian pengacak harus diuji secara individual untuk menyingkirkan bias statistik.

Prosesor yang mengambil sampel keluaran menghitung jumlah aliran bit individual dari setiap kaki dan membandingkannya dengan keluaran pengacak. Jika tidak cocok, kegagalan terdeteksi dan alarm dibunyikan. Selain itu, prosesor membaca dan mengumpulkan N bit dari setiap kaki dan dari keluaran gabungan, serta menghitung jumlah bit yang ditetapkan dalam setiap aliran N-bit. Prosesor menggunakan teori probabilitas untuk mendeteksi bit-bit yang diatur dalam jumlah yang sangat banyak. Misalnya, untuk aliran 256-bit, rentang set bit yang diperbolehkan mungkin 112 hingga 144. Jika semua aliran berisi jumlah set bit yang dapat diterima, maka pengujian berhasil. Tentu saja, ada kemungkinan kecil terjadinya hasil negatif palsu. Jika hal ini terjadi, kumpulan data baru harus diambil dan diuji. Berdasarkan teori probabilitas, jika banyak (misalnya, enam) kumpulan data gagal dalam pengujian rentang bit yang ditetapkan, maka kegagalan dianggap terjamin secara statistik dan alarm dibunyikan; jika tidak, data sampel digunakan sebagai aliran bit acak.

Tes sebelumnya tidak mendeteksi semua jenis kegagalan bias dalam pengacak. Tes keacakan dapat dijalankan saat power up atau secara berkala (misalnya, setahun sekali), mengambil sampel bit yang jauh lebih banyak (misalnya, satu juta) dan mencari anomali tambahan, seperti string yang sangat panjang dengan nilai bit yang sama. Pengacakan praktis menggunakan pengacak non-deterministik (osilator cincin) yang dikombinasikan dengan fungsi linier seperti hash SHA-1 untuk menghasilkan keluaran akhir, seperti yang ditunjukkan pada Gambar 4.31.



Gambar 4.31 Penggunaan fungsi hash untuk mempersempit keluaran pengacak.

Pendekatan ini menghasilkan keluaran yang lebih kecil dengan sekumpulan bit acak dari pengacak dua kaki. Output hash dapat disimpan dalam memori untuk digunakan selanjutnya dalam pembuatan kunci kriptografi. Menggunakan aliran hash menghaluskan bias yang mungkin lolos dari osilator cincin.

Singkatnya, pengacak yang menggunakan osilator cincin harus dirancang dan diterapkan dengan hati-hati untuk memastikan bahwa pengacak tersebut benar-benar acak dan menggabungkan redundansi bawaan dan deteksi kesalahan. Desain pengacak harus memastikan bahwa setiap kaki memiliki entropi (jitter) yang cukup untuk menghasilkan aliran bit acak. Penguncian cincin harus dihindari, dan setiap kaki harus menghasilkan jitter independen tanpa mempengaruhi kaki lainnya. Pengujian jangka pendek dan jangka panjang yang memadai harus dilakukan untuk memastikan pengacak tersebut aman dari kegagalan.

Pembuatan Bilangan Acak Pseudo

Pengacak deterministik menghasilkan keluaran berdasarkan algoritma tertentu dan bukan proses probabilistik seperti osilator cincin. Dengan demikian, jenis pengacak deterministik atau pengacak pseudo-acak ini, meskipun tampak menghasilkan aliran bit acak, menghasilkan keluaran yang benar-benar dapat diprediksi dengan keadaan awal yang diketahui.

> *Generator bilangan acak semu, yang diunggulkan dengan input acak sejati, diperlukan untuk aplikasi yang harus menghasilkan materi acak kriptografi dengan kecepatan lebih tinggi daripada yang dapat dibuat oleh generator bilangan acak sejati lokal suatu sistem.*

PRNG yang tidak diunggulkan dengan input acak yang benar tidak boleh digunakan untuk pembuatan kunci privat. Secara berkala, selama pengoperasian TRNG, benih baru harus dihasilkan dan dimasukkan ke PRNG. Jika algoritme dirancang dengan cermat, pengacak deterministik akan lulus uji statistik tradisional. Otoritas sertifikasi (misalnya NSA, FIPS) akan melaksanakan uji keacakan tersebut untuk memvalidasi PRNG. Algoritme PRNG dapat memiliki kompleksitas yang beragam, mulai dari generator kongruensial linier (LCG) yang menggunakan aritmatika modular hingga kombinasi algoritma enkripsi dan hash.

Generator bilangan acak semu harus dirancang dengan hati-hati untuk menghindari menghasilkan keluaran linier yang mudah rusak. PRNGS harus menggunakan algoritma non-linier seperti generator keystream berbasis AES.

Pilihan PRNG yang aman adalah algoritma yang disetujui FIPS, seperti yang diterbitkan dalam Publikasi Khusus NIST 800-90. Daftar DRBG yang disetujui FIPS dapat ditemukan di Lampiran C standar FIPS 140-2.[22]

Publikasi Khusus NIST 800-90

Entri paling modern dalam daftar yang disetujui FIPS adalah algoritma yang ditemukan dalam NIST Special Publication 800-90, diterbitkan pada tahun 2007 dan diperbarui pada bulan Mei 2011 dengan draf revisi, SP 800-90A.[23] Beberapa algoritma yang lebih populer dari ini standar dibahas pada bagian berikut. Perhatikan bahwa spesifikasi algoritme berikut cocok dengan generator bit pseudo-acak yang ditentukan dalam ANSI X9.82, Bagian 3 Generator Bit Acak Deterministik.[24]

HMAC_DRBG

HMAC_DRBG, sesuai dengan namanya, menggunakan algoritma kode otentikasi pesan HMAC untuk menghasilkan angka pseudo-acak. Variasi HMAC_DRBG menggunakan algoritma HMAC yang berbeda (misalnya, HMAC-SHA1 versus HMAC-SHA-256) dan harus diasumsikan memiliki kekuatan relatif yang sepadan. Antarmuka khas ke DRBG adalah sebagai berikut:

- **Instantiat**e: Metode ini menginisialisasi DRBG dengan memberikan input acak (entropi) bit minimum yang sesuai.
- **Hasilkan**: Inti dari algoritma memanggil algoritma HMAC beberapa kali secara serial, menggunakan entropi sebagai masukan pada berbagai tahapan. Setiap pemanggilan fungsi generate menghasilkan string bit pseudo-acak yang panjangnya sama dengan panjang algoritma hash yang mendasarinya. Untuk HMAC-SHA-256, string bit pseudo-acak 256-bit dihasilkan.
- **Re-seed**: Sebagian besar implementasi akan menyediakan operasi re-seed yang mengambil masukan entropi baru untuk menyegarkan keadaan internal DRBG. Implementasinya mungkin membatasi jumlah bit yang dapat dihasilkan tanpa pemanggilan ulang. Penyemaian ulang secara berkala dipandang sebagai kebijakan yang baik untuk mengatasi ancaman yang membahayakan entropi asli atau keadaan internal DRBG.

Spesifikasi HMAC_DRBG cocok untuk implementasi yang efisien, terutama jika mikroprosesor tertanam sudah memiliki akselerasi perangkat keras untuk hashing. Implementasi HMAC_DRBG mungkin hanya terdiri dari 100 baris kode, sehingga mudah untuk dipahami dan diverifikasi terhadap spesifikasi.

RKT_DRBG

Seperti namanya, CTR_DRBG menggunakan cipher blok dalam mode counter untuk melakukan bit jumbling. AES dalam Mode Penghitung (AES-CTR) adalah pilihan yang tepat, terutama jika sistem sudah menggunakan AES-CTR untuk enkripsi. Mirip dengan

[22] Annex C: Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules (DRAFT) (June 14, 2011).

[23] NIST Special Publication 800-90AdRevision 1. Recommendation for Random Number Generation Using Deterministic Bit Generators (Revised) (May 2011).

[24] ANSI X9.82-3-2007. Random Number GenerationdPart 3: Deterministic Random Bit Generators (September 2007).

HMAC_DRBG, CTR_DRBG memerlukan input acak yang sebenarnya dan biasanya dipanggil menggunakan metode instantiate, generate, dan re-seed. Salah satu keunggulan CTR_DRBG dibandingkan HMAC_DRBG adalah keamanan algoritmik implementasi CTR_DRBG menggunakan AES mengurangi keamanan AES-CTR itu sendiri. Sebaliknya, keacakan semu bukanlah properti fungsi hash yang awalnya dipahami.

Oleh karena itu, ada kekhawatiran bahwa kelemahan dapat ditemukan di HMAC_DRBG meskipun tidak ada kelemahan pada hash yang mendasarinya. Kelemahan pada CTR_DRBG memerlukan kelemahan yang sama pada AES-CTR. Kriptografi tanpa nomor acak yang sebenarnya seharusnya dianggap tidak lebih dari sekadar kebingungan yang dapat dibobol oleh penyerang canggih. Peretas diduga menyusupi konsol game Sony PlayStation 3, mengklaim bahwa akar subversi tersebut disebabkan oleh penggunaan fungsi penghasil nomor acak berikut oleh Sony:

```
Int getRandomNumber ( )
{
     Return 4;
{
```

Pengembang sistem tertanam: jangan biarkan hal ini terjadi pada Anda! Para penyerang, dari sebuah organisasi bernama failOverflow, mempresentasikan serangan mereka di Chaos Communication Congress 2010. Karena generator nomor acak yang tidak aman ini, para penyerang diduga mampu menghitung kunci pribadi yang digunakan untuk menandatangani gambar boot dan kemudian mem-boot Linux pada sebuah PS3.

4.14 MANAJEMEN KUNCI UNTUK SISTEM TERTANAM

Seperti yang telah dibahas sebelumnya, keamanan sistem kriptografi bergantung pada kerahasiaan keseluruhan kunci privat sistem. Musuh yang memperoleh kunci atau segmen kunci, secara umum, dapat memperoleh teks biasa dari teks tersandi yang dihasilkan secara aktif atau ditangkap sebelumnya. Pada pandangan pertama, menjaga kerahasiaan kunci tampak sederhana: sumber kunci (generator kunci) dan tujuan (pengguna) adalah agen tepercaya yang diinstruksikan untuk melindungi kunci saat berpindah dari sumbernya ke tujuan akhirnya dalam sistem tertanam. Banyak sistem saat ini mengandalkan proses ini untuk mengamankan kunci. Kunci dihasilkan dalam bentuk teks biasa, kadang-kadang disebut kunci merah, dan dikirim ke pengguna melalui kurir atau cara aman lainnya dan kemudian dimuat ke dalam sistem tertanam.

Setelah kunci dimuat, banyak sistem tertanam tidak mau repot-repot mengganti kunci secara berkala. Mungkin mengejutkan untuk mengetahui bahwa beberapa sistem perbankan penting yang digunakan saat ini, misalnya anjungan tunai mandiri (ATM) yang menggunakan kriptografi untuk melindungi transaksi kartu, tidak mengubah kunci di dalam ATM kecuali jika terjadi kegagalan. Meski begitu, kunci aslinya sering kali diisi ulang!

Studi Kasus: Kasus Mata-Mata Walker

Peristiwa terkenal lainnya dalam sejarah keamanan dan kriptografi pada dasarnya dapat disebabkan oleh manajemen kunci yang buruk. Sistem rahasia pemerintah AS

menggunakan sistem manajemen kunci yang terlalu sederhana dan terbukti membawa bencana. Serangan dasarnya adalah sebagai berikut: administrator kunci tepercaya (John Anthony Walker, Jr.) mereplikasi serangkaian kunci merah dan memberikannya kepada musuh, seperti yang ditunjukkan pada Gambar 4.32.

Tentu saja, serangan itu tidak mungkin terjadi tanpa adanya ancaman dari dalam. Tidak diragukan lagi, perlindungan dari ancaman orang dalam adalah salah satu tantangan terbesar bagi semua bentuk keamanan. Namun, kami menunjukkan bagaimana manajemen kunci yang tepat dapat membuat serangan orang dalam ini menjadi lebih sulit, bahkan mustahil, untuk dilakukan. Contoh ini menggambarkan bagaimana pemerintah AS menyiarkan pesan ke kapal selam AS di laut. Pesan siaran RF dikirim dari stasiun siaran berbasis darat di wilayah AS ke semua kapal selam. Pendekatan siaran memungkinkan kapal selam yang ditempatkan di seluruh dunia memperoleh informasi penting secara bersamaan dan cepat. Tentu saja, beberapa pesan bersifat sangat rahasia, menggambarkan posisi kapal selam, pesan komando dan kendali nuklir, dan rahasia lainnya. Oleh karena itu, siarannya harus dienkripsi dengan kuat.

Gambar 4.32 Kompromi kunci merah.

Karena harus tetap berada di bawah air untuk jangka waktu yang lama, setiap kapal selam disediakan, sebelum diluncurkan, sebuah tabung berisi banyak kunci merah untuk digunakan selama misi. Daftar kunci dicetak pada pita kertas, dan setiap kunci hanya digunakan untuk jangka waktu terbatas yang telah ditentukan, misalnya satu hari. Kuncinya diganti secara manual dengan kunci tabung berikutnya di penghujung hari. Penjaga atau administrator kriptografi bertanggung jawab untuk memastikan bahwa setiap kunci dimuat dengan benar ke dalam unit komunikasi kriptografi.

Sistem ini disusupi ketika salah satu penjaga kriptografi di kapal selam, John Anthony Walker, Jr., membuat salinan tambahan dari daftar kunci pita kertas dan menjual salinannya ke KGB. Dengan kepemilikan kunci yang disalin, Soviet dapat memantau dan mendekripsi semua pesan yang dikirim oleh stasiun penyiaran. Pada dasarnya, kapal selam Soviet dapat meniru kapal selam AS karena mereka memiliki mekanisme otentikasi: kunci siaran yang digunakan untuk mendekripsi semua pesan. Kebijakan penggantian kunci harian sudah baik: jika kunci disusupi, hanya pesan yang dienkripsi dan didekripsi pada hari yang berlaku yang akan berisiko. Namun, dalam kasus ini, ancamannya bukanlah penyadap yang mencoba

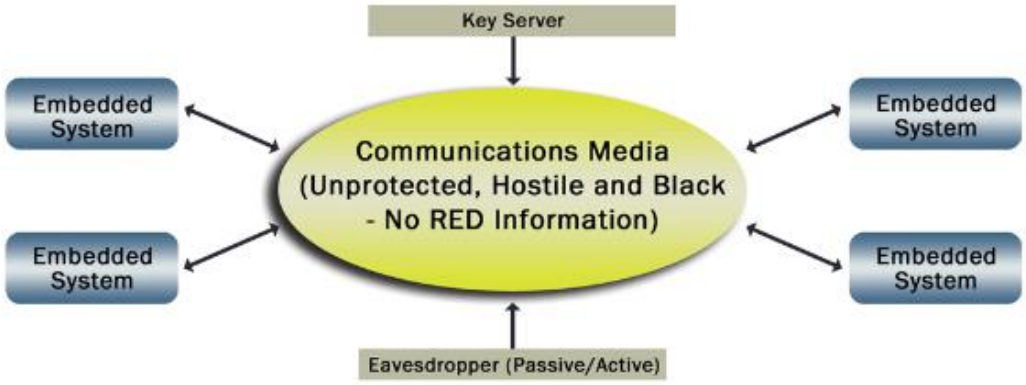
mendapatkan kunci, melainkan ancaman orang dalam yang membocorkan banyak kunci tanpa terdeteksi dalam waktu lama.

Skema manajemen kunci sederhana yang memuat dan memuat ulang kunci merah secara manual oleh personel tepercaya mengakibatkan kegagalan besar sistem kripto, yang diyakini banyak orang sebagai kasus spionase terburuk dalam sejarah AS dan hilangnya keamanan yang benar-benar mengubah keseimbangan kekuatan antara Amerika. Amerika dan Uni Soviet. Contoh ini menyoroti pentingnya sistem manajemen kunci yang kuat untuk produk apa pun yang menggunakan kriptografi untuk melindungi data, baik data tersebut bersifat rahasia, sensitif, atau pribadi. Minimal, rencana pengelolaan utama perlu diartikulasikan terlebih dahulu dan dipresentasikan, didiskusikan, dan disetujui sebelum desain atau implementasi.

Manajemen Kunci Model Umum

Gambar 4.33 menyajikan infrastruktur umum sistem tertanam yang terhubung ke media komunikasi (RF, Internet, sistem peralihan telepon, jaringan perbankan swasta, dll.) dalam berbagai mode komunikasi, termasuk duplex point-to-point, duplex point-to-multipoint, dan siaran simpleks. Dalam model ini, semua jenis kunci diperlukan untuk berbagai konfigurasi dan misi yang terkait dengan perangkat tertanam ini. Untuk meminimalkan keterlibatan manusia dan memaksimalkan otomatisasi dalam manajemen kunci, model ini menyediakan server kunci tepercaya yang bertindak sebagai penghasil kunci pusat untuk semua kunci yang diperlukan.

Model ini membahas keseluruhan siklus hidup suatu kunci, mulai dari pembuatan hingga penghancurannya. Prosesnya dimulai dengan registrasi sistem tertanam dan diakhiri dengan pencabutan kunci (sistem tertanam tidak lagi dipercaya, dimusnahkan, atau tidak ada lagi dalam inventaris).



Gambar 4.33 Infrastruktur komputasi tertanam dengan server kunci terpusat.

(1) Sistem Titik Akhir Registrasi perlu mendaftar ke server kunci, dan ini biasanya memerlukan semacam pemuatan kunci awal (kunci simetris atau publik). Jika diperlukan kunci simetris, biasanya berwarna hitam (tidak boleh merah). Untuk kunci simetris, pemuatan kunci awal harus dilakukan di pabrik atau di fasilitas tepercaya sebelum dikirim ke lokasi. Yang terbaik adalah kunci awal yang dimuat hanya digunakan di dalam perangkat yang tertanam dan digunakan secara ketat untuk

mendekripsi kunci operasional baik yang dikirim oleh server kunci atau dimuat awalnya dalam bentuk hitam.

(2) Pembuatan Kunci Beberapa mikroprosesor menyediakan fungsi pembuatan kunci satu kali yang dapat diprogram (OTP) berbasis perangkat keras. Dalam kasus lain, proses instalasi firmware sistem tertanam dapat mencakup langkah pembuatan kunci internal. Algoritme pembangkitan kunci disertakan dalam sebagian besar implementasi kriptografi perangkat lunak atau perangkat keras.

> *Idealnya, kunci privat dilahirkan di dalam dan tidak pernah meninggalkan sistem tertanam di mana kunci tersebut digunakan. Namun terkadang, kunci privat harus disediakan secara eksternal. Misalnya, kunci Perlindungan Konten Definisi Tinggi (HDCP) dilisensikan dan didistribusikan oleh Intel Corporation. Bagaimana kunci-kunci ini dapat diangkut dengan aman dari server pembuat kunci tepercaya asalnya ke sistem tertanam target? Sistem tertanam harus dikunci secara manual oleh administrator lokal yang tepercaya (bukan solusi yang dapat diskalakan), atau perangkat harus memiliki kemampuan untuk membuat sambungan aman (misalnya, menggunakan IPsec atau TLS) ke server penyediaan kunci. Hal ini tentu saja menimbulkan pertanyaan besar: bagaimana kunci privat untuk koneksi tersebut disediakan? Kunci pribadi awal dapat dipasang pada waktu produksi, memberikan kemampuan untuk berbicara hanya dengan server penyediaan di bawah kendali pelanggan sistem tertanam. Terlepas dari bagaimana kunci disediakan, perancang sistem tertanam harus memiliki rencana dan pendekatan yang memberikan keamanan dan skalabilitas yang memadai untuk memenuhi persyaratan produk. Untuk sebagian besar sistem kriptografi modern, diperlukan beberapa kunci fungsional.*

Menggunakan kunci yang sama untuk fungsi yang berbeda (seperti enkripsi data dan tanda tangan digital) bukanlah ide atau praktik yang baik, karena kompromi kunci (misalnya, karena kelemahan dalam fungsi atau penerapan kunci tertentu) akan mempengaruhi semua fungsi yang menggunakan tombol yang sama. Sistem kriptografi yang paling praktis, aman, dan dirancang dengan baik menggunakan kunci berbeda untuk fungsi berbeda. Beberapa contoh tujuan utama adalah sebagai berikut: Kunci Enkripsi Lalu Lintas (TEK): Kunci ini hanya digunakan untuk enkripsi massal atau dekripsi pesan atau lalu lintas lainnya.

- **Kunci Enkripsi Kunci** (KEK): Kunci ini digunakan untuk mengenkripsi atau mendekripsi TEK; biasanya, server kunci menghasilkan KEK, dan dimasukkan ke dalam sistem tertanam sekali seumur hidup. KEK dimuat dengan aman di fasilitas terpercaya, umumnya di pabrik atau fasilitas muatan utama awal (KLF). Setelah KEK dimuat, KEK digunakan oleh sistem tertanam untuk memuat TEK berikutnya baik secara manual atau melalui udara dari server kunci. TEK dimuat dalam warna hitam (dienkripsi dengan KEK) dan didekripsi oleh sistem tertanam bila diperlukan. KEK unik untuk setiap perangkat yang tertanam, meskipun TEK umum dimuat ke banyak sistem tertanam. Oleh karena itu, kompromi antara satu perangkat tertanam dan KEK-nya tidak membahayakan sistem lainnya. Server kunci, setelah diberitahu bahwa KEK telah disusupi, dapat memberikan TEK baru (jika dibagikan) ke sistem yang tidak dikompromikan.

- **Kunci Keamanan Transmisi** (TSK): Kunci yang digunakan untuk memberikan keamanan transmisi dan bukan keamanan lalu lintas. Misalnya, tautan stasiun bumi satelit ke satelit mungkin dilindungi pada lapisan kedua dengan enkripsi terautentikasi berbasis TSK, meskipun lalu lintas atau muatan data satelit sebelumnya telah dienkripsi oleh TEK sebelum tiba di stasiun bumi. Perlindungan tambahan ini dapat digunakan sebagai fitur anti-jam di beberapa sistem satelit. Untuk sistem darat, TSK terkadang digunakan untuk melindungi sumber internal dan alamat target pesan, menawarkan keamanan arus lalu lintas dan ketersediaan tautan komunikasi. Misalnya, dalam aplikasi suara aman pada umumnya, data suara payload dilindungi oleh enkripsi TEK, namun nomor telepon pihak yang menelepon dan dihubungi dikirim dengan jelas sebelum membuat sambungan aman. Kelemahan ini terdapat pada sebagian besar aplikasi suara aman modern. Seorang penyadap dapat dengan mudah menentukan siapa yang menelepon satu sama lain serta kepadatan lalu lintas. Tautan terenkripsi ATSK, bila digunakan, sering diterapkan antara node utama sistem komunikasi, seperti jalur T1. Seorang penyadap yang memonitor saluran T1 yang diamankan transmisi melihat data sandi TSK yang berkelanjutan meskipun lalu lintas suara payload rendah atau tidak ada.
- **Kunci Penyimpanan** (SK): SK digunakan untuk melindungi data yang tidak digunakan (DAR), disimpan baik secara lokal dalam sistem tertanam atau jarak jauh, seperti penyimpanan terpasang jaringan (NAS) atau jaringan area penyimpanan perusahaan (SAN). SK dapat dibuat secara internal oleh sistem tertanam atau oleh server kunci dan dikirim secara hitam ke perangkat tertanam.

(3) Distribusi Kunci

Sistem tipikal memerlukan kunci (publik) simetris dan asimetris untuk distribusi. Setelah sistem tertanam didaftarkan, kunci tambahan diberikan padanya. Semua kunci yang dikirim dari server kunci ke perangkat yang tertanam dienkripsi dalam mode point-to-point, biasanya menggunakan KEK setiap perangkat yang tertanam.

(4) Perubahan Kunci

Sistem yang kuat dan dirancang dengan baik harus mengganti kunci sesering mungkin untuk meminimalkan jendela kerentanan kunci yang disusupi. Semakin pendek periode penggunaan kunci, semakin pendek volume data yang disusupi. Periode penggunaan ini disebut periode kripto. Untuk sistem tertanam yang telah disusupi dan kuncinya mungkin diperoleh oleh musuh, perubahan kunci sangat penting untuk perangkat tertanam lainnya yang berpotensi menggunakan TEK yang sama dengan sistem yang disusupi. Perubahan kunci harus dimulai segera setelah penyusupan terdeteksi, meskipun hal tersebut mengganggu lalu lintas selama durasi perubahan kunci. Jenis perubahan kunci ini disebut perubahan kunci supersesi.

Sayangnya, banyak sistem tertanam otonom yang diterapkan di lapangan saat ini tidak mengelola perubahan penting secara efektif, dan sistem ini sering kali berisiko mengalami kompromi yang tidak terdeteksi. Faktanya, banyak sistem tertanam yang diterapkan tanpa kemampuan untuk mengubah kunci secara berkala atau supersesi. Hal ini memungkinkan sumber daya disusupi tanpa terdeteksi selama berbulan-bulan

atau bertahun-tahun. Jelasnya, yang terbaik adalah mengganti kunci secara berkala, meskipun memerlukan pemuatan kunci baru secara manual. Kita semua menderita masalah penggantian kunci; misalnya, kita benci mengubah kata sandi yang kita gunakan untuk perbankan online, jejaring sosial, dan PC kita. Namun dari sudut pandang keamanan, dan juga ketenangan pikiran, sangat disarankan untuk sering melakukan perulangan parameter keamanan pribadi.

(5) Pemusnahan Kunci

Semua kunci yang periode kriptonya telah habis masa berlakunya atau telah disusupi harus dimusnahkan di dalam sistem tertanam dan server kunci, dan catatannya harus dipelihara, untuk menghindari penggunaan kembali kunci yang telah disusupi.

(6) Perlindungan Kunci

Perlindungan kunci, baik dari sudut pandang fisik maupun logis/perangkat lunak, adalah salah satu fitur keamanan paling penting dalam sistem tertanam yang menggunakan kriptografi, namun sering kali diabaikan atau tidak ditangani dengan baik.

Jika suatu sistem tidak memiliki perlindungan yang memadai, musuh dapat memperoleh kunci tersebut dan sepenuhnya membahayakan aset apa pun yang dilindungi oleh kunci tersebut. Sayangnya, ada banyak cara, beberapa di antaranya sangat canggih, agar musuh dapat memperoleh kunci yang disimpan dalam sistem tertanam. Walaupun perlindungan dengan bukti penuh (atau sedekat mungkin) mungkin terlalu mahal untuk sistem tertanam, hal ini tidak berarti bahwa masalah ini harus diabaikan. Minimal, perancang sistem harus merumuskan rencana perlindungan berdasarkan lingkungan ancaman dan nilai sumber daya yang dilindungi.

Lingkungan ancaman sering kali dipengaruhi oleh lokasi fisik sistem tertanam dan apakah terdapat bentuk perlindungan fisik lain di luar perangkat tertanam itu sendiri. Perangkat tanpa pengawasan yang tidak berada di lingkungan yang dilindungi secara fisik (seperti bangunan yang dijaga secara wajar) adalah yang paling rentan terhadap serangan pasif dan berpotensi aktif, termasuk pencurian sistem tertanam itu sendiri. Akses fisik langsung terhadap sistem dapat mengakibatkan deteksi atau ekstraksi kuncinya. Ketika sebuah pesawat pengintai militer EP-3 A.S. mendarat darurat di Tiongkok, semua barang elektronik, termasuk komunikasi kriptografi dan perangkat penyimpanan, berada di tangan militer Tiongkok; bayangkan tambang emas informasi yang diekstraksi. Peristiwa ini dikenal dengan nama Insiden Pulau Hainan.

Ada banyak teknik dan praktik desain yang terjangkau yang dapat ditanamkan ke dalam sistem tertanam untuk perlindungan kunci. Beberapa dijelaskan selanjutnya. Penyimpanan kunci non-volatil: Hindari menyimpan kunci teks biasa (dan kata sandi) dalam memori non-volatil. Ini termasuk kunci enkripsi pribadi serta kunci publik yang digunakan untuk otentikasi kriptografi kunci publik. Meskipun bagian publik dari pasangan kunci asimetris tidak perlu dirahasiakan untuk menjaga kekuatan kriptografi, kunci ini sering kali digunakan untuk membuktikan keaslian perangkat lunak, firmware, atau data penting lainnya. Jika penyerang dapat mengganti kunci publik teks biasa dengan kunci penyerang, fungsi autentikasi akan salah memvalidasi firmware berbahaya. Kunci yang disimpan dalam memori permanen harus selalu dienkripsi.

Ketika subsistem kriptografi tidak beroperasi (misalnya, ditempatkan dalam kondisi tidur untuk mengurangi daya), setiap kunci yang dikelola secara internal harus dienkripsi sebelum disimpan ke dalam memori non-volatil. Namun, di sebagian besar sistem tertanam yang menjalankan fungsi keamanan, beberapa kunci teks biasa harus ada secara fisik.

> *Kunci pribadi yang disimpan dalam memori non-volatil utama sistem tertanam (misalnya, memori flash atau hard disk) harus dienkripsi dan didekripsi dengan KEK teks biasa yang idealnya disimpan dalam penyimpanan non-volatil di luar prosesor aplikasi utama dan dilindungi dari serangan fisik.*

Sistem penyimpanan kriptografi ini terkadang disebut elemen aman. Penyimpanan yang dilindungi sering kali digabungkan dengan akselerator kriptografi. Ini datang dalam berbagai bentuk, termasuk

- Mesin kriptografi dalam SoC yang sama dengan prosesor aplikasi
- Pisahkan mikrokontroler diskrit
- Modul Platform Tepercaya (TPM)
- Modul Keamanan Perangkat Keras (HSM)

Tanpa elemen aman yang terpasang pada sistem tertanam, kunci harus disimpan dalam flash atau disk, sehingga lebih rentan terhadap serangan eksternal dan serangan perangkat lunak akibat kelemahan dalam sistem operasi yang mengendalikan platform.

Selain serangan saluran samping berbasis perangkat lunak yang umum menggunakan analisis perilaku cache, serangan terhadap kunci mencakup gangguan fisik dan analisis daya sederhana dan diferensial (SPA, DPA). Serangan ini terbukti cukup praktis di alam liar. Misalnya, pada bulan September 2011, peneliti kriptografi mengungkapkan bahwa mereka mampu memulihkan kunci pribadi dari mikrokontroler kartu pintar komersial, NXP DESFire MFICD40, dalam hitungan jam menggunakan serangan analisis daya yang memerlukan peralatan bernilai beberapa ribu dolar untuk memulihkannya melakukan.[25]

Tujuan dari penyimpanan kunci yang aman adalah untuk mencegah serangan tersebut. Kemampuan pendekatan penyimpanan kunci berbasis perangkat keras untuk mencegah serangan tersebut bervariasi. Untuk elemen keamanan komersial, tingkat sertifikasi FIPS 140-2 merupakan ukuran penting dari ketahanan ini. Sertifikasi FIPS 140-2 dibahas nanti dalam bab ini.

Status yang Dihadiri: Untuk subsistem tertanam yang dihadiri, praktik yang baik adalah menyediakan mekanisme otentikasi pengguna, baik melalui kata sandi atau token fisik, untuk memungkinkan penggunaan subsistem kriptografi. Ketika sistem tidak dalam keadaan aktif, terautentikasi, dan dihadiri, subsistem kriptografi memasuki mode terlindungi di mana semua kunci di dalamnya dienkripsi. Memasukkan token atau memasukkan kembali kata sandi akan membuka subsistem kriptografi, memungkinkan dekripsi kunci dalam memori non-volatil untuk penggunaan operasional.

[25] Oswald D, Paar C. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World CHES 2011, Nara; September 30, 2011

Keadaan Tanpa Pengawasan: Untuk sistem tertanam yang otonom, diperlukan perlindungan kunci yang kuat karena kita harus berasumsi bahwa musuh akan memiliki waktu yang tidak terbatas untuk melakukan serangan pasif atau aktif guna memulihkan kunci. Perlindungan harus mencakup zeroisasi kunci saat mendeteksi pemeriksaan fisik (perangkat keras atau perangkat lunak) dan zeroisasi saat mendeteksi kehilangan daya hingga tingkat tertentu. Ada banyak serangan lain yang mungkin dilakukan oleh musuh, dan keputusan awal harus dibuat untuk memasukkan atau mengecualikan tindakan balasan dalam rancangannya.

Misalnya, musuh mungkin memasukkan dalam repertoarnya serangan aktif yang memaksa kegagalan mikroprosesor subsistem kriptografi. Untuk mengatasi ancaman ini, pertimbangan harus diberikan agar desain subsistem kriptografi aman dari kegagalan. Dengan kata lain, untuk mencegah kebocoran kunci ke antarmuka eksternal ketika terjadi kegagalan perangkat keras, perangkat harus berisi sirkuit alarm yang menghambat semua operasi dan memusatkan semua kunci yang mudah menguap ketika kegagalan subsistem kriptografi terdeteksi. Pendekatan lanjutan untuk pengamanan kegagalan adalah dengan menggunakan sirkuit redundan yang dapat mendeteksi ketidaksesuaian fungsional, memberikan peringatan dini terhadap kegagalan sistematis.

(7) Pencabutan Kunci

Pencabutan sering kali merupakan kejadian umum dalam siklus hidup ketika masa pakai sertifikat kunci publik berakhir.

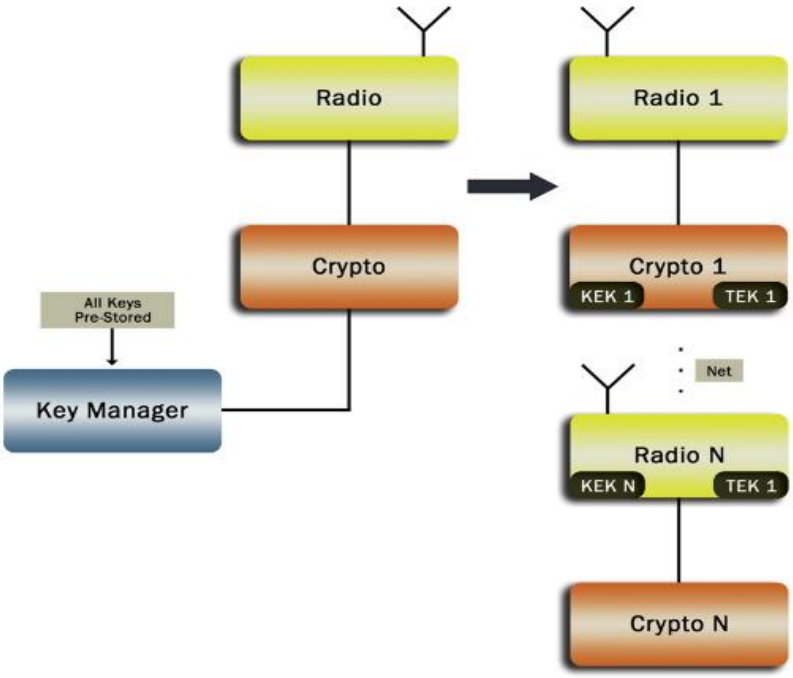
Untuk sistem tertanam yang hilang, dicuri, atau diketahui subsistem kriptografinya telah disusupi, ketentuan harus dibuat untuk mencabut semua kunci yang terkait dengan unit yang disusupi atau hilang. Perangkat lain dalam pandangan dunia sistem yang disusupi (misalnya, rekan komunikasi di jaringan) harus diberi tahu bahwa kunci yang dipermasalahkan telah dicabut. Setiap sistem tertanam pada gilirannya harus memusatkan perhatian pada semua kunci yang terpengaruh.

Kunci yang terkait dengan sistem tertanam dengan masa pakai lapangan yang pendek mungkin tidak mendukung pencabutan sama sekali. Ketika masa pakainya selesai, seluruh sistem yang tertanam akan dibuang, termasuk kunci privatnya. Dalam kasus lain, pencabutan menjadi tidak praktis karena kurangnya kemampuan komunikasi ke server pencabutan yang menyediakan daftar pencabutan terkini. Dalam kasus ini, satu-satunya jalan keluar untuk kunci yang dicabut adalah menginstal ulang calon rekan dengan daftar statis kunci yang dicabut.

Studi Kasus Manajemen Kunci

Studi Kasus: Jaringan Radio Push-to-Talk Nirkabel yang Aman

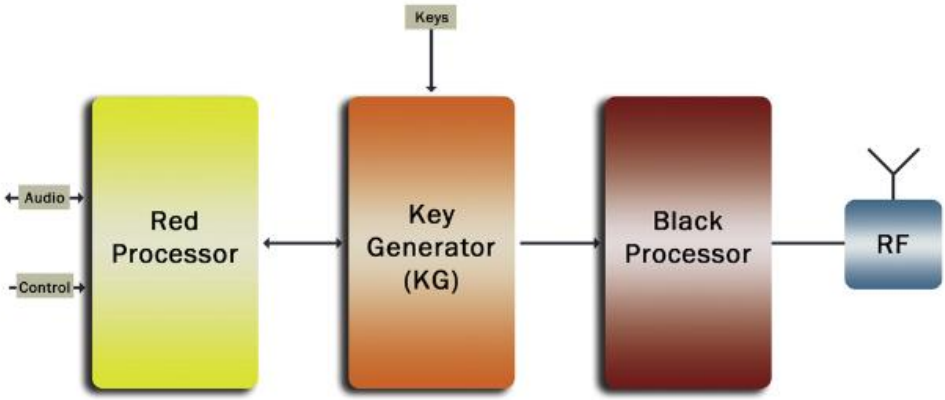
Kasus ini menjelaskan jaringan radio suara genggam nirkabel yang aman dan berbagai opsi untuk mengelola kuncinya. Seperti yang ditunjukkan pada Gambar 4.34, setiap radio dilengkapi dengan subsistem kriptografi tertanam untuk mengamankan lalu lintas suara. Jaringan ini dicirikan oleh komunikasi nirkabel berdaya rendah melalui sejumlah titik akhir (radio) yang relatif kecil, yaitu puluhan atau ratusan, bukan puluhan ribu. Jaringan nirkabel dapat digunakan sebagai bagian dari misi militer rahasia untuk memburu teroris, serangan kejahatan, atau operasi pribadi yang memerlukan jaringan nirkabel yang aman.



Gambar 4.34 Jaringan radio push-to-talk nirkabel yang aman.

Untuk aplikasi militer, setidaknya beberapa algoritma kriptografi yang digunakan dalam subsistem kriptografi kemungkinan besar diklasifikasikan. Untuk aplikasi lain, subsistem kriptografi kemungkinan besar menggunakan AES yang dikonfigurasi sebagai sandi keystream dalam mode OFB. Tergantung pada bandwidth saluran frekuensi radio (RF) (misalnya, pita sempit atau pita lebar), algoritme kompresi suara dapat berupa standar pemerintah seperti Mixed Exitation Linear Prediction (MELP) atau standar komersial seperti

G.729.[26] Untuk saluran pita sempit, kami akan menggunakan tidak menggunakan modulasi kode pulsa (PCM) untuk analog-konversi ke digital dan kompresi suara. Diagram arsitektur umum radio ditunjukkan pada Gambar 4.35.



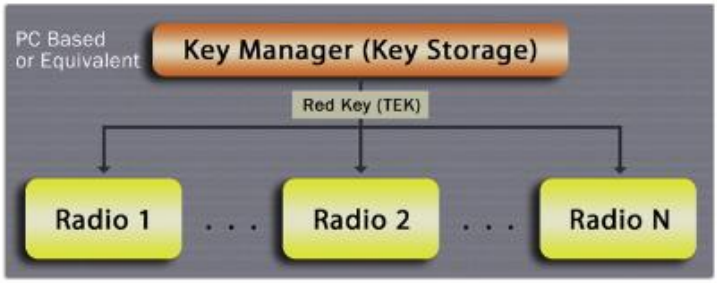
Gambar 4.35 Diagram radio militer aman yang digeneralisasi.

[26] International Telecommunications Union. Telecommunication Standardization Sector, G.729: Coding of Speed at 8 kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear Prediction (CS-ACELP) (January 2007).

Pengoperasian jaringan yang aman dimulai dengan pengguna menekan tombol push-to-talk (PTT) radio. Inisiasi PTT menginterupsi jaringan dan memperingatkan semua pelanggan jaringan bahwa komunikasi suara akan segera dilakukan. Misalnya, pada inisiasi PTT, radio mengirimkan pola teks biasa yang bergantian antara satu dan nol pada saluran, memperingatkan semua penerima lain bahwa sinkronisasi kriptografi akan segera dimulai. Untuk saluran burst atau noise, pesan sinkronisasi kripto yang dikirim oleh pemrakarsa PTT menggunakan skema pengkodean kesalahan yang tepat untuk memastikan penerimaan IV yang sempurna oleh semua penerima. Pelepasan tombol PTT menghasilkan pesan akhir terenkripsi (EOM), memperingatkan semua penerima bahwa semua lalu lintas harus dihentikan hingga sinkronisasi kriptografi baru dikirimkan. Setelah saluran dilepaskan, pengguna lain dapat memulai komunikasi suara menggunakan PTT-nya. Ada banyak pilihan yang tersedia untuk manajemen kunci jaringan PTT dan radionya; mereka dijelaskan selanjutnya.

Opsi Satu: Memuat TEK Merah Biasa ke Setiap Radio di Fasilitas Beban Utama

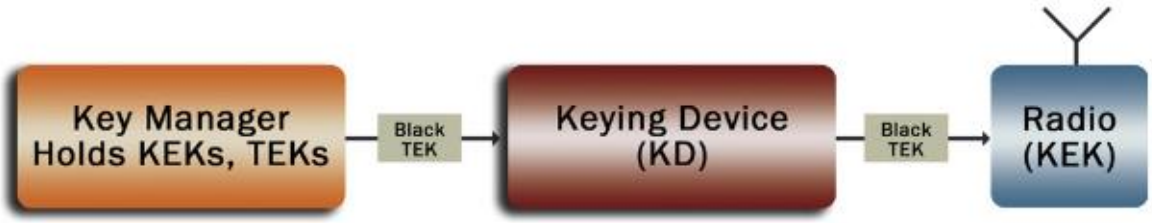
Pengguna tidak boleh memuat TEK merah mereka sendiri. Kebijakan ini menghindari ancaman orang dalam dari pengguna akhir yang membocorkan TEK merah miliknya. Yang terbaik adalah memuat TEK merah di fasilitas pemuatan kunci tepercaya (KLF), seperti yang ditunjukkan pada Gambar 4.36.



Gambar 4.36 Fasilitas beban utama.

Seperti yang ditunjukkan dalam diagram ini, metode umum untuk menyediakan sekumpulan radio adalah dengan menyambungkan setiap port beban utamanya ke jaringan instalasi yang juga menyertakan server kunci. Server kunci mungkin berupa PC sederhana. Jaringan instalasi mungkin berupa jaringan Ethernet lokal yang digunakan untuk proses penyediaan kunci. Merupakan kebijakan yang baik bagi radio untuk memiliki port fisik berbeda yang digunakan secara khusus untuk penyediaan kunci. Hal ini mengurangi kemungkinan terjadinya koneksi yang tidak tepat dengan perangkat lunak pemuatan kunci internal radio. Setelah TEK merah ini dimuat melalui kabel instalasi, TEK tersebut digunakan sebagai TEK umum dari jaringan taktis yang dikerahkan dan tidak diubah selama misi. Di akhir misi, ketika radio dikembalikan, TEK merah baru dimuat di KLF untuk misi berikutnya.

Opsi ini layak dan mudah diterapkan. Namun, jika salah satu radio hilang, disusupi, atau ditangkap di lingkungan yang tidak bersahabat, keamanan komunikasi akan hilang. Satu-satunya cara untuk membangun kembali komunikasi yang aman adalah dengan membatalkan misi tersebut, mengembalikan radio ke KLF untuk dimuat ulang.



Gambar 4.37 TEK hitam dimuat ke radio.

Opsi Kedua: Memuat Kunci Hitam di Bidang

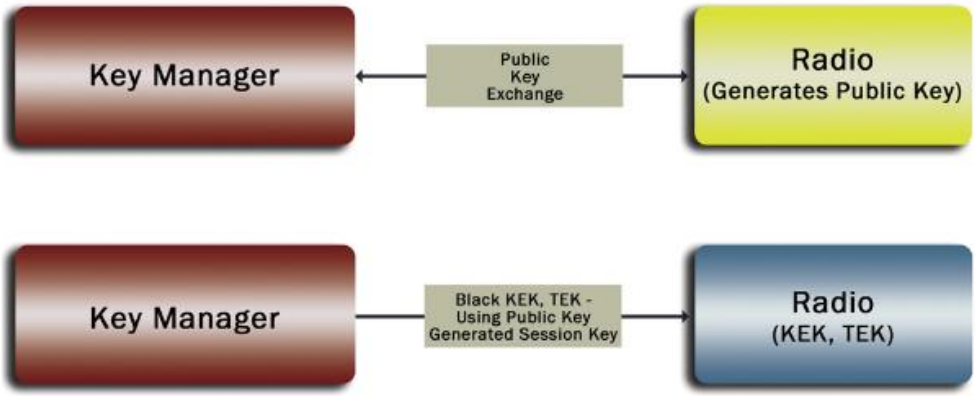
Alih-alih TEK, KEK merah per-radio unik dimuat ke setiap radio di KLF. Selain itu, TEK jaringan umum dienkripsi dengan masing-masing KEK unik dan dimuat ke dalam perangkat kunci portabel (lihat Gambar 4.37), yang dikeluarkan untuk setiap pengguna untuk dibawa ke lapangan bersama dengan radio. Pengguna kemudian memuat TEK hitam ke radio menggunakan perangkat kunci. Terakhir, radio mendekripsi TEK menggunakan KEK uniknya, dan radio kini tersedia untuk komunikasi jaringan.

Opsi ini memungkinkan pengguna memuat ulang TEK baru secara manual ke radionya. Setiap perangkat penguncian dapat disediakan di KLF dengan sejumlah besar TEK hitam, sehingga memungkinkan penguncian ulang dinamis di lapangan untuk jangka waktu yang lama. TEK jaringan umum dienkripsi dalam KEK unik menggunakan skema buku kode elektronik yang disebut key wrap, yang dijelaskan sebelumnya dalam bab ini. Semua KEK dihasilkan dan dimuat di KLF dan tidak pernah diekspos atau digunakan di lapangan.

Metode lain untuk menyediakan perangkat kunci adalah dengan memuat salinan KEK per radio, bukan TEK hitam. Kemudian pemilik perangkat pengunci (petugas penyediaan kunci lapangan) menggunakan antarmuka tepercaya, seperti tombol, untuk secara dinamis menghasilkan dan mengenkripsi TEK yang dapat dipasang ke semua radio. Perangkat kunci harus memerlukan otentikasi multi-faktor yang kuat, seperti kata sandi dan penyisipan kartu pintar, dari petugas penyedia untuk mengaktifkan fungsi pemuatan. Pendekatan ini memungkinkan sekumpulan radio yang dikerahkan untuk dikunci ulang dalam jumlah tak terhingga, selama perangkat kuncinya tetap terlindungi. Karena berisi kunci merah untuk semua radio, perangkat kunci menjadi satu titik kegagalan di mana kompromi kunci internal akan mengekspos semua komunikasi. Perlindungan terhadap kerusakan pada perangkat pengunci ditunjukkan bila perangkat tersebut berisi kunci berwarna merah.

Opsi Tiga: Pemuatan Kunci Menggunakan Kriptografi Kunci Publik

Opsi ketiga mirip dengan opsi satu, kecuali bahwa alih-alih memuat TEK merah ke radio di KLF, TEK dimuat melalui jaringan instalasi KLF dengan terlebih dahulu membuat kunci sesi melalui pertukaran kunci antara server kunci master dan masing-masing server. radio, secara bergantian. Kunci sesi ini digunakan untuk mengirimkan TEK ke setiap radio, seperti yang ditunjukkan pada Gambar 4.38. Untuk opsi ketiga, tidak ada tombol merah yang ditampilkan, bahkan di KLF. Hal ini melindungi terhadap beberapa ancaman orang dalam dalam jaringan instalasi KLF. Penggunaan pertukaran kunci publik mengasumsikan bahwa semua radio telah disediakan sebelumnya, di pabrik, dengan kunci publik statis pribadi dan sertifikat akar untuk jaringan instalasi sebelum ditransfer ke KLF.



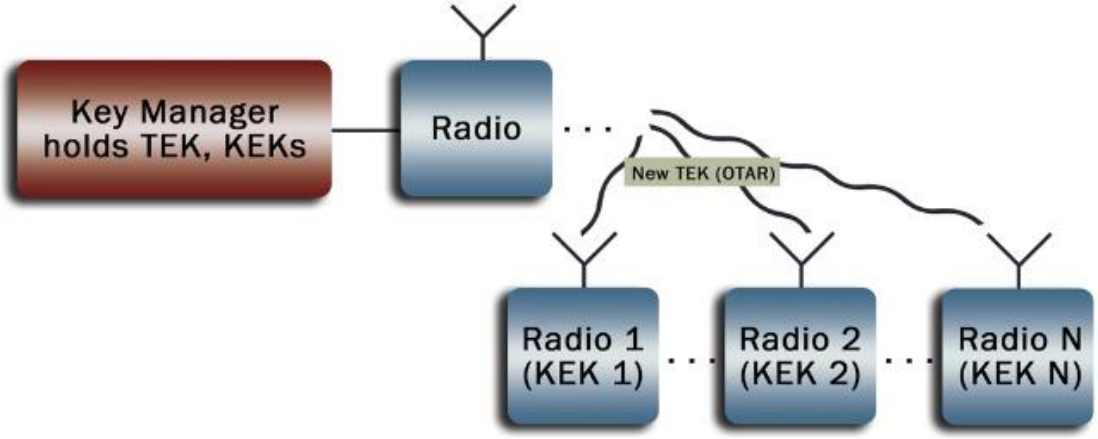
Gambar 4.38 Pemuatan kunci hitam ke radio menggunakan kriptografi kunci publik.

Untuk menyederhanakan proses instalasi, server kunci dan radio yang terhubung pada jaringan penyediaan KLF dapat memperoleh TEK jaringan dari data rahasia bersama yang dibuat selama setiap pertukaran kunci server-radio. Faktanya, beberapa TEK, untuk menjalankan banyak misi, dapat dibuat dan diinstal menggunakan proses ini.

Tentu saja, pertukaran kunci publik dapat diterapkan pada opsi kedua, misalnya, untuk menyediakan perangkat kunci dan radio dengan KEK. Kemudian alat pengunci tersebut dapat digunakan untuk memasang TEK secara berkala sesuai kebutuhan pada radio-radio yang ada di lapangan.

Opsi Empat: Kemampuan Rekey Over-the-Air

Dalam skenario keempat ini, opsi dua atau tiga pertama kali digunakan untuk memuat KEK dan TEK awal ke masing-masing radio di KLF. Kemudian, sesuai kebutuhan di lapangan, server kunci master mengirimkan TEK umum baru ke setiap radio dari jarak jauh, seperti yang ditunjukkan pada Gambar 4.39.



Gambar 4.39 Rekey over-the-air (OTAR) dari TEK baru.

Server kunci berinteraksi dengan radio untuk membuat sesi dan mengirim TEK baru dari jarak jauh. Tentu saja, pendekatan ini hanya berfungsi jika radio berada dalam jangkauan komunikasi server kunci. Server kunci pertama-tama membuat sambungan menggunakan TEK awal dan PTT untuk sinkronisasi. Setelah sesi dibuat, server kunci menggunakan protokol rekey berikut:

1. Server kunci mengirimkan pesan peringatan rekey terenkripsi, memberitahukan semua radio bahwa rekey akan segera dilakukan.
2. Setiap radio meninggalkan status lalu lintas dan menunggu pemuatan kunci dari server.
3. Server kunci membuat sinkronisasi kriptografi melalui pertukaran kunci publik dengan masing-masing radio dan kemudian mengirimkan TEK umum yang baru.
4. Urutan kunci ulang ini diikuti untuk semua radio sampai semuanya menerima TEK baru.

Proses rekey ini digunakan untuk rekey periodik atau supersession, perbedaannya adalah untuk supersession, server kunci hanya akan rekey radio yang tidak dikompromikan. Kunci server menghancurkan KEK unit yang disusupi. Komunikasi yang aman berlanjut setelah radio yang disusupi secara logis dihilangkan dari jaringan melalui proses kunci ulang ini. Bahkan jika penyerang berhasil memulihkan TEK dan KEK dari radio yang disusupi, penyerang tidak akan pernah dapat bergabung dengan jaringan, karena semua radio telah dikunci ulang dengan TEK baru.

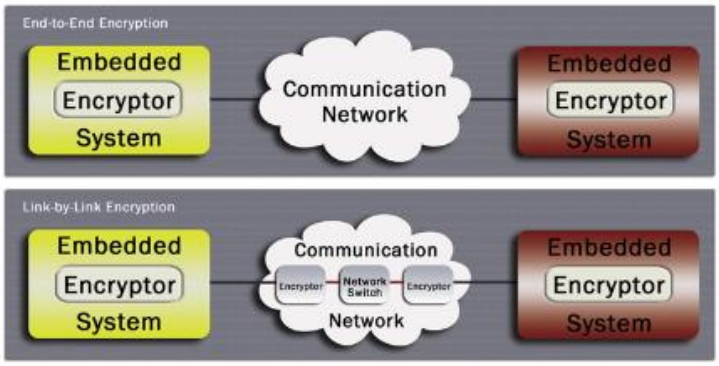
Jika komunikasi OTAR dimungkinkan dalam jaringan taktis yang diterapkan, pendekatan ini efisien namun kuat. Namun, karena dilengkapi dengan kemampuan untuk mengirimkan materi kunci penting melalui jaringan nirkabel, server kunci harus benar-benar dapat dipercaya. Selain menjalankan perangkat lunak yang dapat dipercaya, server kunci harus dilindungi secara fisik. Saat tidak digunakan, atau ditinggalkan, server kunci harus tidak tersedia untuk ekstraksi kunci baik secara terbuka maupun terselubung.

Salah satu cara untuk melindungi akses ke server kunci tanpa pengawasan adalah dengan menguncinya menggunakan token, seperti kartu pintar. Token ini berisi setengah kunci penyimpanan yang digunakan untuk mengenkripsi semua KEK di memori non-volatil server kunci. Separuh lainnya dari kunci penyimpanan terpisah terdapat dalam subsistem kriptografi server kunci. Dalam mode terkunci, semua kunci dienkripsi, dan bahkan jika komputer server kunci dicuri, musuh tidak akan dapat memulihkan kunci apa pun. Hanya ketika server kunci tidak terkunci barulah pemisahan tersebut dihubungkan kembali untuk memulihkan kunci penyimpanan dan membuat kunci radio terenkripsi tersedia untuk dimuat.

Studi Kasus: Komunikasi Telepon Secara Umum

Studi kasus ini menjelaskan cara mengamankan komunikasi suara dan data yang sensitif untuk semua jenis peralatan yang dioperasikan manusia, seperti telepon seluler. Solusi yang dibahas bersifat scalable dan dapat memberikan keamanan end-to-end untuk jaringan yang sangat besar, pada skala Internet. Kami mengeksplorasi beberapa opsi manajemen kunci umum yang tersedia bagi perancang sistem dan membandingkan setiap opsi dalam hal ketahanan keamanan.

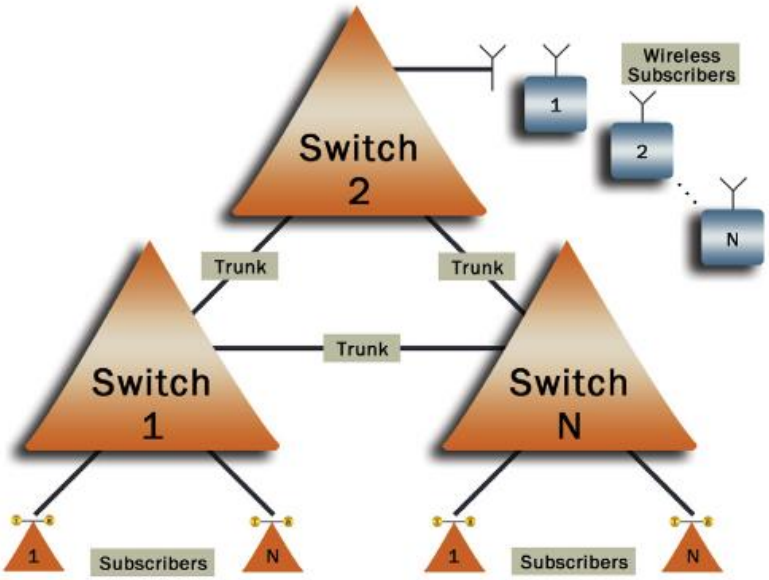
Sistem suara/data terdiri dari saklar-saklar yang dihubungkan satu sama lain melalui jalur utama. Instrumen akhir dapat mengambil berbagai faktor bentuk dan antarmuka termasuk layanan telepon lama biasa (POTS) atau telepon digital dan nirkabel. Kriptografi berada di dalam atau di dekat instrumen akhir, menyediakan apa yang umumnya disebut enkripsi ujung ke ujung dan menghindari enkripsi link-to-link (lihat Gambar 4.40). Enkripsi end-to-end menghindari kompromi “redebridge” yang berpotensi terjadi pada persimpangan antara segmen terenkripsi dalam jaringan multi-link.



Gambar 4.40 Perbandingan enkripsi ujung ke ujung dan tautan ke tautan.

Untuk kasus umum kami, kami tidak melarang penggunaan media jaringan fisik apa pun. Jaringan switching dapat berupa analog (misalnya POTS), digital (misalnya ISDN), Internet (misalnya Voice-over-IP), RF nirkabel, atau kombinasi keduanya. Kriptografi tertanam yang berada pada atau dekat instrumen akhir hanya diterapkan pada lapisan aplikasi model OSI. Suara/data dienkripsi pada instrumen akhir dan melintasi jaringan komunikasi yang tidak tepercaya dan berpotensi menimbulkan permusuhan, yang ketersediaannya tidak dapat dijamin. Namun kami menjelaskan pendekatan mitigasi yang meningkatkan ketersediaan jaringan dengan menggunakan teknik keamanan transmisi. Kami juga menjelaskan teknik-teknik yang melindungi lalu lintas suara/data serta sinyal (indikasi panggilan dan nomor telepon yang dipanggil), kadang-kadang disebut sebagai panggilan aman.

Seperti yang ditunjukkan pada diagram sistem pada Gambar 4.41, instrumen akhir dihubungkan ke sakelar lokal. Panggilan yang dimulai oleh pengguna akhir disalurkan secara lokal atau melalui trunk ke switch jarak jauh dan dihubungkan point-to-point ke pihak yang dipanggil. Untuk studi kasus ini, kami mencakup semua jenis perangkat komunikasi, mulai dari telepon aman yang dikembangkan dari awal hingga telepon kabel atau nirkabel yang tidak dimodifikasi, komersial siap pakai (COTS), dengan kriptografi sebagai aplikasinya.

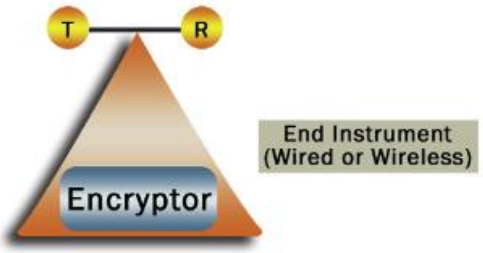


Gambar 4.41 Diagram sistem jaringan telepon.

Opsi Penyematan Subsistem Kriptografi

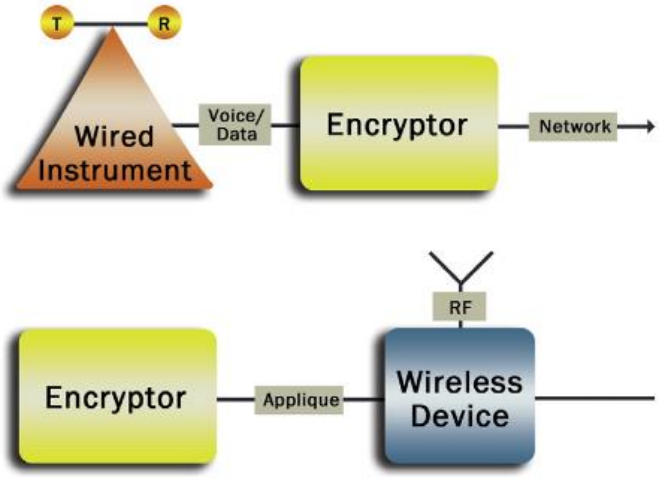
Ada banyak opsi yang tersedia untuk penyematan subsistem kriptografi; mereka dijelaskan selanjutnya.

Opsi Satu: Subsistem Kriptografi yang Tertanam di dalam Instrumen Akhir Opsi satu ditunjukkan pada Gambar 4.42.



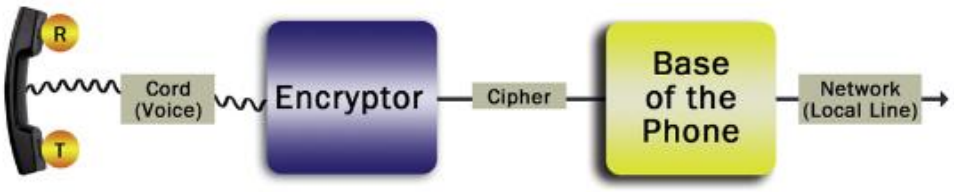
Gambar 4.42 Kriptografi Tertanam dalam Instrumen.

Opsi Kedua: Bump-on-the-Line (Terpasang pada Instrumen Akhir) Opsi dua menggunakan telepon yang tidak dimodifikasi, dan elemen kriptografinya adalah aplikasi yang terhubung ke output telepon, seperti yang ditunjukkan pada Gambar 4.43.



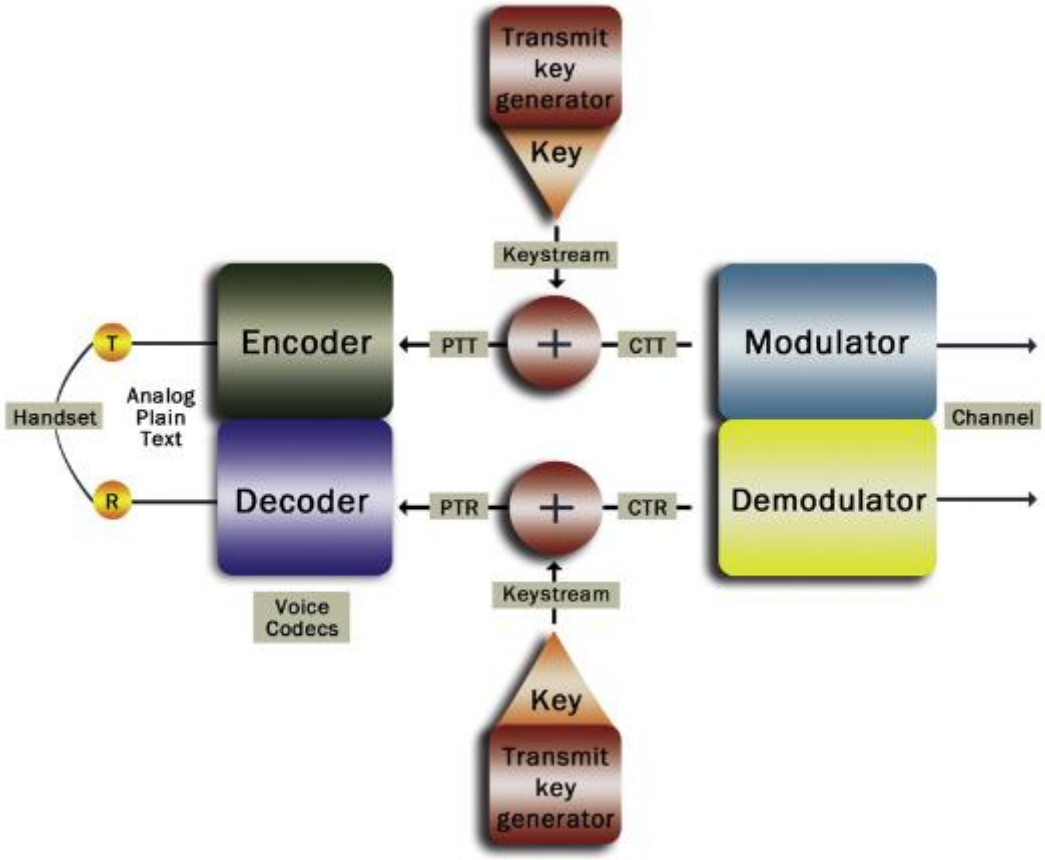
Gambar 4.43 Bump-on-the-line (melekat pada instrumen).

Opsi Tiga: Bump-on-the-Handset Mirip dengan opsi dua, pada opsi tiga, unit kriptografi (enkripsi) dihubungkan antara handset dan dasar instrumen (lihat Gambar 4.44).



Gambar 4.44 Benjolan di ponsel.

Dengan semua opsi penyematan ini, kami dapat mengatakan bahwa keamanan suara/data telepon dapat diterapkan tanpa penggantian semua telepon dan instrumen lama. Diagram blok umum dari unit kriptografi ditunjukkan pada Gambar 4.45.



Gambar 4.45 Diagram blok unit kriptografi tertanam (generator kunci dupleks penuh).

Kompresi suara, bergantung pada klasifikasi lalu lintas, dapat berupa vocoder (codec) tingkat rendah standar yang ditentukan dalam G.729, vocoder yang ditentukan pemerintah seperti MELP, atau vocoder lainnya. Unit kriptografi dapat menyimpan beberapa (atau banyak) vocoder, dan setelah sinkronisasi kriptografi melalui protokol yang ada, menentukan vocoder umum yang terdapat di kedua instrumen akhir. Perhatikan bahwa pengkodean dan penguraian kode secara historis telah dimasukkan ke dalam unit kriptografi meskipun fungsi-fungsi ini tidak terlibat dalam kriptografi. Hal ini dilakukan terutama untuk kinerja; selain itu, dalam beberapa kasus, mungkin ada keinginan untuk merahasiakan codec itu sendiri.

Untuk sistem suara/data aman kami yang umum, panggilan umum dilakukan sebagai berikut:

1. Pihak yang menelepon keluar dari panggilan, memperoleh nada panggil dari saklar, dan memasukkan nomor telepon pihak yang dipanggil.
2. Saklar menerima nomor telepon pihak yang dipanggil dan “membunyikan” pihak yang dipanggil jika bersifat lokal atau mengarahkan nomor tersebut ke saklar jarak jauh yang kemudian “membunyikan” pihak yang dipanggil.
3. Ketika pihak yang dipanggil menjawab, infrastruktur switch membentuk koneksi end-to-end antara pihak yang memanggil dan pihak yang dipanggil.
4. Lalu lintas suara/data yang aman dari ujung ke ujung dibuat oleh instrumen akhir.

Opsi Manajemen Kunci

Setelah panggilan end-to-end dibuat, suara/data terenkripsi dipertukarkan, asalkan instrumen akhir memiliki kunci sesi rahasia bersama. Tentu saja, ada opsi untuk menetapkan kunci sesi ini.

Opsi Satu: Memuat TEK Umum ke Semua Instrumen Opsi pertama adalah opsi yang umum, berbiaya rendah, dan sederhana. Namun saat ini, sudah jelas bahwa opsi ini tidaklah kuat. Risiko terjadinya kompromi kunci untuk kunci yang dibagikan secara luas relatif tinggi. Sekali kuncinya disusupi, seluruh sistem komunikasi pun disusupi. Seperti yang dijelaskan dalam studi kasus sebelumnya, opsi kunci yang dibagikan sebelumnya berpotensi digunakan untuk sejumlah kecil pelanggan dalam misi untuk periode kripto terbatas; jika tidak, hal ini tidak disarankan.

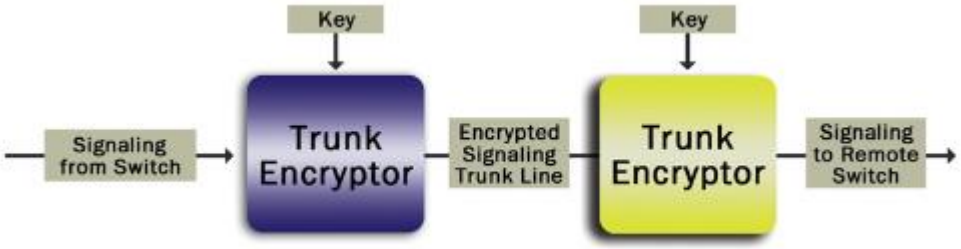
Opsi Dua: Gunakan Pertukaran Kunci Publik untuk Mendapatkan Kunci Sesi Pada opsi dua, setelah koneksi end-to-end dibuat, instrumen melakukan pertukaran kunci publik (misalnya, menggunakan Diffie-Hellman) untuk membuat kunci sesi rahasia untuk enkripsi dan dekripsi lalu lintas selanjutnya. Kunci sesi hanya umum di antara dua instrumen akhir dan dimusnahkan pada akhir sesi. Periode kripto kunci sesi sama pendeknya dengan panggilan telepon. Kunci sesi baru dibuat pada panggilan berikutnya. Dengan demikian, sistem tidak rentan terhadap serangan ruang kunci yang menyeluruh, karena kuncinya diubah setiap sesi. Dengan asumsi titik akhir menggunakan algoritma dan panjang kunci yang direkomendasikan standar, penyadap aktif tidak dapat memecahkan kunci sesi bahkan dengan kekuatan komputasi besar yang dimilikinya.

Kunci publik pada awalnya dimuat ke dalam instrumen akhir oleh server kunci tepercaya atau otoritas sertifikasi (CA). Kunci publik dikemas dalam sertifikat yang ditandatangani oleh CA. Hanya kunci publik tersertifikasi yang dihormati oleh pihak yang memanggil dan dipanggil. Jika instrumen akhir disusupi, CA akan mencabut sertifikat kunci publik instrumen tersebut dan menempatkannya dalam daftar pencabutan yang dikirimkan ke instrumen akhir. Selama pengaturan panggilan, instrumen akhir tidak menerima sertifikat apa pun di daftar pencabutan. CA dapat mendistribusikan daftar pencabutan ke instrumen akhir secara berkala. Alternatifnya, CA dapat mengirimkan daftar pencabutan ke serangkaian instrumen terdekat, dan instrumen tersebut dapat menyebarkan daftar tersebut ke terminal lain dalam rantai distribusi.

Meskipun opsi manajemen kunci satu dan dua memberikan keamanan end-to-end, metode pertukaran kunci adalah solusi yang lebih baik dan dapat diperluas ke sejumlah besar pelanggan. Kedua solusi tersebut merupakan solusi point-to-point. Solusi point-to-multipoint memerlukan TEK umum untuk jaringan multipoint. Baik opsi satu maupun opsi dua tidak dapat melakukan hal ini, karena kunci sesi hanya diberikan kepada dua pihak. TEK umum memerlukan server kunci, serupa dengan studi kasus push-to-talk sebelumnya, untuk distribusi TEK umum menggunakan KEK per instrumen yang telah ditetapkan sebelumnya.

Kedua opsi manajemen utama ini hanya memberikan perlindungan terhadap informasi suara/data; mereka tidak melindungi pengalamatan informasi. Nomor pihak yang menelepon dan dipanggil dihubungi dengan jelas. Seorang penyadap yang memonitor jalur komunikasi tidak dapat mendekripsi lalu lintas tetapi mampu menentukan volume lalu lintas serta siapa

yang menelepon satu sama lain. Untuk sistem yang memerlukan penanganan kerahasiaan, diperlukan pendekatan yang dimodifikasi.



Gambar 4.46 Perlindungan bagasi menggunakan enkripsi.

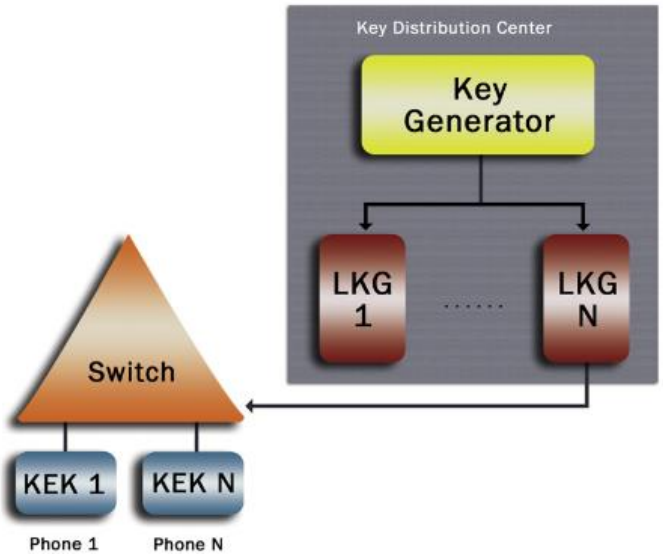
Mengatasi Perlindungan

Opsi Satu: Enkripsi Massal antar Titik Akhir Batang Batang yang membawa banyak nomor telepon pelanggan di antara saklar pusat dapat dilindungi dengan enkripsi link-to-link pada jalur utama, seperti ditunjukkan pada Gambar 4.46.

Solusi ini melindungi alamat instrumen dalam segmen trunk namun tidak melindungi alamat pelanggan switch lokal. Meskipun demikian, pendekatan ini dapat dilakukan dan memberikan keamanan transmisi yang signifikan di seluruh sistem. Seorang penyadap yang aktif, memonitor jalur utama, mengamati aliran acak yang berkesinambungan (sandi dari tautan), apakah ada sinyal atau aktivitas yang terjadi di antara jalur utama atau tidak. Mengacaukan trunk sangatlah sulit, karena memerlukan pengacau kebisingan pita lebar yang sangat kuat.

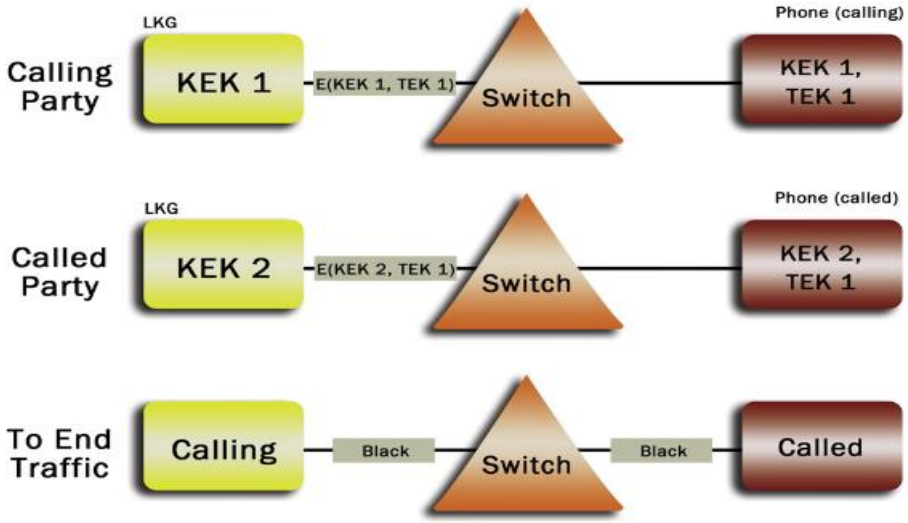
Opsi Kedua: Perlindungan Alamat Lokal Untuk menghilangkan kerentanan opsi satu, diperlukan perlindungan alamat per pelanggan. Ini adalah masalah yang sulit. Salah satu solusi yang mahal dan rumit adalah dengan menyediakan pusat distribusi kunci (KDC) untuk setiap pusat switching. KDC akan menyediakan generator kunci lokal yang menghubungkan ke pelanggan selama nada panggil atau nada dering, mengamankan tautan selama fase ini.

KDC, seperti yang ditunjukkan pada Gambar 4.47, terdiri dari pengontrol kunci, tempat semua kunci simetris dihasilkan dan disimpan, dan satu set generator kunci loop (LKG), yang dapat dihubungkan oleh sakelar selama fase panggilan/dering. Setiap pelanggan sudah diisi dengan KEK unik yang dihasilkan di KDC.



Gambar 4.47 KDC untuk panggilan aman.

Pembentukan panggilan aman akan mengikuti protokol yang ditunjukkan pada Gambar 4.48. Ketika saklar mendeteksi off-hook dari pihak pemanggil, dan sebelum terminal menerima nada sambung, saklar menghubungkan LKG ke pihak pemanggil setelah LKG menerima KEK unik pelanggan. Perhatikan bahwa LKG dilindungi secara fisik di dalam KDC, bersama dengan generator kunci utama dan penyimpanan kunci. KDC juga menghasilkan TEK per sesi dan mengirimkannya ke LKG penelepon dan penerima panggilan. LKG dan pihak pemanggil melakukan sinkronisasi kriptografi menggunakan KEK unik, dan LKG mengirimkan TEK, yang dienkripsi oleh KEK, ke pihak pemanggil.



Gambar 4.48 Pembentukan panggilan yang aman.

Setelah menerima TEK, pihak pemanggil memulai sinkronisasi kriptografi baru dengan LKG menggunakan TEK. Saklar kemudian mengirimkan nada sambung melalui LKG. Penelepon secara bergiliran mengirimkan nomor pihak yang dipanggil dengan aman. Setelah saklar mendapatkan digit dial, LKG dikeluarkan dari koneksi dan tersedia untuk panggilan lainnya.

Saklar akan membunyikan pihak yang dipanggil, dan ketika pelanggan yang dipanggil keluar, LKG yang ditugaskan ke pihak yang dipanggil terlebih dahulu diberikan TEK (kunci sesi) yang sama yang awalnya dihasilkan oleh KDC untuk pihak yang memanggil. Setelah TEK dikirim ke pihak yang dipanggil dengan aman melalui LKG, LKG tersebut dibatalkan dan koneksi end-to-end dibuat antara pihak yang memanggil dan dipanggil. Pihak pemanggil memulai lalu lintas, diamankan dengan TEK, dengan mengirimkan urutan sinkronisasi kriptografi.

Kunci sesi acak baru dihasilkan untuk setiap panggilan. Setelah setiap panggilan, kunci sesi dimusnahkan di KDC dan di instrumen akhir. Serangan kunci yang menyeluruh tidak mungkin dilakukan. Solusi ini aman namun mahal dan tidak dapat diskalakan dengan baik untuk jutaan pelanggan, karena setiap KDC harus menyimpan KEK semua pelanggannya. Antarmuka antara KDC dan switch memiliki komponen berwarna merah yaitu teks biasa ke dan dari LKG. Oleh karena itu, setiap KDC harus ditempatkan dengan saklar di fasilitas yang aman. Sistem kriptografi saat ini, secara umum, tidak melindungi informasi persinyalan (alamat, nomor telepon, protokol persinyalan). Internet tidak melindungi pengalamatan informasi. Akibatnya, penyadap yang aktif dan cerdas dapat menimbulkan malapetaka pada kerentanan ini.

4.15 SERTIFIKASI KRIPTOGRAFI

Karena pentingnya hampir setiap aspek teknologi keamanan, implementasi kriptografi adalah salah satu komponen yang paling banyak diatur dalam dunia komputasi. Meskipun bagian ini tidak berupaya mencakup semua sertifikasi yang terkait dengan kriptografi (misalnya, industri keuangan dan pembayaran melakukan beberapa jenis persetujuan khusus perangkat), kami secara singkat membahas sertifikasi FIPS 140 yang ada di mana-mana serta sertifikasi dan persetujuan kriptografi pemerintah A.S. proses. Kami membahas kapan pengembang mungkin terpengaruh oleh sertifikasi ini, memberikan gambaran umum, dan mengarahkan pembaca ke sumber informasi dan panduan berguna lainnya.

Sertifikasi FIPS 140-2 - Standar FIPS 140-2

Publikasi FIPS 140-2 menetapkan persyaratan keamanan (lihat ringkasan di Tabel 4.6) untuk implementasi perangkat keras dan perangkat lunak kriptografi.[27] Sertifikasi FIPS 140-2 bertindak sebagai tolok ukur kualitas komersial tingkat dasar dan diamanatkan oleh organisasi tertentu, seperti pemerintah A.S. untuk semua komunikasinya yang sensitif namun tidak rahasia (sertifikasi sistem komunikasi rahasia ditangani oleh NSA). Karena banyak sistem tertanam menargetkan pelanggan akhir pemerintah, penggunaan perangkat lunak dan/atau perangkat keras bersertifikasi FIPS 140 sering kali merupakan persyaratan penting bagi perancang sistem tertanam. Selain itu, banyak bidang komersial, seperti komunitas keuangan, telah mengadopsi FIPS 140-2 sebagai persyaratan.

Tabel 4.6: Ikhtisar Persyaratan FIPS 140.

Kategori persyaratan	Keterangan
Spesifikasi modul kriptografi	Dokumentasi menjelaskan batas kiptografi, algoritma yang digunakan, metode penetapan kunci, perangkat keras/arsitektur perangkat lunak/ firmware

[27] Federal Information Processing Standards Publication 140-2: Security Requirements for Cryptographic Modules, National Institute of Standards and Technology (May 2001).

Antarmuka modul kriptografi	Dokumentasi menjelaskan semua antarmuka kriptografi jalur data
Peran, otentikasi, dan layanan	Dokumentasi peranp sistem kriptografi (perangkat lunak dan pengguna); peran, identitas, dan otentikasi multi-faktor persyaratan berdasarkan tingkat keamanan
Lingkungan operasional keamanan perangkat lunak/firmware	Persyaratan perlindungan integritas berdasarkan tingkat keamanan perlindungan dan kontrol akses batas kriptografi dalam sistem
Keamanan fisik	Bukti, deteksi, dan resistensi kerusakan yang invasif dan non-invasif, bergantung pada tingkat keamanan
Manajemen parameter keamanan	Nomor acak dan pembangkitan kunci, zeroisasi, penanganan, penyimpanan, I/O
Tes mandiri	Perlindungan integritas pra-operasional dan runtime dan tes validasi algoritmik
Jaminan siklus hidup	Manajemen konfigurasi, spesifikasi, desain, dan ketelitian pengujian tergantung pada tingkat keamanan.

Dalam beberapa kasus, menggunakan perpustakaan perangkat lunak bersertifikasi FIPS 140 atau kartu jaringan tambahan dalam suatu desain mungkin tidak cukup. Kebijakan pemerintah menyatakan bahwa setiap produk yang menerapkan kriptografi harus bersertifikat FIPS 140. Oleh karena itu, peralatan seperti alat keamanan jaringan harus memiliki sertifikasi FIPS 140.

Penting untuk dicatat bahwa FIPS 140 terutama menetapkan persyaratan fungsional dan sangat sedikit dalam hal persyaratan jaminan; oleh karena itu, validasi diarahkan pada pengujian fungsional kotak hitam. Persyaratan jaminan yang lebih ketat, seperti pengujian penetrasi tingkat kode sumber, tidak diwajibkan baik oleh standar maupun program validasi NIST.

Contoh menonjol dari kurangnya ketelitian jaminan ditemukan dalam persyaratan fungsional berikut (bagian 4.6.1 dari standar FIPS 140-2):

Hanya untuk Tingkat Keamanan 1, modul kriptografi harus mencegah akses oleh proses lain ke kunci pribadi dan rahasia teks biasa, CSP, dan nilai pembangkitan kunci perantara selama modul kriptografi dijalankan/beroperasi.[28]

Persyaratan ini tampaknya menyiratkan bahwa implementasi perangkat lunak kriptografi harus dibuat sebagai proses terpisah, independen dan dilindungi memori dari proses lain dalam sistem, menggunakan API berbasis pesan untuk menjalankan layanan kriptografi. Namun secara praktis semua modul perangkat lunak yang divalidasi FIPS 140-2 diimplementasikan sebagai perpustakaan tautan dinamis (DLL) yang, menurut definisi, berbagi ruang memori yang sama dengan aplikasi yang menggunakannya.

FIPS 140-2 menetapkan empat tingkat kualitatif, satu sampai empat, dengan persyaratan fungsional yang relatif meningkat. Level 1 mencakup persyaratan non-fisik, seperti validasi bahwa implementasi algoritma memenuhi spesifikasinya. Level 2 menambahkan persyaratan deteksi kerusakan. Level 3 memerlukan ketahanan terhadap

[28] Federal Information Processing Standards Publication 140-2: Security Requirements for Cryptographic Modules, (January 11, 1994).

kerusakan. Level 4 memperluas persyaratan perlindungan terhadap kerusakan namun juga menambahkan perlindungan terhadap anomali lingkungan seperti fluktuasi tegangan atau suhu. Namun FIPS 140-2 tidak secara eksplisit memerlukan perlindungan terhadap serangan canggih tertentu, seperti serangan berdasarkan DPA, timing, dan TEMPEST.

FIPS 140-3

FIPS 140-3 diharapkan menggantikan FIPS 140-2. Ratifikasi standar baru ini diharapkan paling lambat pada tahun 2012. Masa tenggang singkat akan menyusul dimana FIPS 140-2 akan tetap diterima oleh pemerintah AS. Perancang sistem tertanam yang berencana meluncurkan produk kriptografi harus mengetahui FIPS 140-3 dan berencana untuk mematuhi standar baru. Tempat yang baik untuk memulai adalah rancangan standar terkini yang dapat ditemukan di situs web rancangan publik NIST.[29]

Kepatuhan FIPS 140-3 akan berdampak terbatas pada perancang sistem tertanam yang menargetkan FIPS 140 Level 1 dan 2. Salah satu pendorong utama FIPS 140-3 adalah menambahkan persyaratan eksplisit untuk beberapa serangan canggih yang disebutkan di atas, yang dimulai pada Level 3 dan 4. Jenis serangan tertentu (termasuk DPA) disebutkan dalam apa yang saat ini disebut Lampiran F rancangan standar. [30]Namun, standar tersebut masih bersifat non-preskriptif sehubungan dengan cara mengatasi serangan-serangan ini. Program validasi kemungkinan besar akan bergantung pada laboratorium pengujian yang disetujui, seperti SAIC, untuk menerapkan vektor serangan yang diketahui dan memvalidasi pengoperasian tindakan pencegahan yang diketahui. Untuk pengembang sistem tertanam yang ingin mempelajari lebih lanjut tentang sertifikasi FIPS 140 (140-2 atau 140-3), sumber informasi yang baik serta pelatihan langsung disediakan oleh perusahaan seperti SAIC, yang memiliki bisnis konsultasi FIPS independen. (firewall) dari laboratorium pengujian pemerintah.

Pengembang peralatan keamanan yang dapat memperoleh manfaat dari nilai pemasaran sertifikasi FIPS 140 disarankan untuk mengikuti kelas pelatihan, yang disediakan oleh SAIC dan pakar sertifikasi FIPS lainnya, yang akan memperkenalkan desainer pada proses umum sertifikasi FIPS. Secara umum, persyaratan dokumentasi dan pengujian untuk sertifikasi FIPS, dengan asumsi bahwa produk telah memenuhi serangkaian persyaratan fungsional yang masuk akal dan tidak mengherankan pada tingkat keamanan FIPS mana pun, bukanlah hal yang ekstrim. Faktanya, perusahaan konsultan FIPS dapat menyediakan templat untuk kebijakan dan panduan pengguna yang akan membantu Anda mencapai tujuan tersebut.

Mungkin bagian yang paling menyakitkan dari proses sertifikasi FIPS adalah lamanya waktu yang dibutuhkan sembilan bulan hingga satu tahun atau terkadang lebih lama untuk mendapatkan sertifikasi setelah produk diserahkan untuk evaluasi. Namun, beberapa organisasi pemerintah mungkin menerima produk yang sedang dievaluasi. Ketika suatu organisasi bekerja sama dengan konsultan FIPS yang berpengalaman, kemungkinan suatu produk tidak lolos evaluasi ketika konsultan tersebut dianggap produk siap untuk diserahkan mendekati nol.

---

[29] Federal Information Processing Standards Publication 140-3: Security Requirements for Cryptographic Modules (Revised DRAFT) (September 11, 2009).

[30] Annex F: Non-Invasive Attack Methods for FIPS PUB 140-3 (Draft), Security Requirements for Cryptographic Modules, NIST Computer Security Division (September 10, 2009).

Sertifikasi NSA - Klasifikasi Produk Kriptografi

Informasi dapat dilindungi dengan berbagai cara, termasuk perlindungan fisik oleh penjaga bersenjata dan bangunan seperti gedung, brankas, atau brankas. Informasi, dalam keadaan diam atau dalam perjalanan, dilindungi dengan cara kriptografi biasanya diterapkan oleh perangkat keras dan/atau perangkat lunak elektronik, sesuai dengan spesifikasi yang dikembangkan oleh dua lembaga AS. Badan Keamanan Nasional (NSA) menetapkan persyaratan desain, dokumentasi, jaminan, dan sertifikasi untuk informasi rahasia atau sensitif tetapi tidak rahasia di AS. Institut Standar dan Teknologi Nasional (NIST) menetapkan persyaratan untuk informasi pribadi atau komersial.

Informasi pemerintah AS dilindungi melalui pengembangan elemen atau perangkat kriptografi yang didefinisikan sebagai

- Tipe 1: Elemen atau produk kriptografi yang melindungi informasi Sangat Rahasia (TS) atau Rahasia (S).
- Tipe 2: Elemen atau produk kriptografi yang melindungi informasi sensitif namun tidak rahasia (SBU).

Perangkat kriptografi Tipe 1 dan Tipe 2 menggunakan algoritma kriptografi Tipe 1 atau Tipe 2 yang secara historis telah ditentukan oleh NSA dan sering kali diklasifikasikan. Namun, seperti yang akan kita bahas lebih lanjut di bab ini, garis ini menjadi agak kabur sejak NSA meluncurkan program terbarunya untuk memungkinkan beberapa perangkat rahasia Tipe 1 menggunakan algoritma NIST berlapis.

Informasi yang tidak diklasifikasikan diatur oleh NIST dalam dua kategori tambahan:

- Tipe 3: Elemen kriptografi yang menggunakan algoritma tertentu NIST, termasuk algoritma simetris dan hash. AES, DES, triple DES, Skipjack, dan SHA adalah contoh algoritma yang ditentukan NIST.
- Tipe 4: Algoritme kriptografi yang terdaftar tetapi tidak ditentukan oleh NIST. Tipe 4 digunakan untuk merujuk pada algoritma yang dapat diekspor; namun, hal ini tidak lagi terjadi, dan karenanya Tipe 4 memiliki sedikit penerapan dalam sistem tertanam modern.

Elemen atau produk kriptografi yang ingin mendapatkan “segel persetujuan” NIST harus menggunakan algoritma Tipe 3 dan menjalani proses sertifikasi FIPS 140. Meskipun elemen kriptografi komersial tidak selalu memerlukan sertifikasi FIPS 140, elemen kriptografi yang dijual ke lembaga pemerintah AS memerlukan sertifikasi FIPS-140.

Perangkat kriptografi Tipe 1 dan Tipe 2 ditentukan, diuji, dan disertifikasi oleh NSA. Persyaratan fungsional dan keamanan untuk perangkat Tipe 1 dan Tipe 2 dijelaskan dalam serangkaian dokumen rahasia dan tidak rahasia. Dokumen penting yang mengatur desain, pengujian, dan verifikasi produk Tipe 1 adalah Unified INFOSEC Criteria (UIC), sebuah dokumen rahasia rahasia. Dokumen lain yang menjelaskan kriptografi dan protokol terkait diklasifikasikan Hanya Untuk Penggunaan Resmi (FOUO).

Spesifikasi Tipe 1 mencakup dua kategori:

- ❖ Sangat Rahasia (TS): Proses yang harus diikuti untuk mengamankan informasi TS tercakup dalam bagian Jaminan Tingkat Tinggi/Tinggi dari UIC.
- ❖ Rahasia (S): Proses yang harus diikuti untuk mengamankan informasi S tercakup dalam bagian Jaminan Tingkat Tinggi/Menengah UIC.

Pengembang sistem tertanam yang merancang perangkat Tipe 1 dapat memilih untuk mengembangkan satu perangkat Tipe 1 yang mencakup TS dan S, atau dua perangkat terpisah, satu untuk TS dan satu lagi untuk S. Perangkat yang dirancang dengan standar TS dapat mengamankan TS, S, atau informasi yang tidak rahasia. Perangkat yang dirancang dengan standar S dapat mengamankan informasi Rahasia atau tidak rahasia (Rahasia-dan-bawah atau SABI). Perangkat Tipe 3 dan Tipe 4 (bersertifikasi FIPS 140 atau terdaftar NIST) secara historis tidak diizinkan untuk mengamankan informasi rahasia Departemen Pertahanan AS (TS, S).

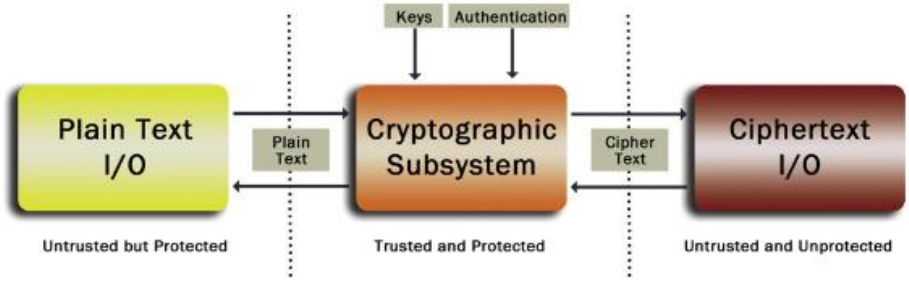
Meskipun perangkat Tipe 1 umumnya berisi algoritme kriptografi yang diklasifikasikan, NSA menyetujui penggunaan algoritme Tipe 3 (misalnya AES) untuk aplikasi Tipe 1. Salah satu contohnya adalah interoperabilitas sekutu di mana NSA menetapkan penggunaan rangkaian algoritme AES untuk digunakan pada perangkat Tipe 1 untuk digunakan dengan sekutu AS. Namun penting untuk dicatat bahwa meskipun NSA mengizinkan penggunaan rangkaian algoritma yang tidak terklasifikasi (AES) untuk informasi rahasia, perangkat tersebut masih harus memenuhi persyaratan Tipe 1 yang ditentukan NSA yang dinyatakan dalam UIC. Faktanya, perangkat Tipe 1 dapat berisi beberapa rangkaian algoritma, misalnya satu set algoritma rahasia untuk penggunaan khusus di AS dan satu set algoritma tidak rahasia untuk penggunaan gabungan.

Persyaratan Kriptografi untuk Perangkat Tipe 1

NSA menentukan desain Tipe 1 dan persyaratan jaminan perangkat kriptografi, menggunakan model umum yang ditunjukkan pada Gambar 4.49. NIST mengikuti proses serupa untuk sertifikasi FIPS 140.

Dalam terminologi NSA, elemen kriptografi sering disebut sebagai End Crypto Unit (ECU), berbeda dengan Electronic Control Unit atau Engine Control Unit di industri otomotif. ECU adalah elemen atau subsistem kriptografi yang dirancang, didokumentasikan, dan diuji sesuai dengan UIC. ECU adalah satu-satunya elemen "tepercaya" dalam model ini. Semua persyaratan keamanan dalam model ini berada pada batas keamanan yang digambarkan.

NSA UIC memerlukan implementasi ECU yang berdiri sendiri, unit atau kotak yang terpisah secara fisik untuk memenuhi persyaratan keamanan tingkat lanjut seperti anti-tamper atau anti-TEMPEST (ekstraksi informasi rahasia secara elektronik). Namun model ini tidak menghalangi penanaman ECU ke subsistem lain atau perangkat tertanam lainnya, asalkan batas keamanan dipertahankan seperti yang ditunjukkan pada Gambar 4.49.



Gambar 4.49 Batasan keamanan kriptografi.

Ketika ECU tertanam ke dalam sistem tertanam, antarmuka keamanan penting mungkin perlu diubah. Misalnya, jika teks biasa kunci pribadi memasuki sistem tertanam dari antarmuka I/O sebelum dimuat ke dalam ECU, maka bagian dari sistem tertanam yang

memproses teks biasa dan meneruskannya ke ECU sekarang harus dimasukkan dalam batas keamanan yang diperluas. . Bayangkan kernel Linux mengelola antarmuka I/O dan jalur ke ECU; tiba-tiba, ratusan ribu baris kode sumber terbuka dibawa ke batas keamanan! Hal ini dapat mengubah proses sertifikasi yang sederhana menjadi proses yang tidak dapat diatasi.

Mengabaikan area permukaan tambahan ini berpotensi mengakibatkan kompromi kunci, karena kita tidak tahu apa lagi yang dilakukan perangkat lunak tidak tepercaya tersebut dengan kunci tersebut. Sekali kunci disusupi, sistem pun ikut disusupi. Yang lebih parah lagi, kita bahkan tidak tahu kapan kompromi itu terjadi; bisa saja pada hari pertama penggunaan sistem.

Untuk aplikasi kriptografi apa pun (terklasifikasi atau tidak terklasifikasi), batas keamanan harus dijelaskan dan digambar dengan cermat. Batasan keamanan yang "tidak jelas" harus dihindari karena pada akhirnya akan menyebabkan rancangan suatu sistem menjadi tidak aman.

Meskipun NSA dan NIST menentukan desain ECU menggunakan model yang ditunjukkan sebelumnya, mereka tidak menjamin bahwa sistem yang menggunakan ECU aman. Misalnya, kita mengembangkan jaringan paket aman untuk menangani informasi rahasia. Penggunaan ECU bersertifikat NSA untuk melindungi informasi yang mengalir melalui jaringan paket diperlukan namun tidak cukup untuk menjamin keamanan jaringan itu sendiri. Sebaliknya, kami memerlukan sertifikasi sistem keamanan. Persyaratan keamanan fungsional dan jaminan harus ditentukan dan dipenuhi pada tingkat sistem sesuai dengan kebijakan keamanan sistem. Kebijakan ini mencakup manajemen kunci dan aspek keamanan sistem lainnya yang berada di luar cakupan ECU itu sendiri.

Sayangnya, sistem biasanya ditentukan untuk fungsionalitas umum, dan elemen pengamannya dipasang kemudian. Hal ini biasanya mengakibatkan kerentanan. Salah satu contoh nyata adalah Internet. Ketika Internet pertama kali dirancang, ditentukan, dan diimplementasikan, persyaratan keamanan tidak dirumuskan. Sekarang kita menggunakan Internet untuk banyak layanan keamanan penting, termasuk semua jenis transaksi keuangan, keamanan semakin ditingkatkan. Namun, Internet mempunyai kerentanan yang luar biasa dalam hampir setiap aspek, mulai dari routing dan manajemen kunci hingga banyak protokol komunikasi inti Internet. Dapat dikatakan bahwa Internet tidak akan pernah benar-benar aman terlepas dari berapa banyak fitur keamanan yang dipasang. Ada terlalu banyak entitas yang tidak dapat dipercaya, dan batas keamanan Internet sangat "tidak jelas".

Strategi Interoperabilitas Kriptografi NSA

Kemajuan teknologi informasi yang berkelanjutan dan pesat di abad kedua puluh satu telah memberikan motivasi bagi NSA untuk mengembangkan strategi kriptografi baru yang dapat disesuaikan untuk melindungi informasi keamanan nasional. NSA berharap bahwa melalui strategi interoperabilitasnya, para pengembang akan menyediakan solusi kriptografi yang kuat yang mudah diadopsi dan dibagikan antara para pejuang perang, responden pertama, dan mitra federal dan internasional.

Bab 5 membahas Suite B dan dampaknya terhadap protokol keamanan jaringan umum seperti SSL dan IPsec. Pemberlakuan Suite B adalah bagian dari inisiatif NSA baru-baru ini untuk menggunakan perangkat keras dan perangkat lunak kriptografi komersial siap pakai untuk informasi yang diklasifikasikan di Secret-and-below (SAB), termasuk sensitif tetapi tidak

rahasia. Selain itu, NSA telah menciptakan kategori baru sertifikasi kriptografi untuk meningkatkan sertifikasi tradisional Tipe-1 yang telah dibahas sebelumnya. Dengan memvariasikan persyaratan sesuai dengan nilai informasi dan lingkungan ancamannya, pengembang dan pemberi sertifikasi dapat memenuhi tujuan keamanan sekaligus mengendalikan biaya dan meningkatkan waktu pemasaran. Kategori sertifikasi untuk SAB dibahas selanjutnya.

OTS untuk RAHASIA

Kontraktor federal membuat produk pemerintah yang siap pakai (GOTS) khusus untuk pemerintah AS. Sertifikasi GOTS-untuk-rahasia dimaksudkan untuk melindungi data hingga Rahasia menggunakan rangkaian algoritma Suite B. Untuk sistem ini, NSA telah merevisi serangkaian persyaratan keamanan dan pengujian serta meminimalkan hasil sertifikasi, sehingga mengurangi waktu pemasaran.

Karena algoritme Suite B tidak terklasifikasi, perangkat GOTS untuk rahasia tidak dianggap sebagai Item COMSEC Terkendali (CCI). Perangkat CCI memiliki persyaratan penanganan yang ketat, termasuk kontrol akuntansi dan penandaan khusus. Dengan menghindari CCI, total biaya kepemilikan berkurang secara signifikan.

COTS untuk RAHASIA

Commercial-off-the-shelf (COTS) mengacu pada produk yang dibuat untuk pasar umum, seperti konsentrator VPN bersertifikat FIPS 140-2. COTS-for-secret memungkinkan, melalui Program Kemitraan Solusi Komersial (CSPP) yang disponsori NSA, vendor untuk mengusulkan solusi yang terdiri dari komposisi beberapa produk COTS. Program COTS-for-secret juga dikenal sebagai Commercial Solutions for Secret (CSfC). COTS-for-secret menetapkan sertifikasi FIPS 140-2 dan profil perlindungan Kriteria Umum pemerintah AS tertentu. Tingkat jaminan Kriteria Umum yang digunakan minimal karena tujuannya dalam konteks ini adalah untuk memastikan interoperabilitas.

Keuntungan utama COTS-for-secret dibandingkan GOTS-for-secret adalah bahwa peralatan itu sendiri, sebelum memuat kunci rahasia apa pun, tidak terklasifikasi meskipun dapat digunakan untuk perlindungan informasi rahasia. Penggunaan perangkat keras dan perangkat lunak COTS meringankan beban logistik bagi pengembang dan membuka pasar komunikasi rahasia ke lebih banyak pemasok.

Meski demikian, sertifikasi tersebut tetap memerlukan proses desain dan dokumentasi yang ketat. Biasanya, urutan proses persetujuan sertifikasi COTS-untuk-rahasia adalah sebagai berikut:

1. Untuk berbagai aplikasi, seperti LAN Nirkabel, VPN, dan enkripsi disk, NSA menerbitkan serangkaian spesifikasi yang menjelaskan arsitektur solusi yang harus diikuti oleh vendor.
2. Vendor mengembangkan solusi tersusun dari arsitektur yang dipublikasikan dan menyajikan solusi tersebut kepada NSA untuk ditinjau dan disetujui.
3. NSA menyetujui usulan penerapannya.

Penting untuk dicatat bahwa persetujuan NSA sangat berbeda dengan sertifikasi NSA tradisional. Meskipun persetujuan menunjukkan bahwa pengembang telah mengikuti pedoman NSA, tanggung jawab akhir atas penggunaan perangkat tersebut diserahkan kepada

Designated Approving Authority (DAA) dari organisasi pengguna akhir, seperti militer atau badan intelijen.

Seperti yang kita bahas di Bab 5, NSA telah bekerja sama dengan NIST dan IETF untuk memastikan bahwa standar dan protokol industri menyertakan algoritma Suite B. Selain itu, NSA telah melakukan banyak inisiatif infrastruktur yang akan mencakup Suite B, termasuk

1. Proyek Infrastruktur Kunci Publik (PKI) Departemen Pertahanan akan beralih dari sertifikat RSA lama ke sertifikat kurva elips pilihan Suite B pada tahun 2012
2. Infrastruktur Manajemen Kunci (KMI): program NSA untuk menyediakan layanan manajemen kunci untuk produk Tipe 1 dan SAB
3. PKI komersial yang sesuai dengan Suite B untuk lembaga sensitif namun tidak rahasia

NSA juga telah melisensikan banyak teknologi dan menyediakannya secara gratis bagi vendor yang membuat produk Tipe 1 atau Rahasia-dan-di bawahnya. Misalnya, NSA melisensikan hak atas 26 paten teknologi kurva elips dan menyediakannya secara gratis bagi industri yang mengembangkan produk untuk pemerintah AS.

Perkembangan besar baru-baru ini dalam strategi kriptografi NSA akan meningkatkan pembagian informasi yang aman ke banyak perusahaan termasuk jaringan Departemen Pertahanan, lembaga federal dan negara bagian, dan jaringan perusahaan komersial. Selain itu, NSA juga berharap bahwa dengan mengembangkan Suite B dengan skalabilitas terkait untuk data berkecepatan tinggi dan ukuran kunci efisien yang penting untuk sistem nirkabel, algoritma lama seperti DES, 3DES, dan SHA-1 yang berkinerja buruk atau menunjukkan keamanan yang lemah akan dapat membantu dibuang.

BAB 5
PROTOKOL PERLINDUNGAN DATA UNTUK SISTEM TERTANAM

5.1 PENDAHULUAN

Banyak sistem tertanam modern memiliki persyaratan perlindungan data-in-motion (keamanan jaringan) atau data-at-rest (penyimpanan terenkripsi), yang didorong oleh perlindungan kekayaan intelektual tertanam, manajemen jarak jauh yang aman, manajemen hak digital, transaksi keuangan di perangkat yang terhubung ke Internet, dan banyak lagi. Bab ini berfokus pada kebijakan kerahasiaan data, integritas, dan otentikasi asal, karena kebijakan tersebut umum di hampir semua protokol perlindungan data. Yang paling tidak dibahas dalam diskusi ini adalah kebijakan kontrol akses data, yang cenderung spesifik pada jenis aplikasi atau media dan kurang dapat diterapkan secara umum dari sudut pandang protokol.

Bab ini memberikan gambaran umum tentang protokol keamanan jaringan terbaru dan terpenting seperti TLS v1.2, DTLS, dan IKE v2. Selain itu, bab ini membahas pendekatan yang kurang terstandarisasi untuk melindungi data yang tidak digunakan, seperti penggunaan enkripsi disk penuh (FDE). Bab ini mendemonstrasikan cara menggunakan alat-alat ini secara praktis dalam sistem tertanam dan memberikan saran berguna mengenai kinerja, jejak kaki, dan standar sertifikasi seperti NSA Suite B. Bab ini ditujukan bagi insinyur sistem tertanam (perangkat lunak, perangkat keras, sistem) yang memiliki pengalaman minimal dalam menggunakan protokol perlindungan data atau memiliki pemahaman umum tetapi kurang pengalaman praktis dalam mengintegrasikan fitur-fitur ini ke dalam sistem tertanam. Materi ini juga berguna bagi insinyur keamanan jaringan yang lebih berpengalaman yang ingin mempelajari revisi standar terbaru dan panduan ahli. Bab ini mengasumsikan pemahaman dasar kriptografi, termasuk algoritma enkripsi simetris, kriptografi kunci publik, dan konsep manajemen kunci. Topik-topik ini dibahas dalam Bab 4.

5.2 PROTOKOL DATA-IN-MOTION

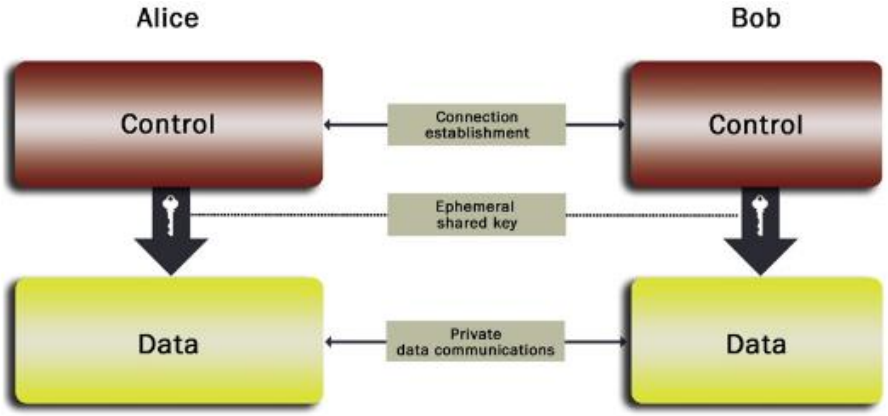
Protokol keamanan data-in-motion (juga disebut sebagai data-in-transit) terlibat dalam pembentukan dan penggunaan saluran yang dilindungi di mana pasangan (point-to-point) atau set (point-to-multipoint atau multipoint- to-multipoint) entitas dapat berkomunikasi dengan aman. Kami terutama memperhatikan keamanan jaringan berbasis Protokol Internet (IP).

5.2.1 Model Umum

Titik-ke-Titik

Anggaplah Alice ingin membuat saluran terlindungi dengan Bob. Pada tingkat tinggi, Alice dan Bob menggunakan program kontrol untuk membangun saluran dan program data untuk kemudian mengirimkan informasi massal melalui saluran yang sudah ada (lihat Gambar 5.1). Ketika Alice dan Bob selesai dengan sesi komunikasi mereka, saluran tersebut dirobohkan (pembersihan ini dianggap sebagai bagian dari program kontrol).

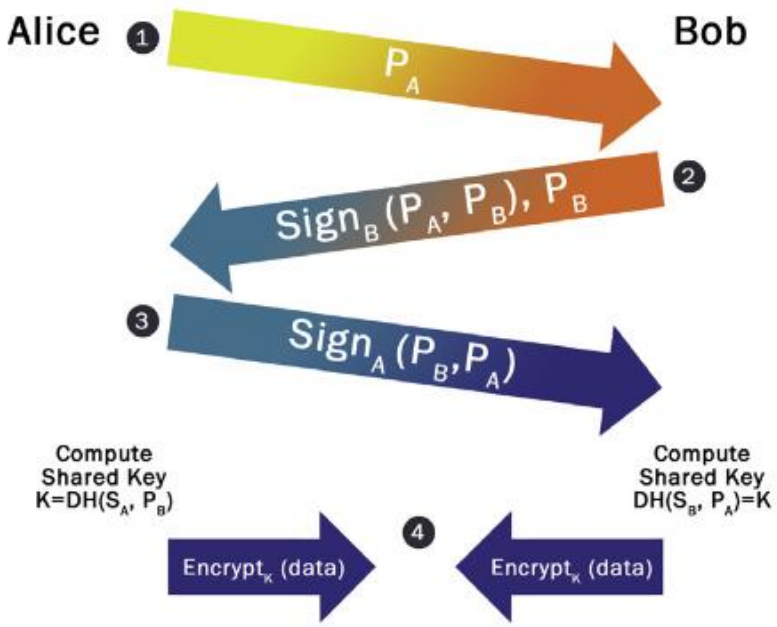
Untuk menelusuri lebih jauh, mari kita asumsikan bahwa kriptografi kunci publik digunakan untuk pembuatan sesi, dan sebelum sesi apa pun, Alice sudah memegang kunci publik jangka panjang (statis) tervalidasi milik Bob, Bob memegang kunci publik jangka panjang (statis) tervalidasi milik Alice. kunci publik, dan mereka hanya ingin berkomunikasi satu sama lain (mereka tidak akan pernah mendapatkan kunci publik lainnya). Yang kami maksud dengan "divalidasi" adalah Alice dan Bob berbagi otoritas sertifikasi (CA) yang sama, yang sertifikat pra-instalnya digunakan untuk memvalidasi kunci publik masing-masing terlebih dahulu. Misalnya, Alice dan Bob mungkin saling menerima sertifikat kunci publik dari server sertifikat tepercaya di jaringan. Perhatikan bahwa dalam beberapa sistem tertanam, kunci publik rekan komunikasi mungkin sudah diinstal sebelumnya di pabrik dengan cara yang tepercaya. Dalam hal ini, Alice dan Bob tidak perlu memvalidasi kunci publik satu sama lain sama sekali, dan tidak ada kunci publik otoritas sertifikat bersama yang perlu diinstal sebelumnya.



Gambar 5.1 Model umum tingkat tinggi dari subsistem komunikasi aman point-to-point.

Pada gambar 5.2 menunjukkan otentikasi umum dan urutan perjanjian kunci yang digunakan untuk membangun saluran antara Alice dan Bob untuk transmisi data massal yang aman.

Pada langkah pertama, Alice membuat pasangan kunci publik sementara yang acak dan tidak dapat diprediksi (misalnya RSA atau kurva elips) dan mengirimkan bagian publik, PA, ke Bob. Alice merahasiakan bagian lainnya, SA. Bob menerima PA dan juga membuat pasangan kunci publik sementara acak (SB, PB). Pada langkah kedua, Bob mengirimkan PB dan tanda tangan digital dari gabungan dua kunci publik yang dipertukarkan, PA dan PB. Bob membuat tanda tangan ini, yang disebut sebagai SignB pada Gambar 5.2, menggunakan kunci privat statisnya (ingat bahwa Alice sudah memegang bagian publik yang cocok). Algoritme tanda tangan sesuai dengan jenis kunci publik statis yang digunakan (misalnya RSA atau ECDSA). Alice menerima pesan ini dan memvalidasi tanda tangan Bob menggunakan kunci publik statis Bob. Pada langkah ketiga, Alice menandatangani rangkaian PB dan PA dan mengirimkan hasilnya ke Bob. Bob menerima dan memvalidasi tanda tangan menggunakan kunci publik statis Alice.



Gambar 5.2 Protokol data-in-motion yang aman dan point-to-point yang digeneralisasi.

Kami hampir selesai dengan pembuatan sesi. Alice dan Bob telah bertukar kunci publik sementara dan oleh karena itu dapat menggunakan algoritma Diffie-Hellman yang elegan dan kuat (lihat Bab 4) untuk membuat rahasia bersama yang dihitung dari kunci pribadi sementara yang disimpan dan kunci publik yang diterima. Bentuk spesifik Diffie-Hellman bergantung pada jenis kunci sementara yang digunakan, misalnya Diffie-Hellman standar atau Elliptic Curve Diffie-Hellman (ECDH) saat menggunakan kunci sementara kurva elips. Karena dihitung dari kunci pribadi yang hanya dimiliki oleh Alice dan Bob, rahasia bersama sekarang dapat digunakan sebagai kunci simetris untuk enkripsi yang diautentikasi secara massal (misalnya, AES-GCM). Alice dan Bob dapat berkomunikasi dengan percaya diri menggunakan kunci ini, mengetahui bahwa tidak ada orang lain selain Alice dan Bob yang dapat membaca data.

Mari kita lihat tujuan umum dari protokol data-in-motion dan bagaimana urutan sebelumnya memenuhi masing-masing tujuan:

- **Otentikasi bersama**: Tanda tangan Bob di PA, yang dikirimkannya pada langkah kedua, memberikan jaminan kepada Alice bahwa dia memang berkomunikasi dengan penerima yang dituju, Bob. Tanda tangan Alice pada PB pada langkah ketiga memberikan jaminan yang sama bagi Bob. Dengan demikian, Bob dan Alice telah saling mengautentikasi.
- **Keaktifan**: Karena PA Alice tidak dapat diprediksi (acak), tanda tangan PA yang dikembalikan oleh Bob menunjukkan kepada Alice bahwa Bob berpartisipasi aktif dalam percakapan (bukan pengulangan komunikasi sebelumnya). Demikian pula, tanda tangan PB Alice membuktikan keaktifan Alice di mata Bob.
- **Perjanjian Utama**: Bob dan Alice telah menghitung kunci pribadi baru yang sama yang kemudian dapat mereka gunakan untuk berkomunikasi dengan aman. Konsekuensi dari perjanjian kunci adalah konfirmasi kunci dimana Alice dan Bob benar-benar memiliki bukti bahwa pihak lain telah menghitung kunci rahasia bersama. Dalam

skenario umum kami, konfirmasi tidak tercapai sampai data massal dikirim dan kemudian berhasil diterima (diautentikasi dan didekripsi).

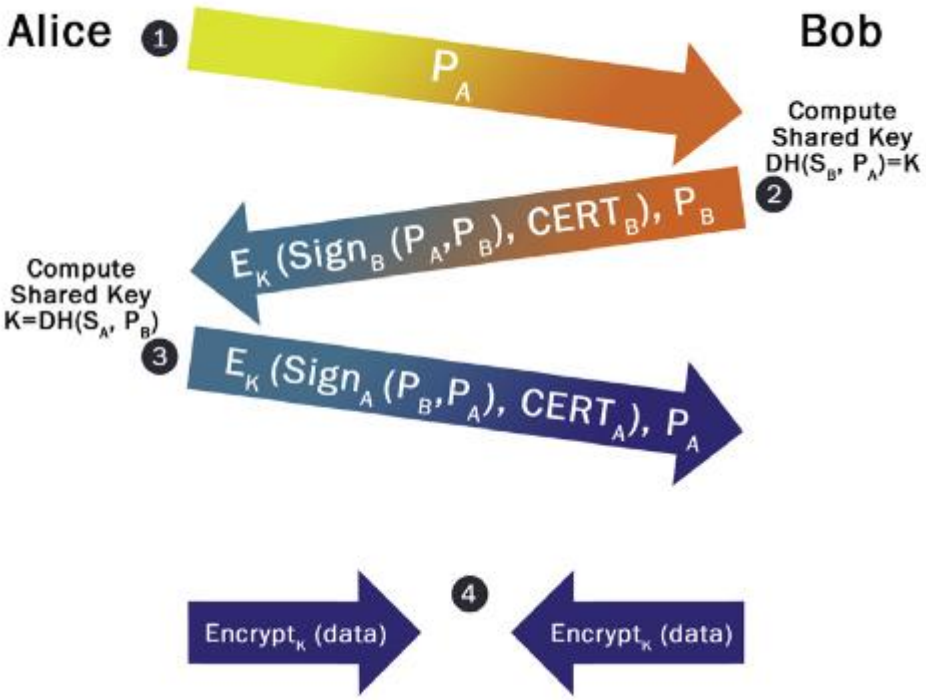
Karena inti dari protokol data-in-motion adalah menggunakan kunci rahasia untuk komunikasi, langkah konfirmasi kunci yang eksplisit selama proses pembuatan tidak sepenuhnya diperlukan.

Anonimitas identitas (bagian dari keamanan transmisi) sering kali menjadi tujuan yang diinginkan dari protokol data-in-motion. Dalam urutan sebelumnya, pesan pertukaran kunci dan tanda tangan terkait dikirim secara jelas. Seorang penyadap yang memegang salinan kunci publik Alice dan Bob (kita biasanya berasumsi hal ini mungkin terjadi karena kuncinya bersifat publik) dapat memvalidasi pesan dan menentukan bahwa Alice dan Bob sedang berkomunikasi. Anonimitas identitas dapat dengan mudah ditambahkan ke urutan sebelumnya. Karena dia memiliki kunci publik sementara Alice, Bob dapat menghitung rahasia bersama sebelum mengirim pesan balasan pada langkah kedua. Oleh karena itu, Bob mengenkripsi tanda tangan dalam pesan ini, sehingga menggagalkan penyadap. Demikian pula, Alice dapat menghitung rahasia bersama sebelum mengirim pesan pada langkah ketiga. Dia juga mengenkripsi tanda tangannya.

Dalam praktiknya, bergantung pada media transportasi yang mendasarinya (misalnya, Protokol Internet di atas Ethernet), informasi identitas dapat disimpulkan dari paket transportasi tersebut jika identitas yang berkomunikasi dapat dipetakan ke lokasi jaringannya. Seperti yang akan kita bahas nanti di bab ini dalam contoh mode Terowongan IPsec, mengatasi anonimitas mungkin perlu ditangani secara independen.

Model sederhana yang dibahas sejauh ini mengasumsikan bahwa Alice dan Bob hanya dapat berbicara satu sama lain. Dalam praktiknya, Alice dan Bob hanyalah dua konstituen dalam jaringan yang lebih besar dengan lebih banyak entitas. Dalam jaringan tertutup dengan jumla entitas yang dapat dikelola (misalnya, intranet perusahaan), setiap titik akhir dapat diinstal sebelumnya dengan daftar mitra komunikasi potensial, dan informasi identitas harus dipertukarkan dalam protokol untuk memberi tahu Alice dan Bob publik mana kunci yang digunakan untuk validasi pesan. Dalam jaringan terbuka dengan jumlah entitas yang hampir tidak terbatas (misalnya Internet), sertifikat kunci publik biasanya dikirim sebagai bagian dari jabat tangan. Dengan kata lain, sebelum Alice dapat berbicara dengan Bob, Alice tidak akan mengirimkan identitas sederhananya melainkan seluruh sertifikat kunci publiknya, dan Bob perlu memvalidasi tanda tangan tersebut selama proses pembuatan. Begitu pula Bob akan membalasnya dengan sertifikatnya.

Urutan yang ditingkatkan, termasuk enkripsi tanda tangan dan transmisi sertifikat, ditunjukkan pada Gambar 5.3.



Gambar 5.3 Protokol keamanan jaringan point-to-point yang digeneralisasi dengan keamanan transmisi yang ditingkatkan.

Meskipun rincian lainnya dihilangkan dalam model umum komunikasi aman point-to-point data-in-motion ini, hal ini membentuk dasar protokol keamanan jaringan point-to-point kunci publik modern, seperti IKE/IPsec, SSL, dan DTLS , dibahas nanti dalam bab ini. Bagi pembaca yang suka berpetualang dan ingin mendalami sejarah, keragaman, dan bukti formal dari metode pembentukan kunci, kami merekomendasikan buku yang sangat bagus, *Protocols for Authentication and Key Creating* (Protokol untuk Otentikasi dan Pembentukan Kunci).[1]

Titik ke Multititik

Misalkan Alice ingin berkomunikasi dengan aman ke sejumlah besar penerima. Membangun sesi point-to-point dengan masing-masing penerima mungkin tidak praktis dan tentunya tidak efisien, sehingga memerlukan data yang sama untuk dikirim berulang kali. Contoh masalah sistem tertanam yang memerlukan point-to-multipoint adalah mengkomunikasikan perubahan manajemen jarak jauh (modifikasi konfigurasi, pembaruan firmware) ke sejumlah besar sistem yang diterapkan. Ketika komunikasi multicast digunakan (data dikirim satu kali dan media yang mendasarinya menangani pengiriman data ke semua penerima dalam grup multicast tertentu), bagian kontrol menjadi agak rumit. Daripada menggunakan protokol pertukaran kunci sederhana, skema manajemen kunci grup harus digunakan. Skema ini harus menangani rekeying berkala di seluruh grup.

Aplikasi multipoint memaksa pengelolaan kunci dan tantangan pertukaran untuk dipisahkan dari protokol data massal: pengembang harus menemukan cara untuk mendistribusikan kunci bersama ke seluruh grup dan kemudian paket multicast dilindungi dengan kunci tersebut. Pembaca yang tertarik untuk mempelajari lebih lanjut tentang

[1] Boyd C, Mathuria A. Protocols for Authentication and Key Establishment. Heidelberg: Springer-Verlag; 2003

multicast dengan IPsec harus berkonsultasi dengan RFC 5374 dan 4046[2].[3] Diskusi IPsec selanjutnya dalam bab ini hanya berfokus pada kasus point-to-point tradisional.

Siaran adalah kasus khusus point-to-multipoint di mana komunikasi hanya dilakukan satu arah (misalnya transmisi satelit ke penerima terestrial). Siaran membuat protokol keamanan jaringan berbasis IP tradisional menjadi lebih tidak praktis karena ketidakmampuan untuk bertukar pesan apa pun. Sekali lagi, kunci yang dibagikan sebelumnya harus dibuat antara penyiar dan penerima, dan kemudian data yang dienkripsi dengan kunci yang dibagikan sebelumnya dapat ditandatangani dan dienkripsi sebelum disiarkan. Ini adalah cara konten kabel dan satelit berbayar dikirim antara penyedia layanan dan sejumlah besar rumah. Pendekatan penyiaran akan dibahas nanti dalam bab ini.

Konferensi adalah contoh klasik dari multipoint-to-multipoint: semua peserta perlu berkomunikasi dengan semua peserta lainnya. Namun, informasi dihasilkan secara individual oleh masing-masing peserta; oleh karena itu, skenario ini hanyalah sebuah kasus khusus dari beberapa koneksi point-to-multipoint secara bersamaan.

5.2.2 Memilih Lapisan Jaringan untuk Keamanan

Seperti yang ditunjukkan pada Gambar 5.4, perancang sistem tertanam dapat memilih di antara berbagai lapisan dalam tumpukan komunikasi untuk menerapkan protokol perlindungan data. Karena masalah efisiensi kinerja, sebagian besar sistem ingin menerapkan perlindungan pada satu tingkat.

> *Perlindungan data berlapis dapat digunakan sebagai strategi pertahanan mendalam dan juga dapat terjadi ketika lapisan atas tidak menyadari atau tidak mempercayai kemampuan perlindungan lapisan bawah. Seperti yang diharapkan, pilihan protokol praktis bervariasi tergantung pada lapisan yang dipilih.*

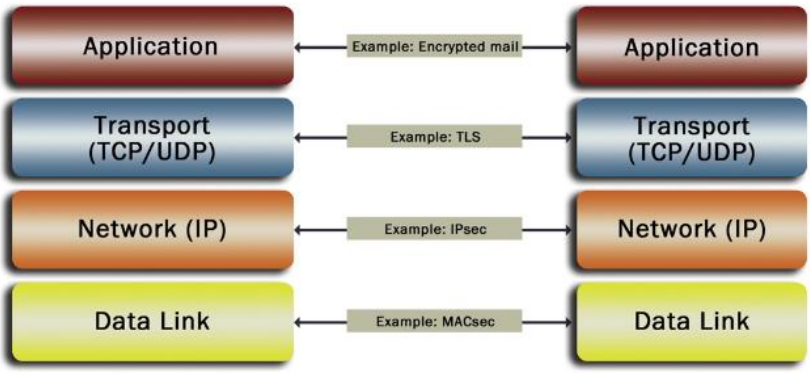
Pilihan lapisan perlindungan jaringan dapat berdampak signifikan terhadap fungsionalitas, kinerja, interoperabilitas, dan pemeliharaan. Memahami trade-off ini merupakan tujuan penting bab ini.

Pilihan terendah yang tersedia untuk keamanan jaringan adalah lapisan data link pada model OSI dan/atau lapisan link pada model TCP/IP. Setiap tautan data yang diketik Bluetooth, Ethernet, Modem, ATM, IEEE 802.15.4, Frame Relay (hanya beberapa di antaranya) biasanya menggabungkan pendekatan perlindungan data berbeda yang disesuaikan dengan karakteristik pembingkaian dan kontrolnya. Terkadang, enkripsi tautan data dibangun langsung ke dalam perangkat keras periferal komunikasi.

---

[2] Multicast Extensions to the Security Architecture for the Internet Protocol. Internet Engineering Task Force, Request for Comments: 5374; November 2008.

[3] Multicast Security (MSEC) Group Key Management Architecture. Internet Engineering Task Force, Request for Comments: 4046; April 2005.



Gambar 5.4 Kemungkinan perlindungan keamanan jaringan berdasarkan lapisan.

Penanganan protokol dan enkripsi yang efisien dalam perangkat keras dan kurangnya sumber daya pemrosesan dan penyimpanan aplikasi cadangan memaksa sistem tertanam tertentu untuk mengandalkan perlindungan lapisan data link daripada pilihan lapisan atas yang lebih fleksibel.

Contoh bagusnya adalah perangkat nirkabel berdaya sangat rendah seperti smart meter. Perangkat kontemporer yang mendukung ZigBee hampir secara eksklusif mengandalkan keamanan lapisan data link untuk mempertahankan masa pakai baterai yang optimal dan kinerja jaringan yang baik melalui radio berkecepatan rendah.

Pendekatan pengelolaan kunci mungkin juga sangat berbeda berdasarkan lapisannya. Misalnya, protokol lapisan data link cenderung menggunakan skema manajemen kunci yang sederhana. Di beberapa jaringan ZigBee, satu kunci simetris bersama digunakan untuk seluruh jaringan; pengekuncian ulang secara berkala memerlukan flashing ulang dan mungkin tidak pernah dilakukan dalam praktiknya; kuncinya dipasang dari pabrik ke perangkat administratif di jaringan; dan perangkat titik akhir dikirimi kunci bersama melalui udara, secara jelas. Mereka yang telah membaca Bab 4 (atau bahkan mereka yang belum), kami berharap, melihat banyak kelemahan dalam skema pengelolaan kunci ini.

Kelemahan terbesar perlindungan data lapisan tautan adalah, menurut definisi, skema ini hanya dapat bekerja pada jaringan yang terdiri dari perangkat yang menggunakan media tautan data homogen. Oleh karena itu, smart meter berbasis ZigBee tidak dapat memperoleh koneksi langsung dan aman dengan infrastruktur daya back-end yang berada di seluruh WAN. Satu-satunya cara praktis untuk mendapatkan koneksi aman end-to-end antara entitas ini adalah dengan naik ke tingkat yang lebih tinggi seperti IPsec melalui 6LoWPAN (lebih lanjut tentang ini nanti).

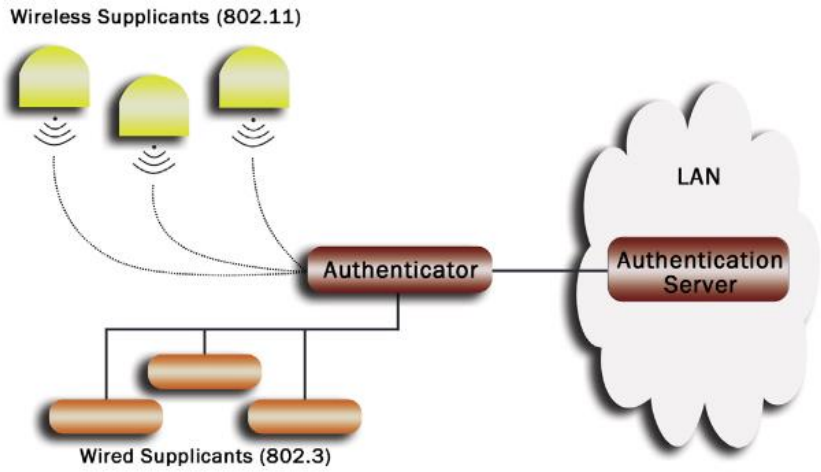
5.2.3 Protokol Keamanan Ethernet

Sisa bab ini berfokus pada protokol tingkat atas, seperti IPsec, karena penerapannya yang lebih umum. Namun, sebelum kita membahasnya, salah satu bentuk keamanan lapisan data link perlu disebutkan: manajemen kunci dan pembentukan sesi serta protokol enkripsi massal, masing-masing, untuk Ethernet (baik kabel maupun nirkabel). Ethernet, khususnya nirkabel (Wi-Fi), adalah salah satu jenis tautan data yang paling umum dalam sistem tertanam.

802.1X

Standar IEEE 802.1X mendefinisikan persyaratan untuk penggabungan titik akhir yang aman ke dalam jaringan Ethernet. Perangkat pengautentikasi 802.1X bertindak sebagai

gerbang, menolak semua lalu lintas Ethernet (kecuali permintaan koneksi 802.1X) yang tidak berasal dari titik akhir yang diautentikasi dengan benar. Authenticator 802.1X seringkali merupakan perangkat yang relatif sederhana, seperti switch Ethernet atau titik akses Wi-Fi. Logika keputusan otentikasi biasanya dipindahkan ke server otentikasi dalam LAN, yang pada gilirannya dapat memanfaatkan server tambahan seperti database kredensial pengguna. Dalam terminologi 802.1X, titik akhir permintaan disebut pemohon[4]. Arsitektur server pemohon tiga arah, autentikator, dan autentikasi 802.1X ditunjukkan pada Gambar 5.5.



Gambar 5.5 Arsitektur entitas IEEE 802.1X.

Kerangka protokol untuk otentikasi, Extensible Authentication Protocol (EAP), ditentukan di 802.1X. Namun, metode otentikasi khusus yang digunakan pada jaringan tertentu tidak ditentukan oleh 802.1X. EAP didefinisikan dalam RFCs 3748 dan 5247[5,6]. Otentikasi mungkin sesederhana memvalidasi bahwa alamat MAC Ethernet pemohon ada dalam daftar putih titik akhir yang diizinkan atau sekomprehensif skema ganda yang memerlukan nama pengguna/kata sandi yang valid dan otentikasi berbasis sertifikat kunci publik bersama.

Lebih dari 40 metode EAP telah ditetapkan selama bertahun-tahun, dan metode EAP yang spesifik diperlukan untuk mengklaim kesesuaiannya dengan standar tertentu. Misalnya, RFC 4017 mendefinisikan persyaratan metode EAP yang diperlukan untuk jaringan nirkabel IEEE 802.11[7].

Banyak informasi rinci tentang 802.1X tersedia di buku dan di Internet. Salah satu judul yang disarankan adalah Implementing 802.1X Security Solutions for Wired and Wireless Networks karya Jim Geier[8].

---

[4] IEEE Standard 802.1x-2001. IEEE Standard for Local and Metropolitan Area NetworksdPort-Based Network Access Control. IEEE Computer Security; June 2001.

[5] Extensible Authentication Protocol (EAP). Internet Engineering Task Force, Request for Comments: 3748; June 2004.

[6] Extensible Authentication Protocol (EAP) Key Management Framework. Internet Engineering Task Force, Request for Comments: 4017; March 2005.

[7] Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs. Internet Engineering Task Force, Request for Comments: 5247; August 2008.

[8] Geier J. Implementing 802.1x Security Solutions for Wired and Wireless Networks. New York: Wiley; 2008.

802.11i

Pemetaan persyaratan *Extensible Authentication Protocol* (EAP) ke LAN Ethernet nirkabel dan persyaratan keamanan Ethernet nirkabel lainnya ditentukan dalam IEEE 802.11i-2004.

802.11i menggabungkan standar 802.1X untuk otentikasi[9]. Model umum untuk koneksi point-to-point yang disajikan sebelumnya dalam bab ini, termasuk otentikasi timbal balik yang kuat, kunci sesi yang baru disepakati, dan keaktifan, semuanya tercakup dalam 802.11i. Namun, keamanan data 802.11i ditujukan pada hubungan antara titik akhir dan titik akses, bukan antara titik akses dan ujung komunikasi logis lainnya, di belakang titik akses.

Fungsi keamanan yang ditentukan dalam IEEE 802.11i melindungi terhadap penyadap lokal dan peretas yang mencoba mendapatkan akses ke jaringan melalui titik akses tetapi tidak memberikan perlindungan ujung ke ujung. Perancang sistem tertanam harus menggunakan VPN atau protokol tingkat tinggi lainnya untuk menghubungkan sistem tertanam ke jaringan back-end, bahkan ketika 802.11i sedang digunakan. Kita tidak boleh mempercayai jaringan publik (misalnya Internet) yang terletak di antara titik akses dan sistem back-end. Faktanya, ada yang berpendapat bahwa jaringan yang dijaga ketat dengan konsentrator VPN yang menerapkan autentikasi timbal balik yang kuat ke semua titik akhir jarak jauh (yang menjalankan klien VPN yang dikonfigurasi dengan tepat) tidak perlu menggunakan keamanan tautan data sama sekali dalam koneksi tersebut. Hal ini sedikit berbeda dengan penggunaan mobile smartphone dan laptop dimana endpoint digunakan baik untuk koneksi VPN maupun koneksi Wi-Fi publik yang sama-sama memerlukan keamanan data.

Pada saat penulisan ini, 802.11i menetapkan penggunaan AES-CCM, mode enkripsi terotentikasi yang dibahas di Bab 4, untuk kerahasiaan data dan perlindungan integritas. Spesifikasi 802.11i mengenai bagaimana AES-CCM diterapkan pada frame Ethernet 802.11 disebut Counter Mode dengan *Cipher Block Chaining Message Authentication Code Protocol* (CCMP). Menarik untuk dicatat bahwa AES-GCM, yang lebih disukai oleh NSA Suite B dalam protokol tingkat yang lebih tinggi, belum diadopsi dalam protokol keamanan tautan data. Pada saat tulisan ini dibuat, cakupan NSA Suite B tidak mencakup keamanan tautan data, sehingga menyiratkan preferensi untuk IPsec dan pendekatan lapisan lebih tinggi lainnya. Karena kecepatan Ethernet nirkabel terus meningkat, ada kemungkinan besar AES-GCM akan ditambahkan ke standar IEEE 802.11 karena efisiensi kinerja GCM yang unggul.

Keamanan data link umumnya lebih penting dalam nirkabel dibandingkan LAN kabel. Jaringan nirkabel dicirikan oleh topologi yang lebih dinamis, dimana jaringan secara teratur bergabung dengan titik akhir yang mungkin tidak dikelola dengan baik oleh administrator jaringan. Jaringan nirkabel lebih rentan terhadap penyadapan dan serangan man-in-the-middle lainnya. Terakhir, jaringan nirkabel menggantikan jaringan kabel di sebagian besar aplikasi tertanam karena fleksibilitasnya. Standar dasar untuk keamanan Ethernet nirkabel, 802.11i merupakan bacaan wajib bagi pelaksana keamanan Ethernet nirkabel yang serius. Wi-Fi Alliance telah menjadi yang terdepan dalam sertifikasi kepatuhan keamanan dan

[9] IEEE Standard 802.11i. Supplement to Standard for Telecommunications Information Exchange Between SystemsdLAN/MAN Specific RequirementsdPart 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification for Enhanced Security; July 2004.

interoperabilitas untuk jaringan Ethernet nirkabel (Wi-Fi). Aliansi Wi-Fi mendasarkan pengujian sertifikasinya saat ini pada IEEE 802.11i.

WPA2

Bertentangan dengan anggapan umum, Wi-Fi Protected Access 2 (WPA2) bukanlah standar keamanan definitif, melainkan sebuah kuasi-standar sertifikasi yang dimaksudkan untuk memberikan keyakinan kepada industri bahwa perangkat bersertifikasi Wi-Fi memenuhi persyaratan IEEE 802.11i.

Aliansi Wi-Fi memperkenalkan sertifikasi WPA2 pada tahun 2004, bersamaan dengan rilis IEEE 802.11i. WPA2 telah diamanatkan untuk produk bersertifikasi Wi-Fi sejak tahun 2006 dan menggantikan WEP dan WPA, pendekatan teknis yang terkenal lebih lemah yang diadopsi oleh program sertifikasi Alliance. Gambaran bagus mengenai penerapan standar keamanan Wi-Fi Alliance yang berpuncak pada sertifikasi WPA2 saat ini dapat ditemukan di situs web Wi-Fi Alliance.[10] Pengaruh Wi-Fi Alliance terhadap keamanan Ethernet nirkabel sangat luas sehingga sebagian besar pengguna perangkat Ethernet nirkabel menyamakan keamanan Ethernet nirkabel dengan nama WPA2 daripada standar protokol IEEE penyusunnya.

Pengembang sistem tertanam yang memperoleh perangkat lunak Ethernet nirkabel untuk digabungkan ke dalam perangkat harus memilih tumpukan yang tidak hanya mengklaim kesesuaian dengan WPA2, namun juga telah berhasil diterapkan pada produk bersertifikasi Wi-Fi. Perlu diperhatikan bahwa sertifikasi Wi-Fi Alliance hanya dilakukan pada produk akhir.

802.1AE

Dikenal sebagai MACsec, standar IEEE 802.1AE adalah analog dari IEEE 802.11i untuk jaringan Ethernet berkabel (IEEE 802.3)[11]. Berbeda dengan 802.11i, yang mewajibkan penggunaan 802.1X untuk bergabung dengan jaringan, cakupan MACsec hanya mencakup bidang data, tidak termasuk otentikasi akses jaringan. Namun, rancangan standar IEEE P802.1af menambah 802.1X dengan rincian yang diperlukan untuk penerapan ke 802.1AE. Draf ini telah dimasukkan ke dalam revisi berikutnya dari 802.1X, 802.1X-2010, yang belum diselesaikan oleh IEEE pada saat penulisan ini. Standar 802.1X secara de facto akan menjadi protokol autentikasi endpoint-to-switch Ethernet berkabel karena memungkinkan jaringan back-end mengimplementasikan infrastruktur server autentikasi terpadu (menggunakan RADIUS dan EAP) untuk endpoint berkabel dan nirkabel. Penulis 802.1AE cukup bijaksana mengenai peningkatan pesat kecepatan Ethernet kabel (maksimum 100 gigabit pada saat penulisan ini, dengan beberapa permintaan untuk 400 gigabit dan bahkan terabit)[12] untuk membuat AES-GCM, bukan AES-CCM, sandi default wajib untuk standar ini.

Singkatnya, protokol data link cocok untuk menghubungkan sistem tertanam dalam jaringan fisik homogen yang memerlukan manajemen kunci sederhana dan pemrosesan enkripsi yang efisien. Dan keamanan data link lebih baik daripada tidak ada keamanan data-in-transit jika sistem tertanam tidak mampu mempertahankan protokol tingkat yang lebih tinggi seperti IPsec. Protokol keamanan Ethernet, tentu saja, penting bagi pengembang peralatan switch dan titik akses Ethernet.

---

[10] The State of Wi-Fi Security. Wi-Fi Certified WPA2 Delivers Advanced Security to Homes, Enterprises and Mobile Devices. Wi-Fi Alliance; September 2009.

[11] IEEE Standard 802.1AE-2006. IEEE Standard for Local and Metropolitan Area NetworksdMedia Access Control (MAC) Security. IEEE Computer Security; August 2006.

[12] Lawson S. Facebook Sees Need for Terabit Ethernet. Computerworld; February 3, 2010.

Pengembang yang mencari keamanan data-in-motion yang paling portabel, fleksibel, dan dapat diperluas harus melihat di atas lapisan data link menuju jaringan (IP) dan lapisan yang lebih tinggi.

5.2.4 IPsec versus SSL

Dua keluarga protokol keamanan jaringan yang paling umum digunakan dalam sistem tertanam adalah IPsec dan *Secure Sockets Layer* (SSL). Pada tingkat tinggi, protokol-protokol ini memenuhi tujuan keamanan yang sama: kerahasiaan data yang dikirimkan, perlindungan integritas data yang dikirimkan, dan otentikasi asal data. Perbedaan utama antara kedua keluarga ini adalah tingkat tumpukan jaringan tempat mereka beroperasi.

IPsec beroperasi pada lapisan tiga (lapisan Protokol Internet), sehingga tidak terlihat oleh protokol tingkat yang lebih tinggi (misalnya, TCP) serta aplikasi. SSL dan varian modernnya, Transport Layer Security (TLS), beroperasi pada lapisan empat, lapisan transport; biasanya aplikasi menggunakan TLS dengan menggunakan API seperti soket.

Keuntungan utama dari TLS adalah ia dapat memanfaatkan jaminan pengiriman paket sesuai pesanan dari TCP untuk menyederhanakan protokol dibandingkan dengan IPsec yang harus menangani pesan yang hilang dan pengiriman yang tidak sesuai pesanan. Penggunaan TLS memungkinkan aplikasi untuk memilih kapan mereka membutuhkan keamanan (dan bersedia membayar biaya overhead yang terkait). Misalnya, browser web menggunakan TLS untuk mengamankan lalu lintas web. Sebaliknya, setelah koneksi IPsec dibuat, semua aplikasi dan layanan yang berjalan pada koneksi jaringan tersebut akan dipaksa untuk mematuhi kebijakan yang mendasarinya.

Keuntungan penting dari IPsec adalah aplikasi dapat memperoleh manfaat keamanan tanpa memerlukan keputusan tingkat aplikasi atau modifikasi kode apa pun (seperti menyesuaikan penggunaan API soket yang dimodifikasi).

5.2.5 IPSek

Terintegrasi versus Bump-in-the-Stack

IPsec bersifat opsional di IP versi empat (IPv4) dan diamanatkan di IP versi enam (IPv6). Sebagian besar tumpukan jaringan komersial, baik pihak ketiga atau yang dibangun ke dalam sistem operasi, sudah memiliki IPsec yang terintegrasi sebelumnya.

Beberapa vendor perangkat lunak tertanam menyediakan IPsec mandiri yang dapat berjalan secara independen dari subsistem komunikasi jaringan yang sudah ada; arsitektur ini disebut "*bump-in-the-stack*" (BITS) karena komponen IPsec merupakan lapisan terpisah ("bump") antara tumpukan tingkat yang lebih tinggi dan lapisan data link tingkat yang lebih rendah (misalnya, Ethernet). Secara umum, jika suatu organisasi telah menggunakan tumpukan TCP/IP dengan IPsec terintegrasi, maka organisasi tersebut harus memanfaatkan tumpukan tersebut. Jika tidak, penggunaan BITS mungkin merupakan pilihan yang baik karena kesederhanaannya, ukuran yang lebih kecil, dan kemudahan integrasi, meskipun kinerjanya mungkin tidak sebaik implementasi tumpukan yang telah disesuaikan dengan baik.

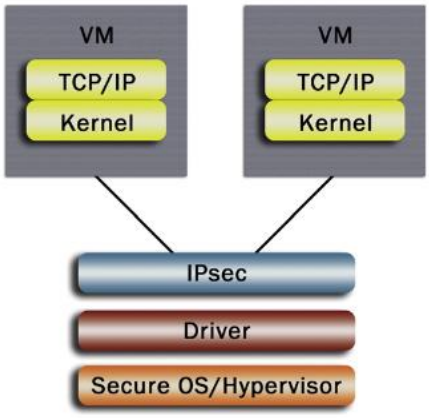
Contoh pilihan yang baik untuk bump-in-the-stack (BITS) IPsec adalah lingkungan tervirtualisasi di mana sistem operasi tamu sudah memiliki tumpukan jaringan yang terintegrasi, namun ada keinginan untuk menerapkan keamanan jaringan di luar mesin virtual dimana kelemahan dalam operasi tamu sistem dapat menggagalkan keamanan yang diinginkan.

Hypervisor dapat mengimplementasikan lapisan BITS IPsec-nya sendiri (lihat Gambar 5.6) di bawah semua sistem operasi tamu, bahkan tanpa sepengetahuan tamu dan aplikasinya. IPsec sering digeneralisasikan untuk merujuk pada penggunaan dua protokol terpisah yang terkoordinasi: IPsec dan *Internet Key Exchange* (IKE). IPsec berkaitan dengan aliran massal paket terenkripsi antar titik akhir jaringan, sedangkan IKE menangani masalah pelik seputar manajemen kunci dan membangun koneksi awal yang diautentikasi antar node.

IKE bekerja dengan IPsec dengan membuat kunci enkripsi massal sementara (disebut kunci sesi) untuk setiap koneksi baru dan kemudian memberi tahu lapisan IPsec tentang kunci sesi yang akan digunakan.

RFC IPsec

IPsec telah distandarisasi sejak lama dengan sedikit revisi. RFC asli diterbitkan pada tahun 1998 dan diperbarui pada tahun 2005. RFC 4301 adalah standar IPsec tingkat atas, mencakup IPv4 dan IPv6.[13] Meskipun mungkin agak sulit, membaca RFC yang disebutkan di seluruh bab ini sangat disarankan bagi pengembang yang ingin untuk mendapatkan pemahaman yang masuk akal tentang protokol daripada hanya menggunakannya.



Gambar 5.6 Penggunaan ipsec bump-in-the-stack di lingkungan mesin virtual.

IPsec menentukan dua sub-varian utama: *Authentication Header* (AH) dan *Encapsulate Security Payload* (ESP). Protokol ini masing-masing dicakup oleh RFC 4302 dan 4303[14].[15] AH hanya digunakan untuk perlindungan integritas dan otentikasi asal. Karena IPsec biasanya digunakan untuk komunikasi rahasia, AH jarang digunakan; sebaliknya, ESP adalah varian yang paling menarik. ESP memiliki dua mode operasi: Transportasi dan Terowongan. Mode transport ditujukan untuk koneksi host-to-host (bukan host-to-gateway dan gateway-to-gateway) dan tidak menyembunyikan informasi header paket IP asli.

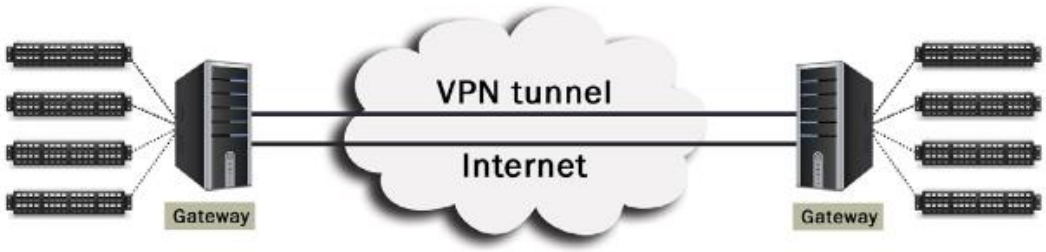
Sebaliknya, mode Tunnel sepenuhnya merangkum paket IP asli di dalam paket IP baru dengan header yang benar-benar baru. Tunneling memungkinkan informasi terenkripsi menyebar melalui jaringan terbuka, seperti Internet; header luar dan pengalamatan terkait digunakan untuk merutekan data antara titik akhir VPN (titik akhir klien dan server/konsentrator) melalui jaringan terbuka (lihat Gambar 5.7).

---

[13] Security Architecture for the Internet Protocol. Internet Engineering Task Force, Request for Comments: 4301; December 2005.

[14] IP Authentication Header. Internet Engineering Task Force, Request for Comments: 4302; December 2005.

[15] IP Encapsulating Security Payload (ESP). Internet Engineering Task Force, Request for Comments: 4303; December 2005.



Gambar 5.7 Penerowongan VPN.

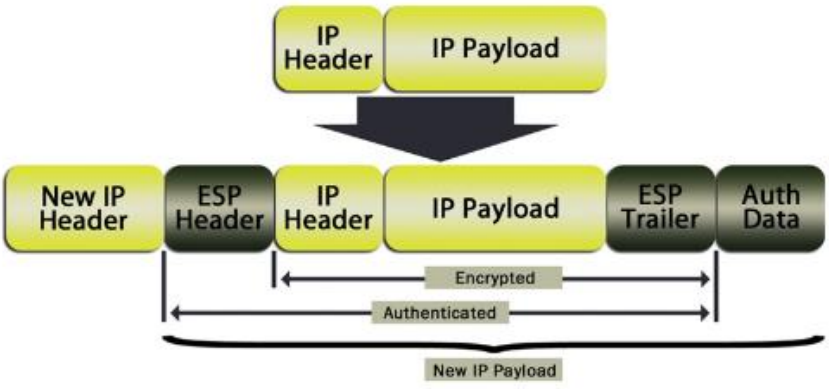
Selain mengenkripsi data paket, informasi alamat IP asli disembunyikan (dienkripsi) di jaringan.

Mode Terowongan ESP adalah kombinasi pilihan IPsec ketika keamanan dan kerahasiaan maksimum diperlukan. Enkapsulasi paket IP masukan ke dalam paket mode Terowongan ESP IPsec digambarkan pada Gambar 5.8. Data autentikasi tambahan digunakan untuk memverifikasi integritas data bingkai yang dienkapsulasi dan header ESP yang membantu mengelola hubungan keamanan antara pengirim dan penerima.

Mode terowongan menambahkan lebih banyak overhead jaringan daripada mode Transport karena header IP baru. Mode Transportasi dan Terowongan dibahas secara rinci dalam RFC IPsec yang disebutkan di atas.

IKE

Internet Key Exchange (IKE) mengikuti model umum untuk pembuatan koneksi aman yang dibahas di awal bab ini. Rekan yang ingin membuat koneksi aman dengan host jarak jauh mengirimkan informasi identifikasinya kepada host (misalnya, sertifikat kunci publik) serta beberapa data unik yang digunakan untuk memvalidasi bahwa pesan tersebut aktif (tidak diputar ulang dari sesi sebelumnya) dan sebagai benih untuk algoritma pembentukan kunci sesi. Pemrakarsa juga menandatangani data untuk menegaskan asal usulnya. Rekan penerima akan memverifikasi tanda tangan untuk mengautentikasi pemrakarsa dan kemudian mengirimkan kembali data yang ditandatangani, acak, dan informasi identifikasi untuk operasi sebaliknya. Setiap rekan menghitung kunci sesi berdasarkan data yang dipertukarkan dan algoritma yang disepakati, biasanya merupakan varian dari Diffie-Hellman. Kunci-kunci ini diteruskan ke modul IPsec untuk komunikasi sesi.



Gambar 5.8 Enkapsulasi mode terowongan IPsec ESP.

IKE juga mampu melakukan rekeying. Secara umum, masa pakai kunci sesi IKE/IPsec atau TLS (disebut periode kripto) harus dibatasi dan kunci sesi baru dibuat kembali untuk membatasi kerusakan jika kunci sesi disusupi. Institut Standar dan Teknologi Nasional (NIST) memberikan panduan periode kripto dalam publikasi khususnya 800-57[16], tetapi pada akhirnya, pilihan periode kripto yang tepat bergantung pada jumlah data yang digunakan untuk satu kunci serta nilai prorata waktu dari kunci tersebut. data. Untuk data taktis yang nilainya menurun segera setelah digunakan, periode kripto bisa relatif lama karena kunci yang disusupi akan memberikan penyerang akses hanya ke informasi yang sudah usang. Untuk data strategis dengan umur yang panjang (misalnya kode kendali senjata nuklir), periode kripto mungkin jauh lebih pendek dibandingkan dengan panduan NIST. Beberapa implementasi keamanan jaringan mungkin tidak memberikan perancang kemampuan untuk menyesuaikan periode kripto. Saat memilih produk keamanan jaringan, perancang harus berkonsultasi dengan vendor dan/atau dokumentasi produk untuk memahami apakah karakteristik penting ini dapat dikonfigurasi.

IKE versi 1 (IKEv1) dilindungi oleh RFC 2409[17], dan secara perlahan digantikan oleh IKEv2 (RFC 4306), yang memperbaiki beberapa masalah penting. IKEv2 tidak dapat dioperasikan dengan IKEv1, sehingga sulit untuk digunakan sendiri kecuali jika pengembang yakin bahwa rekan-rekan akan selalu mendukung versi yang lebih baru. Namun alternatifnya adalah menggunakan protokol lama yang diketahui memiliki kelemahan dan inefisiensi keamanan.

Secara umum, jika interoperabilitas dapat terjamin, IKEv2 sangat disarankan dibandingkan IKEv.[18] Pada akhir tahun 2010, RFC baru untuk IKEv2 diterbitkan, RFC 5996, yang tujuan utamanya adalah memperjelas dan memperkuat berbagai aspek spesifikasi RFC 4306 pendahulunya yang mengalami kebingungan selama lima tahun penggunaan.[19]

Pembongkaran Perangkat Keras IPsec

Semakin banyak prosesor tertanam yang canggih yang menggabungkan mesin keamanan yang dapat mempercepat kinerja IPsec secara dramatis sambil memindahkan pekerjaan ini dari CPU utama. Hal ini sangat penting dalam sistem tertanam dimana kekuatan pemrosesan sering kali dibatasi secara ketat. Mesin keamanan juga mengurangi dan menyederhanakan konten perangkat lunak keamanan jaringan, sehingga mengurangi kebutuhan memori secara keseluruhan. Misalnya, sebagian besar prosesor jaringan multi-inti modern, termasuk sebagian besar prosesor keluarga Freescale QorIQ dan Cavium OCTEON, memiliki mesin offload IPsec. Akselerasi dapat mencakup salah satu atau kedua bagian kriptografi kunci simetris publik dan massal dari IKE/IPsec.

Aliran-melalui versus Lookaside

Prosesor QorIQ dan OCTEON juga memberikan contoh yang baik dari dua pendekatan perangkat keras offload IPsec dan model pemrograman yang berbeda. QorIQ memiliki satu mesin keamanan kaya fitur yang menyediakan layanan keamanan atas nama semua inti prosesor umum (delapan PowerPC untuk produk P4080). Mesin keamanan P4080

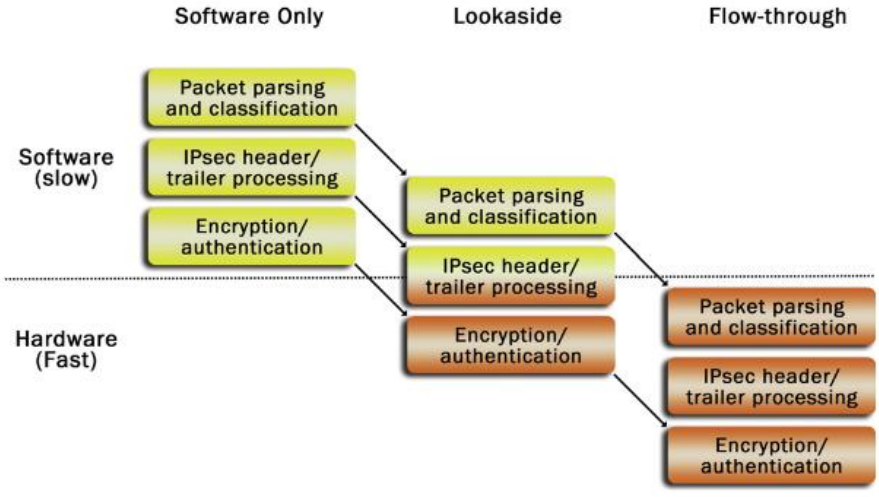
[16] NIST Special Publication 800-57. Recommendation for Key ManagementdPart 1: General (Revised); March 2007.
[17] The Internet Key Exchange. Internet Engineering Task Force, Request for Comments: 2409; November 1998.
[18] Internet Key Exchange (IKEv2) Protocol. Internet Engineering Task Force, Request for Comments: 4306; December 2005
[19] Internet Key Exchange Protocol Version 2 (IKEv2). Internet Engineering Task Force, Request for Comments: 5996; September 2010.

memberikan kemampuan flow-through, yang berarti mesin offload dapat menangani pemrosesan IPsec massal dengan sedikit atau tanpa intervensi inti CPU. Setelah materi kunci IKE dibuat dan diprogram ke dalam mesin, ia akan menjalankan semua algoritma kriptografi (enkripsi, dekripsi, dan perlindungan integritas) serta pemrosesan header IPsec, seperti menambahkan header dan trailer paket IPsec ESP. P4080 menyediakan dukungan offload untuk protokol keamanan jaringan lainnya, termasuk TLS.

Cavium OCTEON memiliki prosesor keamanan offload yang terpasang di setiap inti tujuan umum (hingga 16 untuk keluarga OCTEON 58xx). Namun, OCTEON menyediakan pemrosesan lookside, bukan flow-through. Ini berarti bahwa mesin membongkar pemrosesan enkripsi kelas berat tetapi harus bergantung pada inti aplikasi utama untuk tugas protokol IPsec lainnya, seperti pemrosesan header dan trailer serta penguraian dan klasifikasi paket (mengidentifikasi konfigurasi terowongan, atau asosiasi keamanan mana, yang harus digunakan untuk paket masuk). Kemanjuran pendekatan mana pun bergantung pada aplikasi dan persyaratan sistem. Karena kekuatan pemrosesan IPsec-nya berskala seiring dengan jumlah inti, OCTEON mampu mencapai throughput agregat yang unggul. Namun, P4080 mampu mencapai kinerja luar biasa tanpa berdampak pada inti utama.

Banyak prosesor Arsitektur Intel yang menyertakan AES-NI, instruksi khusus untuk mempercepat AES. Sebagai pendekatan sampingan, AES-NI mengklaim manfaat tambahan dalam menghindari serangan saluran samping berbasis memori lokal yang telah mengganggu implementasi AES khusus perangkat lunak.

Gambar 5.9 mengilustrasikan perbedaan batasan perangkat keras/perangkat lunak antara implementasi perangkat lunak murni IPsec, lookaside, dan flow-through. Perhatikan bahwa beberapa akselerator memiliki garis yang ditarik di tempat yang sedikit berbeda. Misalnya, akselerator lookaside dapat menangani sebagian dari prosesor header IPsec atau tidak sama sekali. Lebih jauh lagi, perhatikan bahwa mesin flow-through mungkin memerlukan pertukaran kunci perangkat lunak melalui IKE, dan mesin lain mungkin juga menangani negosiasi IKE. IKE dihilangkan dari Gambar 5.9.



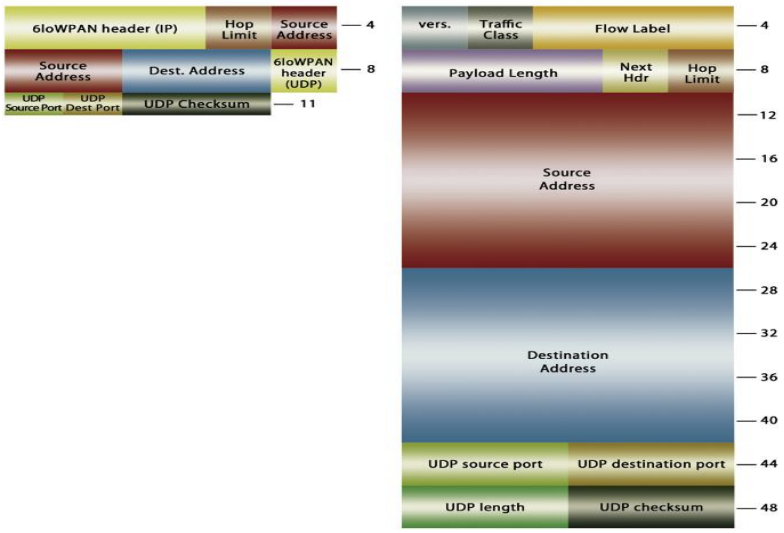
Gambar 5.9 Perbandingan pembongkaran IPSec.

IPsec untuk Perangkat Berdaya Sangat Rendah

Inisiatif energi cerdas global telah menimbulkan kekhawatiran mengenai kemampuan perusahaan listrik dan pemasoknya dalam menyediakan keamanan jaringan yang memadai untuk peralatan pintar generasi baru, meter pintar, gerbang, dan pusat data jaringan serta jaringan area rumah/pribadi (HAN). /PANs) dan jaringan area lingkungan (NAN) yang dikerahkan untuk menghubungkan mereka. IPv6 sangat penting karena keinginan untuk menghubungkan, pada akhirnya, miliaran titik akhir jaringan energi pintar. Namun, implementasi IKE/IPsec yang berfitur lengkap dan sesuai dengan RFC sangat canggih dan tidak cocok untuk sensor cerdas bertenaga baterai berbiaya rendah yang dipersenjatai dengan mikrokontroler dan flash puluhan kilobyte serta RAM yang berkomunikasi dengan daya rendah. Jaringan nirkabel berbasis 802.15.4 berukuran paket kecil.

Faktanya, perangkat berbasis Zigbee saat ini menggunakan keamanan lapisan tautan data dasar yang terbukti mudah untuk dibobol (misalnya, kunci simetris pribadi yang disediakan melalui jaringan secara jelas). Intinya, para perancang menegaskan bahwa perimeter keamanan paling baik diterapkan di gateway dan bahwa ancaman terhadap HAN atau bahkan NAN tidak perlu dikhawatirkan.

Namun ada pula yang mengabaikan sikap ini dan memikirkan masa depan di mana melindungi rumah dari peretas sangatlah penting. Salah satu standar yang menjanjikan untuk perangkat berdaya sangat rendah adalah 6LoWPAN (RFC 4944), yang menetapkan skema kompresi untuk IPv6 (dan implikasinya, IPsec) untuk memungkinkan penggunaannya melalui jaringan 802.15.4.[20] Sayangnya, 6LoWPAN (dan jaringan IP secara umum) tidak kompatibel dengan ZigBee generasi saat ini; namun, 6LoWPAN ditentukan dalam ISA 100.11a,[21] sebuah standar terbuka untuk komunikasi nirkabel dalam otomasi industri dan sistem kontrol proses, dan Dokumen Persyaratan Teknis ZigBee Smart Energy Profile 2.0[22], yang dimaksudkan untuk menghasilkan spesifikasi ZigBee Smart Energy Profile yang baru.



Gambar 5.10 Perbandingan ukuran header 6LoWPAN dan IPv6 standar untuk paket yang dapat dirutekan.

[20] Transmission of IPv6 Packets over IEEE 802.15.4 Networks. Internet Engineering Task Force, Request for Comments: 4944; September 2007.

[21] ISA-100.11a. Wireless Systems for Industrial Automation: Process Control and Related Applications. The International Society for Automation; 2009

[22] Zigbee Smart Energy 2.0 Technical Requirements Document, draft version 0.7. Zigbee Alliance; 2011.

Banyak aplikasi 802.15.4 menggunakan UDP, bukan TCP, untuk transportasi. Misalnya, sistem pemantauan energi mungkin menerima datagram periodik yang toleran terhadap penurunan yang dikirim dari sensor energi titik akhir. Seperti yang ditunjukkan pada Gambar 5.10, format paket UDP/IPv6 yang sesuai dengan 6LoWPAN memungkinkan kompresi berbagai bidang, termasuk alamat IPv6 sumber dan tujuan (masing-masing 128 bit, pelanggar jejak kaki terburuk), versi IP (enam, tersirat), panjang muatan (disimpulkan dari lapisan data link), kelas lalu lintas, label aliran, dan port UDP sumber dan tujuan.

Dalam kasus terbaik dari paket non-routable (lokal), 6LoWPAN dapat mengurangi ukuran header UDP/IPv6 menjadi hanya 4 byte (relatif terhadap 48 byte biasa); kasus terbaik untuk paket yang dapat dirutekan menghasilkan ukuran header 11 byte. Sumber informasi yang bagus tentang 6LoWPAN dapat ditemukan di 6LoWPAN, The Wireless Embedded Internet, oleh Shelby dan Bormann[23].

HAIPE

Pada tahun 2004, NSA memilih untuk membuat versi protokol IPsec yang disesuaikan karena masalah keamanan dengan standar komersial. Standar High-Assurance Internet Protocol Encryptor (HAIPE) (setidaknya ini adalah standar NSA) digunakan untuk sistem jaringan pribadi virtual (VPN) Tipe 1 yang bersertifikat NSA. Meskipun berbasis IKE dan IPsec, HAIPE tidak dapat dioperasikan. Meskipun IPsec tradisional adalah protokol point-to-point, HAIPE mendukung point-to-multipoint (multicast) menggunakan kunci grup yang telah ditentukan sebelumnya selain aplikasi point-to-point yang lebih umum.

Spesifikasi awal HAIPE hanya digunakan untuk algoritma rahasia (Suite A); namun, revisi berikutnya menambahkan opsi Suite B untuk memungkinkan interoperabilitas di masa mendatang antara produk Tipe 1 dan produk Suite B (tidak diklasifikasikan). Karena IKEv2 dan pedoman penggunaan kriptografi Suite B di IKEv2 dan IPsec, NSA telah menyatakan bahwa HAIPE pada akhirnya akan diganti dengan IKEv2/IPsec untuk semua produk Tipe 1 Suite B. Produk-produk ini akan berinteroperasi dengan peralatan komersial lainnya asalkan mereka juga mengikuti protokol IPsec yang diselaraskan dan menggunakan PKI umum.

5.2.6 SSL/TLS

Secure Sockets Layer ditemukan oleh Netscape dan memiliki tiga versi utama (SSL v1, v2, dan v3) sebelum digantikan oleh protokol Transport Layer Security (TLS) yang serupa namun lebih baik, yang sedang dalam revisi ketiga pada saat ini. menulis. Versi TLS ditentukan dalam RFC 2246 (Versi 1.0, 1999),[24] 4346 (Versi 1.1, 2006),[25] dan 5246 (Versi 1.2, 2008).[26] Mirip dengan IKE, TLS bertujuan untuk membuat koneksi aman antara dua rekan menggunakan asal otentikasi dan pembuatan kunci sesi. Mirip dengan IKE, berbagai revisi SSL dan TLS bertujuan untuk meningkatkan keamanan dan kegunaan berdasarkan masukan dari penggunaan secara luas. Namun demikian, produk IKE atau TLS yang sepenuhnya sesuai dengan RFC adalah paket perangkat lunak canggih yang rentan terhadap kesalahan implementasi dan kesalahan konfigurasi. Sekali lagi, cara terbaik adalah menggunakan revisi protokol terbaru dan terhebat

---

[23] Shelby Z, Bormann C. 6LoWPAN, The Wireless Embedded Internet. New York: Wiley; 2009.

[24] The TLS Protocol Version 1.0. Internet Engineering Task Force, Request for Comments: 2246; January 1999.

[25] The Transport Layer Security (TLS) Protocol Version 1.1. Internet Engineering Task Force, Request for Comments: 4346; April 2006.

[26] The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force, Request for Comments: 5246; August 2008.

dalam hal ini, TLS v1.2. Namun, mirip dengan IKE, berbagai varian TLS tidak sepenuhnya kompatibel dengan versi sebelumnya. Secara khusus, implementasi TLS yang hanya mendukung versi 1.2 tidak akan dapat terhubung dengan implementasi yang tidak mendukung versi 1.2. Seperti IPsec, pemrosesan kriptografi TLS dapat dipindahkan ke mesin keamanan jika ada.

OpenSSL

OpenSSL adalah SSL/TLS open source yang populer dan perangkat algoritma kriptografi. Port OpenSSL tersedia untuk sebagian besar sistem operasi tertanam modern, termasuk Linux dan INTEGRITY. Pada saat penulisan ini, TLS 1.2 belum didukung di OpenSSL. Oleh karena itu, jika perangkat tertanam suatu organisasi harus terhubung dengan rekan OpenSSL, maka perangkat tersebut mungkin terpaksa menggunakan TLS 1.1.

OpenSSL akan segera mendukung TLS 1.2, bahkan mungkin pada saat buku ini diterbitkan. OpenSSL adalah contoh bagus mengapa pengembang sistem tertanam tidak boleh berasumsi bahwa produk aman hanya karena pemasoknya menyatakan demikian atau bahkan karena produk tersebut didedikasikan khusus untuk fungsi keamanan. Penelusuran pada Basis Data Kerentanan Nasional CERT AS akan menunjukkan lebih dari 100 kerentanan perangkat lunak di OpenSSL selama bertahun-tahun, termasuk lima yang diterbitkan pada tahun 2011. Sekitar seperempat dari kelemahan perangkat lunak yang dipublikasikan hingga saat ini tergolong parah. Misalnya, CVE-2007-4995: “kesalahan satu per satu dalam implementasi DTLS di OpenSSL 0.9.8 sebelum 0.9.8f memungkinkan penyerang jarak jauh mengeksekusi kode arbitrer.”

Pengembang sistem tertanam yang menggabungkan OpenSSL ke dalam produk harus memverifikasi bahwa versi yang digunakan telah menerapkan patch keamanan terbaru. Jika versi TLS dengan ketahanan tinggi diperlukan, hubungi pemasok perangkat lunak tertanam Anda untuk mendapatkan opsi tersertifikasi yang didukung secara komersial.

Toolkit OpenSSL terdiri dari dua komponen utama: libssl, implementasi protokol SSL, TLS, dan DTLS; dan libcrypto, perpustakaan kriptografi berfitur lengkap yang dapat, dan sering kali, digunakan secara independen dari protokol terkait SSL. Toolkit OpenSSL membangun komponen-komponen ini menjadi pustaka gabungan (biasanya pustaka tautan bersama atau dinamis) serta alat baris perintah untuk menjalankan perintah berguna seperti membuat kunci kriptografi, menjalankan tolok ukur, dan mengenkripsi konten file.

Untuk pengembang sistem tertanam produk dengan memori terbatas, penting untuk memahami konsekuensi jejak memori dari penggabungan paket protokol keamanan jaringan. Dalam kasus OpenSSL, jejak statis libssl (hanya kode dan data, tidak termasuk alokasi memori dinamis) berukuran sekitar 200 KB. Pustaka algoritma kriptografi, libcrypto, berukuran sekitar 1 MB. OpenSSL yang dapat dieksekusi, yang menggabungkan fungsi perpustakaan standar lainnya, memiliki ukuran sekitar 1,5 MB. Angka-angka ini berubah dari satu versi ke versi lainnya seiring dengan perubahan protokol, penambahan algoritma, dan seterusnya. Dan, tentu saja, jumlahnya bervariasi berdasarkan arsitektur CPU target dan kompiler yang digunakan. Selain itu, karena kode sumber tersedia, pengembang dapat menyesuaikan perpustakaan untuk menghapus bagian yang tidak diperlukan dan mengurangi jejak. Sayangnya, API standar terbuka tidak tersedia untuk SSL atau TLS atau protokol perlindungan

data lainnya yang dibahas dalam bab ini. OpenSSL API didokumentasikan di situs web www.openssl.org.

GnuTLS

GnuTLS adalah pustaka SSL/TLS sumber terbuka lainnya. Berbeda dengan lisensi gaya BSD OpenSSL, GnuTLS menggunakan Lesser General Public License (LGPL). GnuTLS mendukung standar terbaru, termasuk DTLS dan TLS 1.2, dan menyertakan perpustakaan kriptografi.

Kami menggunakan GnuTLS untuk mengilustrasikan salah satu cara TLS diintegrasikan ke dalam kode jaringan pada umumnya. Alur umum di GnuTLS bagi klien untuk membuat koneksi soket aman ke server TLS jarak jauh yang kompatibel adalah sebagai berikut:

1. Gunakan antarmuka soket standar untuk membuat koneksi yang tidak terlindungi.
2. Buat saluran terlindungi di atas soket asli yang tidak terlindungi menggunakan jabat tangan TLS.
3. Gunakan API soket TLS untuk mengirim dan menerima paket pada saluran yang dilindungi.

Fragmen kode IPv4 berikut menunjukkan bagaimana GnuTLS digunakan bersama dengan API soket standar untuk membuat sesi yang dilindungi TLS. Panggilan dan tipe GnuTLS API ditampilkan dalam huruf tebal:

```
int my_tls_connect(struct addrinfo *dest, gnutls_session_t session)

int fd =socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
... /* some more initialization here */
if (fd < 0)
return -1;
if (connect(fd, dest->ai_addr, dest->ai_addrlen) < 0)
return -1;
/* fd: standard, unprotected socket connection */
/* associate the original socket with a TLS session */
gnutls_transport_set_ptr(session, (gnutls_transport_ptr_t)fd);
/* perform the TLS handshake */
if (gnutls_handshake(session) < 0)
return -1;
/* Now session refers to an established TLS session */
return 0;
}
/* The TLS analog to a socket send() */
void my_tls_send(gnutls_session_t s. void *data. uint32_t len)
{
  gnutls_record_send(s, data, len);
}
/* The TLS analog to a socket recv() */
int my_tls_recv(gnutls_session_t s, void *data, uint32_t maxlen)
{
  return gnutls_record_recv(s, data, maxlen);
}
-
```

Meskipun sangat disayangkan bahwa kurangnya standarisasi TLS API memastikan bahwa kode sebelumnya tidak akan berfungsi dengan implementasi TLS lainnya, kode tersebut menunjukkan bahwa urutan tipikal tidak rumit. Faktanya, pengembang dapat menyusun kode mereka untuk mengabstraksi panggilan dan tipe yang bergantung pada implementasi ke dalam modul portasi tunggal yang mudah untuk ditukar masuk dan keluar berdasarkan perpustakaan yang dipilih.

Dalam praktiknya, kode TLS ini juga perlu menyertakan beberapa panggilan inisialisasi global dan per sesi serta panggilan untuk menentukan metode autentikasi (berbasis sertifikat X.509, kunci yang dibagikan sebelumnya, dll.) dan parameter konfigurasi lainnya. Dokumentasi rinci untuk GnuTLS dapat ditemukan di Internet publik di www.gnutls.org.

5.2.7 Klien VPN Tertanam

Jaringan pribadi virtual (VPN) memberikan kerahasiaan data, integritas, dan autentikasi asal di seluruh jaringan yang tidak tepercaya seperti Internet. Kami menggunakan VPN untuk menghubungkan laptop kami menggunakan hotspot Wi-Fi terbuka di kafe ke intranet perusahaan kami. Teknologi VPN dapat digunakan untuk memungkinkan sistem tertanam dikelola dengan aman dari jarak jauh di seluruh jaringan heterogen (selama sistem ini mengikuti atau dapat melakukan terowongan Internet Protocol). Mengingat uraian ini, tidak mengherankan jika VPN sering diimplementasikan dengan IKE/IPsec. Klien VPN IPsec pada umumnya akan menggabungkan kode protokol IKE sambil memanfaatkan fungsionalitas IPsec yang dibangun ke dalam sistem operasi yang mendasarinya. Jika sistem operasi yang mendasari suatu organisasi tidak mendukung IPsec, maka organisasi tersebut harus mengadopsi teknologi VPN yang menggabungkan IKE dan IPsec.

Kurangnya API atau kerangka kerja standar untuk IKE dan IPsec telah menyebabkan terciptanya aplikasi klien VPN untuk merangkum beberapa konfigurasi yang membosankan, seperti mengidentifikasi alamat IP server dan informasi otentikasi, yang terkait dengan protokol ini. Aplikasi VPN juga menggabungkan berbagai fungsi tambahan lainnya yang terkait dengan IKE/IPsec, termasuk dukungan untuk penguraian dan verifikasi sertifikat digital (misalnya X.509), penggunaan panggilan sistem di belakang layar untuk mengkomunikasikan materi kunci yang dibuat oleh IKE. ke tumpukan IPsec, dan antarmuka ke metode otentikasi tambahan dan perangkat keras kriptografi seperti kartu pintar.

Pembaca tidak boleh tertipu dengan berpikir bahwa memasukkan aplikasi VPN ke dalam sistem tertanam adalah hal yang sepele. Pengembang harus bersiap untuk membaca banyak dokumentasi mengenai cara membangun dan mengkonfigurasi perangkat lunak VPN. Sayangnya, pemasok klien VPN sendiri hanya melakukan sedikit upaya untuk melakukan standarisasi API yang akan digunakan oleh pengembang sistem tertanam untuk menggabungkannya. Untuk sistem yang dihadiri, aplikasi klien VPN mungkin menyertakan antarmuka pengguna grafis untuk konfigurasi dan pembuatan sesi. Untuk sistem tertanam tanpa pengawasan, beberapa bentuk kit pengembangan dan dokumentasi harus disediakan. Pengembang sistem tertanam harus mempertimbangkan untuk membuat API independen untuk menjalankan layanan VPN, mengisolasi perangkat lunak aplikasi dari pilihan paket klien VPN.

Openswan dan Racoon2

Openswan adalah contoh paket VPN IPsec yang dapat digunakan dalam mode klien atau server. Openswan adalah produk open source yang populer untuk sistem operasi Linux. Semula, Openswan menyertakan patch IPsec ke kernel Linux serta daemon mode pengguna IKE, yang disebut pluto. Namun pada Linux 2.6, IPsec dimasukkan ke dalam distribusi kernel standar dan didukung dengan baik, mengurangi Openswan menjadi IKE ruang pengguna dan layanan pembuatan koneksi serta konfigurasi terkait. Openswan telah diuji interoperabilitasnya dengan berbagai teknologi VPN terbuka dan eksklusif, seperti Cisco dan Windows. Pada saat penulisan ini, informasi dan perangkat lunak Openswan dapat ditemukan di Internet di www.openswan.org. Implementasi Openswan modern akan memerlukan sekitar 300 KB kode dan penyimpanan data (tapak statis), tidak termasuk algoritma kriptografi yang dibutuhkan oleh IKE.

Banyak buku VPN telah ditulis, seperti VPNs Illustrated: Tunnels, VPNs, dan IPsec karya Jon Snader.[27] Sebuah buku tentang Openswan yang ditulis oleh pengembangnya juga tersedia.[28]

Sebagian besar Openswan diatur oleh GPLv2. Pengembang yang mencari alternatif berlisensi yang lebih ramah sebaiknya mempertimbangkan Racoon2, yang menggunakan lisensi gaya BSD. Racoon2 menyediakan implementasi IKE (v1 dan v2) dan telah diadaptasi untuk digunakan dengan tumpukan IPsec di berbagai sistem operasi, termasuk BSD, Linux, dan MacOS. Pada

OpenVPN

OpenVPN adalah teknologi VPN sumber terbuka lainnya. Namun, tidak seperti Openswan, OpenVPN menggunakan varian SSL/TLS untuk membuat terowongan dan oleh karena itu memerlukan rekan OpenVPN yang kompatibel. OpenVPN menggunakan kriptografi OpenSSL secara ekstensif. Karena seluruhnya terdiri dari kode ruang pengguna, OpenVPN dianggap lebih mudah untuk di-porting ke beberapa sistem operasi, yang telah dilakukan dalam praktiknya, tidak seperti Openswan, yang hanya diturunkan ke sistem Linux.

OpenVPN adalah contoh dari apa yang disebut sebagai teknologi SSL VPN, yang mana terdapat banyak produk komersial berpemilik. Karena mereka menggunakan protokol SSL/TLS (dan banyak yang mendukung DTLS untuk lalu lintas UDP, dijelaskan di bagian berikut), VPN dapat dibuat pada tingkat aplikasi, sehingga mengurangi hak istimewa dan overhead dibandingkan dengan VPN IPsec yang membuka keseluruhan pipa jaringan untuk semua aplikasi yang berjalan pada sistem. Selain itu, SSL VPN lebih mudah dipasang dan dikonfigurasi.

Sayangnya, sebagian besar implementasi SSL VPN, termasuk OpenVPN, tidak dapat dioperasikan. Keputusan untuk menggunakan klien Juniper SSL VPN kemungkinan memerlukan penerapan peralatan konsentrator Juniper SSL VPN. Klien OpenVPN hanya berbicara dengan rekan OpenVPN di bagian belakang.

---

[27] Snader JC. VPNs Illustrated: Tunnels, VPNs, and IPsec. Addison-Wesley Professional; November 2005.

[28] Bantoft K, Wouters P. Openswan: Building and Integrating Virtual Private Networks: Learn from the Developers of Openswan How to Build Industry Standard, Military Grade VPNs and Connect Them with Windows, MacOSX, and Other VPN Vendors. Birmingham, UK: Packt Publishing; February 2006.

5.2.8 DTLS

Datagram TLS (DTLS) dirancang untuk memberikan kemampuan yang sama seperti TLS untuk protokol transport berbasis UDP dan aplikasi yang menggunakannya.

DTLS diterbitkan pada tahun 2006 dan didefinisikan dalam RFC 4347[29]. DTLS semakin populer karena meningkatnya penggunaan protokol streaming suara dan video yang dapat mentolerir paket yang dijatuhkan dan lebih memilih overhead UDP yang lebih rendah daripada TCP. Sayangnya, DTLS, sebagian besar, harus menemukan kembali roda yang dibangun oleh IPsec untuk menangani paket yang terjatuh dan rusak yang dapat terjadi selama protokol jabat tangan TLS. Oleh karena itu, DTLS menambah komplikasi pada protokol TLS yang sudah kompleks. Namun demikian, seperti halnya TLS, DTLS adalah pilihan yang baik ketika sistem belum menyertakan tumpukan IPsec atau ketika ada keinginan untuk mengaktifkan kontrol keamanan jaringan tingkat aplikasi yang terperinci.

5.2.9 SSH

Secure Shell (SSH), sesuai dengan namanya, pada awalnya dirancang untuk menyediakan akses jarak jauh yang aman ke komputer oleh pengguna atau komputer lain. Aplikasi asli SSH menggantikan protokol lama seperti telnet, rlogin, dan rsh yang mengandalkan transmisi teks biasa tanpa otentikasi kriptografi. Pada dasarnya, SSH menambahkan otentikasi kunci publik dan perlindungan transit data ke sesi soket TCP. Hasilnya adalah cara yang lebih aman bagi pengguna dan komputer untuk membuat sesi interaktif yang aman dengan komputer jarak jauh lainnya. Aplikasi khas SSH dalam sistem tertanam adalah menjalankan server SSH dalam sistem tertanam dan memungkinkan administrator jarak jauh untuk mengelola sistem tertanam menggunakan protokol SSH dan perintah shell khusus sistem.

Sejarah SSH menunjukkan sulitnya mencapai tingkat keamanan yang tinggi dalam protokol perlindungan data yang canggih. SSH awalnya dikembangkan pada tahun 1995 dan dalam waktu 5 tahun telah memiliki jutaan pengguna di seluruh dunia. Namun 13 tahun setelah SSH didirikan, kerentanan ditemukan di semua versi SSH. Kerentanan ini memungkinkan penyerang memulihkan hingga 32 bit teks biasa dari blok teks tersandi sembarang ketika mode CBC (lihat Bab 4) digunakan untuk enkripsi.[30] Kerentanan ini ada bahkan di versi terbaru SSH, SSH-2 . Tidak ada versi standar terbaru yang tidak memiliki kerentanan ini. Solusi yang dipublikasikan adalah dengan menggunakan mode RKT sebagai pengganti mode CBC.

Beberapa kerentanan parah pada versi SSH sebelumnya ditemukan selama bertahun-tahun sebelum kelemahan CBC ini. Faktanya, perbaikan bug yang dilakukan untuk mengatasi kerentanan dalam protokol versi SSH-1 sebenarnya memperkenalkan kerentanan baru dan serius yang memungkinkan penyerang jarak jauh mengeksekusi kode arbitrer di dalam server SSH, yang dijalankan dengan hak akses root pada banyak sistem operasi tujuan umum. sistem.[31]

---

[29] Datagram Transport Layer Security. Internet Engineering Task Force, Request for Comments: 4347; April 2006.
[30] SSH CBC Vulnerability, US-CERT Vulnerability Note VU#958563.
[31] SSH CRC32 Attack Detection Code Contains Remote Integer Overflow, US-CERT Vulnerability Note VU#945216.

Tidak seperti banyak protokol jaringan lain yang didefinisikan secara ringkas dalam satu atau dua RFC, protokol SSH-2 ditentukan oleh lima standar inti: RFC 4250,[32] RFC 4251,[33] RFC 4252,[34] RFC 4253,[35] dan RFC 4254.[36] Kompleksitas standar ini mungkin menunjukkan kompleksitas protokol dan kesulitan keamanan historisnya.

Protokol Tambahan Berbasis SSH

SSH telah digunakan sebagai dasar untuk membuat versi aman dari aplikasi komunikasi populer lainnya yang melibatkan akses komputer jarak jauh oleh pengguna dan komputer lain. Ini termasuk *SSH File Transfer Protocol* (SFTP), yang secara konseptual mirip dengan *File Transfer Protocol* (FTP), dan *Secure Copy Protocol* (SCP), sebuah pendekatan keamanan yang ditingkatkan untuk protokol penyalinan jarak jauh (RCP). SFTP dijelaskan dalam rancangan RFC yang sudah kadaluwarsa,[37] tetapi tidak ada RFC aktif yang mendefinisikan protokolnya. Demikian pula, tidak ada RFC untuk SCP, sehingga interoperabilitas diserahkan kepada vendor (biasanya bukan ide yang bagus).

OpenSSH

Mungkin salah satu penjelasan atas kurangnya standarisasi seputar penggunaan SSH ini adalah keberadaan OpenSSH di mana-mana, sebuah implementasi SSH open source yang mencakup implementasi sisi klien dan server serta dukungan SFTP dan SCP. Pengembangan OpenSSH dipimpin oleh komunitas OpenBSD dan menikmati lisensi BSD yang liberal. OpenSSH memiliki perpustakaan kriptografi yang lengkap dan telah di-porting ke berbagai sistem operasi tertanam. Pengembang mengharapkan klien SSH memerlukan penyimpanan ROM/flash sekitar 300 KB untuk kode dan data yang dapat dieksekusi.

Protokol Keamanan Jaringan Khusus

Karena kompleksitas dan jejak besar IPsec dan TLS, sejumlah pemasok perangkat lunak komersial dan peneliti universitas telah mengusulkan protokol keamanan jaringan alternatif yang lebih ringan; bagi perancang sistem tertanam yang lebih mementingkan kesederhanaan dan jejak serta tidak terlalu peduli dengan interoperabilitas dengan rekan-rekan yang sewenang-wenang, protokol khusus mungkin merupakan alternatif yang tepat. Sangat penting untuk mendapatkan jaminan bahwa para ahli kriptografi telah menganalisis dan memvalidasi protokol khusus secara independen.

Mari kita lihat contoh penyederhanaan SSL berdasarkan area protokol yang terbukti rentan. SSL memungkinkan rekan-rekan untuk menegosiasikan rangkaian sandi untuk memastikan interoperabilitas. Hal ini masuk akal: jika server mendukung 3DES dan AES untuk enkripsi massal dan klien hanya mendukung 3DES, maka negosiasi ciphersuite memungkinkan rekan untuk menyimpulkan algoritma 3DES yang umum sehingga komunikasi dapat terjadi. Namun, administrator keamanan yang menyebarkan SSL di server mungkin menginginkan

---

[32] The Secure Shell (SSH) Protocol Assigned Numbers. Internet Engineering Task Force, Request for Comments: 4250; January 2006

[33] The Secure Shell (SSH) Protocol Architecture. Internet Engineering Task Force, Request for Comments: 4251; January 2006.

[34] The Secure Shell (SSH) Authentication Protocol. Internet Engineering Task Force, Request for Comments: 4252; January 2006.

[35] The Secure Shell (SSH) Transport Layer Protocol. Internet Engineering Task Force, Request for Comments: 4253; January 2006.

[36] The Secure Shell (SSH) Connection Protocol. Internet Engineering Task Force, Request for Comments: 4253; January 2006

[37] SSH File Transfer Protocol. Internet-Draft from the Secure Shell Working Group

perlindungan AES 256-bit dan tidak menyadari bahwa klien mungkin menegosiasikan server ke algoritma yang lebih lemah seperti 3DES. Selain itu, serangan man-in-the-middle telah berhasil memaksa penurunan peringkat ciphersuite antar peer. Salah satu contohnya mempengaruhi sebagian besar versi pengiriman OpenSSL dan disebut sebagai CVE-2010-4180 di NIST National Vulnerability Database.[38]

Kerentanan lain dari SSL mempengaruhi negosiasi ulang, sebuah fitur protokol yang memungkinkan rekan meminta pembentukan kembali (mengautentikasi dan membuat kunci bersama baru) dalam sesi yang telah ditetapkan sebelumnya. Kerentanan ini, yang disebut sebagai CVE-2009-3555,[39] memungkinkan penyerang man-in-the-middle memasukkan teks berbahaya selama sesi yang valid antara dua rekan yang tidak menaruh curiga. Karena penyerang tidak dapat membaca teks terenkripsi sesi tersebut, kerentanan ini pada awalnya dianggap relatif tidak berbahaya. Namun, para peneliti segera menemukan cara untuk menggunakan teks yang dimasukkan untuk membujuk server web agar mengungkapkan data sensitif. Secara khusus, serangan yang berhasil dilakukan terhadap Twitter: teks yang dimasukkan sebenarnya menginstruksikan Twitter untuk secara jahat menge-tweet informasi nama pengguna dan kata sandi pengguna yang mengirim tweet. Penyerang kemudian mengambil tweet terlarang tersebut dan dapat menggunakan informasi yang dicuri untuk mengambil identitas pengguna yang tidak menaruh curiga.

Penyederhanaan potensial terhadap SSL adalah menghilangkan kompleksitas seputar negosiasi rangkaian kriptografi antar rekan serta kemampuan untuk melanjutkan atau menegosiasikan ulang sesi. Bayangkan perancang sistem tertanam ingin dilindungi dengan serangkaian cipher suite tertentu, seperti tanda tangan digital kurva elips 384-bit untuk otentikasi dan perjanjian kunci serta AES-GCM 256-bit untuk enkripsi terautentikasi semua data dalam transit. Implementasi SSL klien dapat memulai sesi yang menentukan rangkaian ini dan tidak mengizinkan sesuatu yang lebih lemah. Klien akan menggunakan kunci enkripsi sesi yang berhasil dibuat untuk periode kripto yang ditentukan klien dan kemudian mengakhiri sesi secara permanen pada akhir periode ini. Tidak ada sesi dimulainya kembali atau negosiasi ulang yang diizinkan. Jika rekan perlu mengubah asosiasi keamanan, ia cukup menutup soket dan memulai ulang protokol. Meskipun pendekatan ini mungkin tidak cocok untuk konsentrator SSLVPN, pendekatan ini mungkin cocok untuk sistem tertanam dan mempunyai keuntungan dalam menyederhanakan kode sistem tertanam, mengurangi jejak, dan menghindari kelemahan yang sudah ditemukan dan mungkin kelemahan lain yang belum ditemukan.

Dengan enam versi dan kelemahan yang ditemukan secara aktif, SSL adalah contoh yang baik dari kegagalan menerapkan prinsip PHASE dalam meminimalkan kompleksitas. Seperti banyak perangkat lunak canggih, bahkan yang dibuat untuk menjalankan fungsi keamanan, kumpulan pembuat SSL tidak memiliki disiplin untuk mengorbankan fitur yang tidak perlu demi menciptakan desain yang lebih sederhana, bersih, dan pada akhirnya lebih aman. Menarik untuk dicatat bahwa, sebagai respons terhadap kerentanan negosiasi ulang,

[38] National Institute of Standards and Technology National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/detail?vulnId¼CVE-2010-4180.

[39] National Institute of Standards and Technology National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/detail?vulnId¼CVE-2009-3555

solusi yang diterapkan pada server web adalah dengan menonaktifkan fitur negosiasi ulang dalam implementasi SSL yang mendasarinya. Solusi ini telah diterapkan secara luas, membuktikan bahwa fitur tersebut tidak diperlukan. Namun demikian, pembaca mungkin merasa lucu bahwa solusi utama IETF terhadap masalah ini, alih-alih menghentikan negosiasi ulang dan menyederhanakan protokol, adalah dengan menambahkan fitur lain ke SSL (RFC 5746) yang menghubungkan jabat tangan negosiasi ulang dengan sesi aslinya, sehingga mencegah terjadinya perselisihan. serangan man-in-the-middle.[40]

Mari kita pertimbangkan lingkungan rumah yang dilengkapi dengan meteran pintar yang berkomunikasi dengan gerbang jaringan area lingkungan yang dirancang dan dipasok oleh perusahaan yang sama dengan meteran pintar tersebut. Ini adalah situasi dimana protokol yang disesuaikan dan disederhanakan, yang patuh namun belum tentu sesuai dengan RFC, mungkin sesuai. Sebagai alternatif, sebuah komite yang bertugas menciptakan standar keamanan untuk jaringan pintar dapat mengadopsi penyesuaian protokol ini. TLS hanyalah salah satu contoh; penyederhanaan dan perbaikan serupa pada protokol IKE dan IPsec dapat dilakukan.

Jika desain tertanam memiliki tapak yang ketat atau memerlukan implementasi dengan jaminan lebih tinggi dan dapat menoleransi kurangnya kesesuaian dan interoperabilitas RFC penuh, pengembang harus berkonsultasi dengan sistem operasi dan/atau penyedia perangkat lunak sistem untuk mendiskusikan alternatif.

5.2.10 Kerahasiaan Arus Lalu Lintas

Seperti disebutkan sebelumnya, mode Tunnel IPsec ESP memberikan peningkatan keamanan transmisi dengan merangkum informasi pengalamatan paket IP asli dalam header baru yang biasanya berhubungan dengan gateway yang menghadap ke jaringan besar. Penyadap dapat menentukan jaringan asal dan tujuan tetapi tidak dapat menentukan identitas atau penargetan spesifik paket ke node individual di belakang gateway.

Contoh lain dari keamanan transmisi adalah hilangnya informasi arus lalu lintas. Arus lalu lintas mencakup kecepatan dan ukuran masing-masing paket. Penyadap mungkin bisa mendapatkan akses tidak sah terhadap informasi tentang sifat komunikasi dengan mengamati arus lalu lintas.

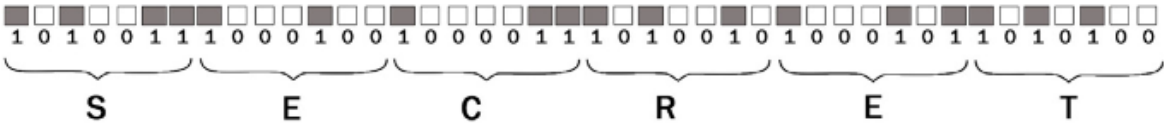
Misalnya, node berbahaya yang disusupi dalam jaringan dapat mengirimkan informasi sensitif ke penyadap yang berkolusi dengan menggunakan saluran pengaturan waktu jaringan rahasia. Mode yang dikompromikan mentransmisikan nilai numerik, N, dengan mengirimkan rangkaian N paket cepat yang diawali dan diikuti dengan penundaan yang lama. Nilai numerik tertentu dapat dipilih untuk menandakan awal dan akhir pesan rahasia baru, yang dibangun dari urutan paket ini.

Pendekatan terkenal lainnya melibatkan aplikasi jahat yang mengirimkan muatan dengan ukuran tertentu untuk menyampaikan informasi. Misalnya, para pembuat kolusi dapat menafsirkan muatan yang berukuran kurang dari 50 byte sebagai biner nol dan muatan yang lebih besar dari 50 byte sebagai biner. Ukuran payload khusus yang telah diatur sebelumnya dapat digunakan untuk memberi sinyal awal dan akhir pesan rahasia baru, yang dibangun dari

[40] Transport Layer Security (TLS) Renegotiation Indication Extension. Internet Engineering Task Force, Request for Comments: 5746; February 2010.

urutan paket ini. Serangan ukuran paket ditunjukkan pada Gambar 5.11, di mana ukuran setiap paket mengkomunikasikan angka 0 atau 1, menghasilkan pesan ASCII "SECRET."

Dalam kedua contoh ini, enkripsi muatan dari protokol keamanan jaringan tidak melakukan apa pun untuk mencegah orang dalam yang jahat mengungkap informasi sensitif di jaringan.



Gambar 5.11 Suatu bentuk komunikasi rahasia yang dimungkinkan oleh kurangnya kerahasiaan arus lalu lintas.

Meskipun standar terbuka saat ini untuk protokol keamanan jaringan, seperti TLS, DTLS, SSH, dan IPsec, tidak memberikan kerahasiaan arus lalu lintas penuh, beberapa pemasok perangkat lunak tertanam menyediakan versi protokol ini yang ditingkatkan untuk menyediakan fitur ini.

Misalnya, implementasi IPsec dapat memasukkan semua payload ke ukuran yang konstan, mencegah serangan panjang paket yang disebutkan di atas. Overhead padding tambahan mungkin dapat diterima atau bahkan diperlukan untuk beberapa lingkungan. Selain itu, pengembang produk tertanam yang menggabungkan fitur keamanan jaringan dapat memperoleh keunggulan kompetitif dengan menawarkan fitur kerahasiaan arus lalu lintas kepada pelanggan mereka.

5.2.11 Penerapan Kriptografi dalam Protokol Keamanan Jaringan

Tentu saja, protokol keamanan jaringan hanya seaman sandi yang mendasarinya. Konyol sekali menggunakan TLS untuk menegosiasikan kunci simetris lemah yang dapat dengan mudah dipatahkan dengan serangan brute force. Luasnya algoritma dan mode kunci publik, kunci simetris, dan hash/MAC yang digunakan dalam sistem tertanam saat ini sangat mencengangkan. Bab 4 memberikan gambaran umum mengenai semua bidang utama kriptografi serta panduan pemerintah untuk penerapan umumnya.

Panduan NSA Suite B

Sejumlah RFC telah diterbitkan yang menentukan persyaratan algoritma Suite B untuk digunakan dengan protokol keamanan jaringan populer. RFC 4869 mencakup persyaratan Suite B untuk IKE dan IPsec. RFC 5430 mencakup TLS. Fokus utama RFC ini adalah untuk menentukan cryptosuite yang dapat diterima (misalnya, AES dalam Mode Penghitung Galois dengan panjang kunci tertentu) untuk kepatuhan Suite B saat menggunakan protokol ini.

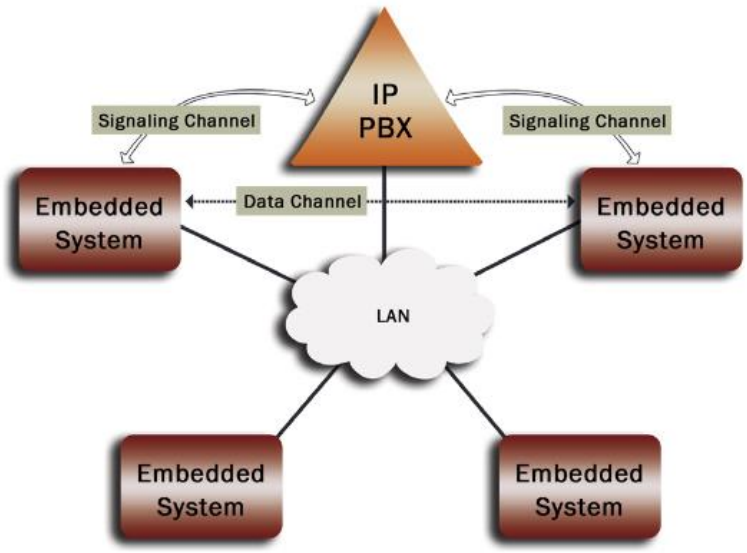
5.2.12 Protokol Multimedia yang Aman

Protokol multimedia mengatur transmisi data yang disebut real-time, bukan dalam pengertian hard real-time yang tertanam, melainkan dalam arti bahwa kualitas komunikasi bergantung pada minimalisasi kehilangan paket, latensi rendah, dan jitter. Contohnya termasuk audio (misalnya suara), video, pengambilan sampel lingkungan berkelanjutan, dan jenis aplikasi kendali dan manajemen jarak jauh tertentu.

Dalam beberapa kasus, NSA telah mengembangkan protokolnya sendiri karena masalah keamanan pada protokol komersial yang ada atau karena kurangnya protokol komersial untuk aplikasi tertentu. Untuk aplikasi suara yang aman (berkabel dan nirkabel), NSA mengembangkan protokol suara yang aman, *Secure Communications Interoperability Protocol* (SCIP), yang menetapkan prosedur pembentukan asosiasi yang aman untuk panggilan suara menggunakan serangkaian codec suara dan untuk digunakan dalam komunikasi point-to- aplikasi point dan point-to-multipoint. SCIP awalnya digunakan pada tahun 2001 dan sekarang diadopsi secara luas untuk peralatan suara bersertifikasi NSA Tipe 1 dan di seluruh NATO.

SCIP tidak digunakan dalam produk komersial dan karenanya tidak berguna bagi pengembang sistem tertanam (misalnya radio) yang digunakan di luar lingkungan sensitif pemerintah. Oleh karena itu, kami tidak membahas SCIP secara detail di sini. Penggunaan protokol non-komersial ini menimbulkan masalah bagi strategi interoperabilitas kriptografi NSA baru-baru ini (lihat Bab 4) yang berupaya memaksimalkan penggunaan standar komersial seperti IPsec dan algoritma kriptografi yang dipublikasikan. Oleh karena itu, kita dapat memperkirakan sikap NSA terhadap suara aman akan berubah dengan cepat, mendorong konversi dari SCIP ke protokol berbasis standar terbuka.

Dalam dunia komersial, suara yang aman menjadi semakin penting karena perusahaan telah mengadopsi penggunaan perangkat seluler untuk komunikasi sensitif dan apa yang disebut telepon lunak yang menggunakan Internet untuk transmisi suara. Keluarga umum protokol untuk transmisi suara melalui Protokol Internet disebut Voice over IP (VoIP).



Gambar 5.12 Jalur pensinyalan dan saluran data independen adalah hal yang umum dalam aplikasi suara yang aman.

VoIP mencakup banyak protokol berpemilik (misalnya, Skype) dan standar terbuka, yang pembahasan komprehensifnya tidak praktis dalam bab ini. Kami hanya membahas beberapa hal yang lebih penting, terutama yang cukup umum untuk mencakup transmisi multimedia umum (misalnya audio dan video). Protokol multimedia yang aman secara umum dapat dipecah menjadi dua bagian, protokol pensinyalan (juga disebut bagian kontrol) yang

bertanggung jawab untuk menemukan lokasi penerima (atau penerima dalam aplikasi multicast) dan membuat sambungan, dan protokol transportasi data yang bertanggung jawab untuk memindahkan data multimedia terenkripsi dari pengirim ke penerima. Dalam beberapa kasus, codec kompresi multimedia ditempatkan di dalam bidang data; algoritma ini berada di luar cakupan diskusi ini. Gambar 5.12 menunjukkan jalur pensinyalan dan jalur data dalam jaringan area lokal yang terhubung ke dunia luar melalui *private Branch Exchange* (PBX).

Protokol Persinyalan

Protokol bidang kendali multimedia mungkin cukup canggih, karena mereka bertanggung jawab untuk menemukan penerima yang sesuai, menegosiasikan kemampuan format media yang umum, melintasi dan melakukan terowongan di seluruh jaringan heterogen dan media komunikasi melalui router dan gateway, otentikasi titik akhir bersama, dan pembongkaran sesi. H.323 dan *Session Initiation Protocols* (SIP) adalah protokol persinyalan standar terbuka yang dominan digunakan saat ini. H.323 didefinisikan oleh sektor standardisasi telekomunikasi badan PBB untuk teknologi informasi dan komunikasi (ITU-T). Untuk informasi rinci tentang H.323, pembaca dianjurkan untuk membaca spesifikasi ITU-T untuk H.323 dan protokol tambahannya.[41]

Berbeda dengan H.323, yang dirancang untuk jaringan Circuit-Switched tradisional, SIP pada awalnya dirancang untuk jaringan IP. Banyak perbandingan H.323 dan SIP dapat ditemukan dalam literatur.[42] Meskipun basis instalasi H.323 besar, SIP tampaknya mengambil alih sebagai protokol yang dominan. Seperti banyak standar berbasis Protokol Internet lainnya, SIP ditentukan oleh IETF dalam hal ini, RFC 2543.[43] Bagi pembaca yang tertarik dengan betapa rumitnya protokol kontrol multimedia ini, SIP RFC memiliki panjang sekitar 270 halaman dibandingkan dengan IPsec RFC terbaru, 4301, yang panjangnya sekitar 100 halaman. SIP juga merupakan protokol bidang kendali pilihan untuk jaringan nirkabel seluler yang sedang berkembang, seperti LTE, karena keinginan untuk memaksimalkan kompatibilitas dengan standar Internet IETF.

Masalah keamanan untuk protokol persinyalan mencakup kemampuan untuk mencuri identitas titik akhir, penolakan layanan, keamanan transmisi (mendeteksi titik akhir mana yang berkomunikasi), dan non-penyangkalan (ketidakmampuan memverifikasi asal). Kekhawatiran ini muncul dari kenyataan bahwa SIP adalah protokol yang tidak terlindungi, rentan terhadap replay dan serangan man-in-the-middle lainnya. Kelemahan ini diatasi oleh standar IETF yang didedikasikan untuk eamanan SIP: RFC 3329.[44]

Banyak mekanisme ad hoc lainnya yang dapat digunakan untuk membuat sesi multimedia antar titik akhir pada jaringan IP. Pada dasarnya, ketika titik akhir dapat menentukan alamat IP tujuannya, maka saluran data dapat dibuat menggunakan mekanisme transport IP. Misalnya, aplikasi ponsel dapat menggunakan pesan SMS untuk menginterogasi pengguna bernama melalui jaringan seluler dan mendapatkan alamat IP aktif ponsel pengguna. Rincian sinyal SIP yang digunakan untuk menjalin interaksi SMS disembunyikan dari

---

[41] Recommendation ITU-T H.323. Packet-Based Multimedia Communications Systems. Telecommunications Standardization Section of ITU; December 2009

[42] Papageorgiou P. A Comparison of H.323 vs. SIP; June 4, 2001.

[43] SIP: Session Initiation Protocol. Internet Engineering Task Force, Request for Comments: 2543; June 2002.

[44] Security Mechanism Agreement for the Session Initiation Protocol (SIP). Internet Engineering Task Force, Request for Comments: 3329; January 2003.

aplikasi ponsel. Selain itu, aplikasi ponsel dapat menggunakan otentikasi tingkat aplikasinya sendiri melalui saluran data, tidak bergantung pada keamanan jaringan seluler yang mendasarinya.

Protokol Transportasi Multimedia

Setelah sesi dibuat, data multimedia harus dikirim secara efisien dan rahasia melalui link. Seperti disebutkan sebelumnya, otentikasi titik akhir saluran data sering kali diinginkan. Oleh karena itu, orang mungkin berpikir sesi TLS standar akan sesuai dengan kebutuhan. Namun, TLS berjalan di atas TCP, yang menimbulkan terlalu banyak latensi untuk aplikasi multimedia. Karena komunikasi multimedia umumnya toleran terhadap paket yang dijatuhkan sesekali, transport UDP berlatensi rendah sering digunakan. Oleh karena itu, orang mungkin berpikir bahwa DTLS akan sesuai dengan kebutuhannya. Faktanya, DTLS mungkin cocok untuk tautan jaringan bandwidth tinggi (misalnya, obrolan video dalam LAN). Namun, streaming multimedia sering kali berjalan melalui Internet dan/atau jaringan nirkabel area luas (misalnya LTE) yang bandwidthnya terbatas dan terputus-putus. Untuk alasan ini, protokol transport aman yang dioptimalkan diindikasikan[45].

SRTP

Real-Time Transport Protocol (RTP) adalah protokol kerangka kerja populer untuk transportasi multimedia. Kerangka kerja RTP dijelaskan dalam RFC 3550.[46] Penerapan kerangka kerja pada format informasi tertentu tercakup dalam standar pendamping. Misalnya, penerapan konferensi audio dan video distandarisasi dalam RFC 3551.[47]47 Standar ini mencakup banyak pengkodean audio dan video. Salah satu perbedaan utama antara RTP dan aliran UDP sederhana adalah penyertaan informasi kualitas layanan yang disampaikan ke titik akhir untuk tindakan perbaikan potensial dan informasi pengurutan yang digunakan untuk merekonstruksi aliran media dari paket yang rusak. Selain itu, RTP menangani multicasting beberapa sumber multimedia secara bersamaan dengan kecepatan yang bervariasi (multipoint-to-multipoint), membuat pekerjaan pengembang aplikasi menjadi lebih mudah.

Meskipun ada protokol transport multimedia non-standar yang dipatenkan, sebagian besar transportasi multimedia komersial dan sumber terbuka (terutama suara dan video) melalui protokol IP menggunakan RTP dan protokol keamanan tambahannya, Secure Real-Time Transport Protocol (SRTP).

SRTP didefinisikan dalam RFC 3711.[48] Pada tingkat tinggi, SRTP menyediakan layanan keamanan yang sama dengan protokol transport aman yang lebih umum untuk IP (IPsec, TLS, DTLS): otentikasi asal data, integritas (termasuk perlindungan pemutaran ulang), dan kerahasiaan. Tidak seperti IPsec, yang agnostik terhadap pilihan cipher enkripsi yang mendasarinya, SRTP menerapkan penggunaan mode cipher yang ramah terhadap multimedia. Secara khusus, SRTP menentukan variasi mode counter dan umpan balik keluaran yang,

[45] Schulzrinne H. Issues in Designing a Transport Protocol for Audio and Video Conferences and Other Multiparticipant Real-Time Applications. expired Internet Draft; October 1993.

[46] RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force, Request for Comments: 3550; July 2003.

[47] RTP Profile for Audio and Video Conferences with Minimal Control. Internet Engineering Task Force, Request for Comments: 3551; July 2003.

[48] The Secure Real-Time Transport Protocol (SRTP). Internet Engineering Task Force, Request for Comments: 3711; March 2004.

seperti dibahas dalam Bab 4, tidak menyebarkan kesalahan ketika sebuah paket dijatuhkan. SRTP menggunakan HMAC-SHA-1 untuk otentikasi pesan. Perbedaan penting lainnya antara SRTP dan IPsec yang disalurkan adalah bahwa SRTP mempertahankan header RTP asli, memungkinkan data kualitas layanan tetap bertahan dalam saluran terenkripsi (tentu saja dengan mengorbankan kerahasiaan lalu lintas).

SRTP, seperti IPsec tetapi tidak seperti TLS, tidak mencakup pembentukan kunci antara pihak-pihak yang berkomunikasi. Protokol pertukaran kunci populer yang digunakan dengan SRTP termasuk Multimedia Internet Keying (MIKEY) yang ditentukan dalam RFC 3830,[49] Deskripsi Sesi Deskripsi Protokol Keamanan untuk Aliran Media (SDES) yang ditentukan dalam RFC 4568, dan ZRTP yang ditentukan dalam RFC 6189.[50] SDES mendukung pesan SIP dan karenanya tidak cocok untuk aplikasi yang menggunakan mekanisme persinyalan lain. Demikian pula, MIKEY ditujukan untuk memberi sinyal jalur pertukaran kunci. Sebaliknya, ZRTP melakukan pertukaran kunci dalam saluran media dan oleh karena itu tidak bergantung pada mekanisme pensinyalan. Ada beberapa perdebatan mengenai apakah lebih masuk akal untuk melakukan pertukaran kunci dalam saluran sinyal (yang sudah membangun koneksi) atau tetap memisahkannya. Mengingat beragamnya jenis perangkat, format, dan jaringan yang menjadi ciri dunia digital modern, diperlukan pendekatan terpisah, dengan asumsi efisiensi yang memadai.

Pengembang aplikasi multimedia yang aman dapat menyediakan otentikasi dan enkripsi end-to-end tanpa khawatir tentang bagaimana pihak-pihak yang berkomunikasi terhubung. Namun, para pembaca yang tertarik didorong untuk meninjau tandingan untuk aplikasi tertentu, seperti VoIP.[51] Pembaca juga didorong untuk memeriksa RFC 5479,[52] sebuah dokumen informasi yang membahas berbagai pilihan manajemen kunci baik dalam bidang sinyal maupun dalam bidang media.

DTLS-SRTP

DTLS-SRTP, yang didefinisikan dalam RFC 5764,[53] menghadirkan hibrid di mana pihak yang berkomunikasi jalur sinyal terikat pada sertifikat yang digunakan dalam pembuatan kunci jalur media. Alasan mengapa hal ini penting adalah karena *Public Key Infrastructure* (PKI) mungkin tidak praktis di antara pihak-pihak yang berkomunikasi dalam semua hal. Dengan DTLS-SRTP, pihak-pihak yang berkomunikasi menggunakan sertifikat yang ditandatangani sendiri yang diikatkan pada identitas pemberi sinyal mereka. Jika Alice menghubungi Bob dan terhubung melalui bidang sinyal, maka Alice memercayai bidang sinyal tersebut dan sekarang ingin memastikan bahwa data suaranya dienkripsi dengan kunci yang hanya dimiliki oleh Bob (sebagaimana ditentukan oleh nomor teleponnya).

Pesawat pemberi sinyal meneruskan sertifikat, dan perjanjian kunci sebenarnya dilakukan oleh DTLS melalui saluran media. Salah satu ciri yang diinginkan dari DTLS-SRTP

---

[49] MIKEY: Multimedia Internet KEYing, Internet Engineering Task Force. Request for Comments: 3830; August 2004

[50] ZRTP: Media Path Key Agreement for Unicast Secure RTP. Internet Engineering Task Force, Request for Comments: 6189; April 2011

[51] Orrblad J. Alternatives to MIKEY/SRTP to Secure VoIP. ftp://ftp.it.kth.se/Reports/DEGREE- PROJECTREPORTS/ 050330-Joachim-Orrblad.pdf; March 2005.

[52] Requirements and Analysis of Media Key Management Protocols. Internet Engineering Task Force, Request for Comments: 5479; April 2009.

[53] Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP). Internet Engineering Task Force, Request for Comments: 5764; May 2010.

adalah bahwa DTLS adalah protokol pertukaran kunci umum yang digunakan di seluruh Internet untuk data berorientasi datagram (seperti multimedia). SRTP adalah protokol transport aman de facto untuk multimedia. Oleh karena itu, DTLS-SRTP bisa dibilang merupakan kombinasi standar protokol Internet aman end-to-end untuk multimedia yang paling mudah dipahami.

Mungkin mengejutkan, panduan NSA Suite B saat ini mengarah pada DTLS-SRTP. Apa yang membuat hal ini mengejutkan adalah kepercayaan yang melekat pada protokol terhadap domain persinyalan. Bayangkan seorang analis intelijen menyampaikan informasi rahasia ke dalam telepon pintar, berkomunikasi melalui jaringan seluler publik yang terhubung ke belahan dunia lain. Jelas, ini merupakan ide yang buruk, karena perusahaan telepon asing yang disusupi dapat mengarahkan data suara rahasia ke pendengar yang jahat.

Oleh karena itu, penggunaan DTLS-SRTP dalam domain rahasia mengasumsikan bahwa panggilan suara terhubung melalui jaringan pribadi, diisolasi dari jaringan telepon umum (PSTN) dan Internet global, atau bahwa bidang sinyal menerapkan otentikasi yang dianggap dapat dipercaya oleh NSA. . Jika sinyal tidak dapat dipercaya, maka protokol kepemilikan atau ZRTP akan diindikasikan. Perhatikan bahwa ZRTP memiliki opsi untuk menggunakan PKI penuh untuk otentikasi. Alternatifnya, pemerintah A.S. sedang mempertimbangkan penggunaan operator jaringan virtual seluler (MVNO) yang menyediakan domain sinyal pribadi yang diimplementasikan sebagai jaringan seluler virtual yang dilapiskan pada jaringan seluler fisik yang sudah ada.

Pada saat penulisan ini, panduan Suite B untuk penggunaan DTLS-SRTP masih dalam rancangan status RFC.[54] Rancangan standar tersebut menggantikan sandi data massal SRTP dengan rangkaian AES-GCM pilihan Suite B dan memerlukan kriptografi kurva elips, ECDSA, dan ECDH. , untuk algoritma autentikasi dan perjanjian kunci. Pengembang sistem tertanam yang membuat produk dengan aplikasi multimedia dan menargetkan penggunaan pemerintah AS (kendaraan otonom adalah contoh yang menonjol) disarankan untuk mempertimbangkan penggunaan DTLS-SRTP.

5.2.13 Keamanan Siaran

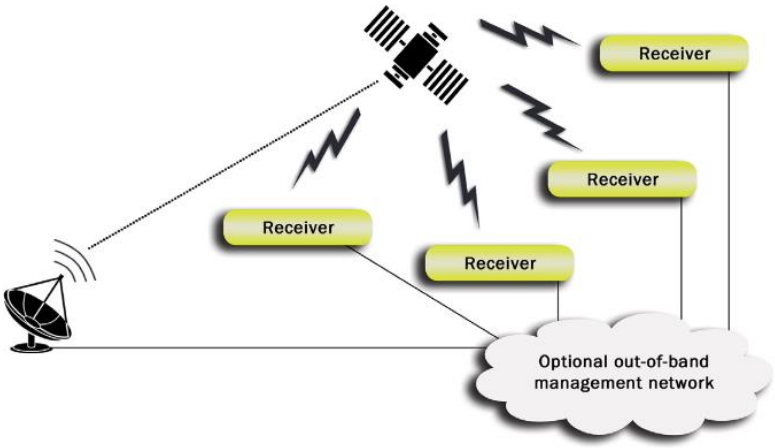
Sejauh ini, protokol keamanan jaringan yang dibahas dalam bab ini semuanya bergantung pada koneksi dupleks: pemancar dan penerima keduanya harus mengirimkan informasi untuk melengkapi protokol. Praktisnya semua protokol berbasis IP mengasumsikan lapisan data link yang mendasarinya yang secara inheren memungkinkan titik akhir mana pun untuk mengirim dan menerima data. Kebanyakan protokol middleware, seperti klien/server CORBA, juga menggunakan media komunikasi dupleks. Siaran, di sisi lain, adalah contoh komunikasi simpleks: semua informasi dikirim dalam satu arah, dari pengirim ke penerima. Contoh sistem yang memerlukan keamanan komunikasi siaran adalah sistem satelit digital yang mentransmisikan konten multimedia terlindungi ke penerima set-top box yang didistribusikan secara luas (lihat Gambar 5.13). Penerima dekoder tidak dapat mengirimkan data langsung kembali ke satelit.

Untuk lebih memperumit masalah, sistem penyiaran sering kali dicirikan oleh sekumpulan penerima yang besar dan dinamis serta kumpulan informasi dinamis yang perlu

[54] Suite B Profile for Datagram Transport Layer Security/Secure Real-Time Transport Protocol (DTLS-SRTP). Internet Engineering Task Force, Draft RFC; April 8, 2011

ditransmisikan, beroperasi pada jaringan fisik yang memiliki bandwidth terbatas, dan memiliki penyimpanan terbatas yang dilindungi secara fisik (misalnya, tamper). -kartu pintar tahan) dalam setiap penerima. Seperti banyak sistem tertanam lainnya, penerima siaran sering kali tidak diawasi oleh administrator keamanan resmi dan tidak dapat mentolerir skema manajemen kunci yang canggih.

Persyaratan yang saling bertentangan untuk konten dan penerima dinamis serta manajemen kunci yang sederhana menghadirkan tantangan yang signifikan bagi pengembang sistem tertanam. Oleh karena itu, kami menjelaskan dan membandingkan beberapa pendekatan untuk penggunaan kunci enkripsi informasi di lingkungan siaran. Kami fokus pada dua jenis kunci enkripsi utama: *key encryption key* (KEK) dan *transmission encryption key* (TEK). TEK adalah kunci enkripsi data simetris yang digunakan untuk perlindungan konten informasi. Misalnya, informasi dalam transmisi mungkin dilindungi menggunakan algoritma enkripsi massal yang toleran terhadap kesalahan seperti AES-CFB (dijelaskan dalam Bab 4). KEK digunakan untuk mengenkripsi (membungkus) TEK sedemikian rupa sehingga hanya penerima yang memiliki KEK yang dapat mendekripsi dan menggunakan TEK.



Gambar 5.13 Contoh jaringan siaran berbasis sistem tertanam.

TEK Tunggal

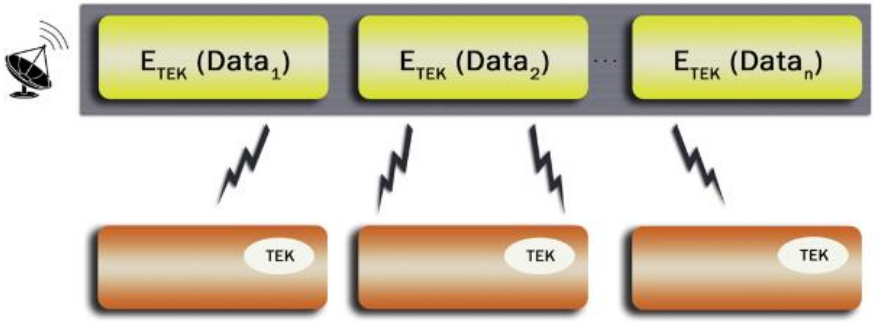
Pendekatan TEK tunggal menggunakan satu kunci konten bersama di antara semua penerima, yang semuanya telah disediakan sebelumnya dengan TEK di pabrik. Penerima memerlukan penyimpanan lokal dalam jumlah kecil, cukup untuk menampung satu TEK. Penyiar mengirimkan informasi terenkripsi yang dilindungi oleh TEK. Melindungi kunci di dalam penerima adalah hal yang sangat penting karena penyerang yang berhasil memulihkan kunci dari memori penerima mana pun atau mengkloning seluruh penerima kemudian dapat memperoleh akses yang tidak sah. Untuk informasi bernilai tinggi, elemen kriptografi berkualitas tinggi yang mampu menahan serangan fisik canggih akan digunakan untuk penyimpanan kunci.

Karena perangkat tersebut mahal, seringkali ukuran memorinya sangat terbatas, sehingga pendekatan TEK tunggal menjadi menggoda. Menambah godaan ini, TEK tunggal juga efisien dalam hal overhead komunikasi karena tidak ada materi kunci tambahan yang disiarkan ke penerima. Tentu saja, keamanan TEK tunggal sangat lemah karena satu penerima yang disusupi akan memungkinkan semua konten yang pernah dikirim dengan TEK disusupi.

Pendekatan TEK tunggal melanggar prinsip-prinsip penting manajemen kunci kriptografi yang dijelaskan dalam Bab 4 dan harus dihindari. Pendekatan TEK tunggal ditunjukkan pada Gambar 5.14.

TEK Kelompok

Kelompok TEK yang terbatas dapat digunakan untuk pengumpulan konten guna mengurangi kompromi informasi ketika TEK tertentu diekspos sambil menghindari kebutuhan untuk memperbarui TEK di dalam penerima. Misalnya, sistem satelit digital untuk TV berbayar dapat menggunakan TEK terpisah untuk program tertentu dan/atau waktu tertentu dalam sehari dan/atau kelompok penerima. Konten yang disiarkan mencakup metadata yang memungkinkan penerima mengidentifikasi TEK yang diperlukan untuk mendekripsi aliran tertentu.



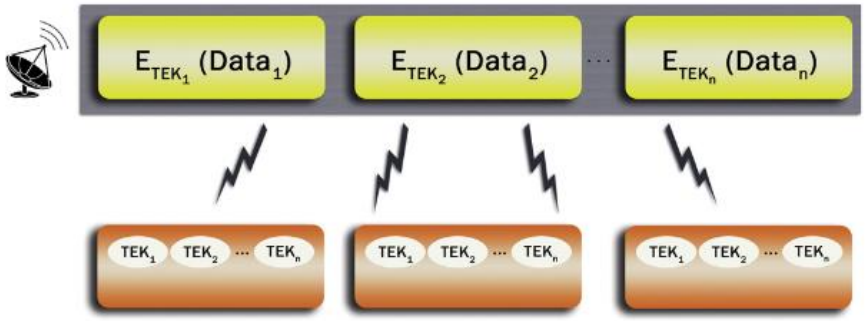
Gambar 5.14 Tunggal, TEK bersama.

Skenario ini menyiratkan bahwa penerima harus menyimpan banyak TEK secara lokal yang mungkin masih tidak dapat digunakan karena kendala perangkat keras. Bahkan jika kita mengasumsikan sejumlah besar TEK yang dapat digunakan, pendekatan kelompok-TEK masih mempunyai kelemahan periode kripto: TEK yang dikompromikan akan mengekspos semua informasi yang pernah dikirimkan menggunakan TEK. Pendekatan multiple TEK, yang ditunjukkan pada Gambar 5.15, adalah efisiensi bandwidth komunikasi.

KEK Tunggal

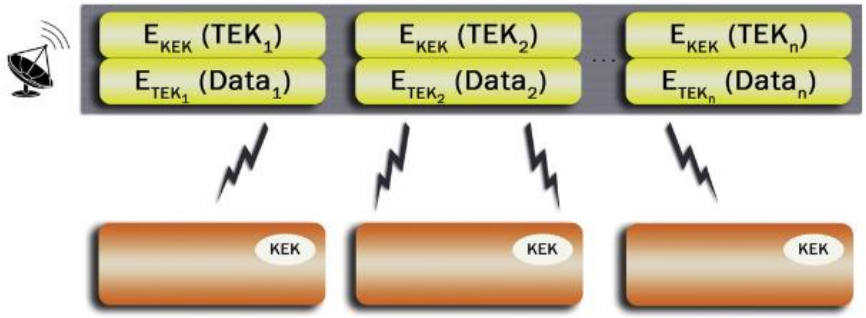
Pendekatan KEK tunggal menggunakan satu KEK yang dibagikan kepada semua penerima untuk melindungi TEK. Hal ini juga mengatasi masalah cryptoperiod karena siaran dapat menggunakan TEK unik untuk setiap transmisi. TEK yang disusupi hanya akan mengekspos konten terbatas yang dilindungi oleh TEK tersebut. Penerima memerlukan penyimpanan lokal dalam jumlah kecil, cukup untuk menampung satu KEK. Penyiar mengirimkan aliran informasi yang mencakup TEK yang dibungkus dan informasi yang terkait dengan TEK tersebut. Persyaratan penyiaran TEK berdampak pada bandwidth komunikasi. Namun, trade-off ini biasanya dapat diabaikan, hanya beberapa byte data saja.

Mengingat sifat-sifat yang menguntungkan ini, pendekatan KEK tunggal tampaknya merupakan kompromi yang masuk akal. Faktanya, sistem televisi satelit DirecTV telah menggunakan skema seperti itu. Lihat Gambar 5.16.



Gambar 5.15 Beberapa TEK.

Pembagian satu kunci simetris untuk semua konten siaran yang dilindungi melanggar penerapan paling dasar dari prinsip hak istimewa paling rendah dan akan dianggap kutukan bagi kriptografer mana pun.



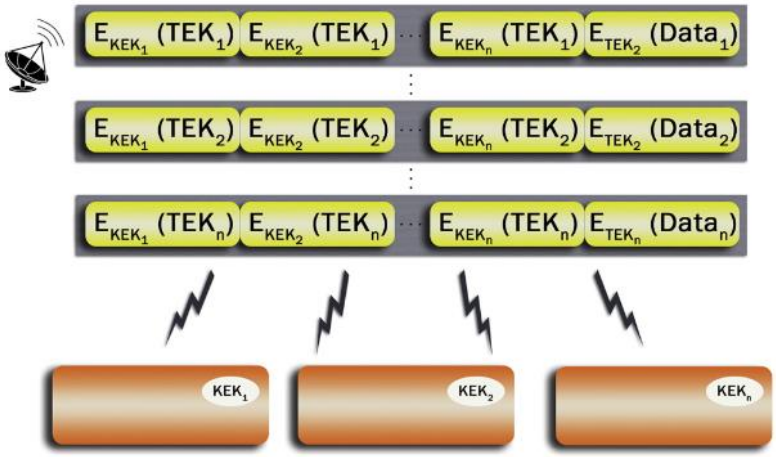
Gambar 5.16 KEK tunggal dengan TEK siaran.

Namun demikian, keberadaan pendekatan ini dalam praktiknya menunjukkan bagaimana pengembang sistem tertanam terkadang mengorbankan keamanan demi kemudahan pengembangan dan pemeliharaan, bahkan untuk informasi bernilai tinggi.

Peretas berhasil membobol receiver DirecTV untuk mencuri KEK dan kemudian kontennya. Kompromi tersebut diyakini tidak terdeteksi selama bertahun-tahun dan menyebabkan kerugian nilai yang sangat besar. Ketika akhirnya terdeteksi, peretasan tersebut menyebabkan perang antara DirecTV dan para pencuri, dengan DirecTV mencoba berbagai perbaikan, diikuti dengan upaya yang semakin canggih untuk menerobos. Akhirnya, DirecTV memasang patch yang secara akurat akan mendeteksi receiver ilegal dan kemudian menonaktifkan receiver tersebut secara permanen dari jarak jauh. Namun, kita harus bertanya-tanya seberapa besar penderitaan yang bisa dicegah jika DirecTV mengadopsi pendekatan yang lebih aman sejak awal.

KEK Per Penerima

Jelasnya, kami ingin memiliki KEK unik per receiver, yang digunakan untuk membungkus TEK per transmisi (lihat Gambar 5.17). Kami akan memiliki efisiensi penyimpanan receiver lokal yang sangat baik dan menghindari masalah yang terkait dengan kompromi kunci bersama. KEK penerima yang dikompromikan dapat dicabut begitu saja tanpa mempengaruhi penerima lainnya.



Gambar 5.17 KEK per penerima dengan TEK terenkripsi per-KEK yang disiarkan.

Sayangnya, pendekatan ini mematikan efisiensi bandwidth komunikasi. Siaran harus menyertakan N salinan TEK yang dibungkus, satu untuk masing-masing N penerima di jaringan. Untuk jaringan dengan satu juta penerima dan TEK 128-bit, 16 MB materi utama akan disiarkan untuk setiap transmisi. Ini setara dengan sekitar 13 detik pada koneksi media 10-Mbit, jelas tidak dapat diterima untuk banyak aplikasi.

Kompromi yang jelas adalah dengan mengelompokkan penerima sedemikian rupa sehingga KEK dapat dibagikan, namun dengan cara yang terbatas (misalnya, berdasarkan geografi atau cara lain) untuk mengurangi overhead bandwidth tanpa mengorbankan sepenuhnya hak istimewa yang paling sedikit.

Banyak penelitian akademis telah diterapkan pada masalah penciptaan kelompok penerima dan kunci hierarki yang efisien untuk mencapai keseimbangan terbaik antara bandwidth komunikasi dan keamanan. Pembaca yang giat dan tidak keberatan dengan prosa akademis yang padat mungkin tertarik dengan makalah ACM "Manajemen Kunci untuk Siaran Terenkripsi."[55]

KEK Tunggal dengan TEK Kelompok

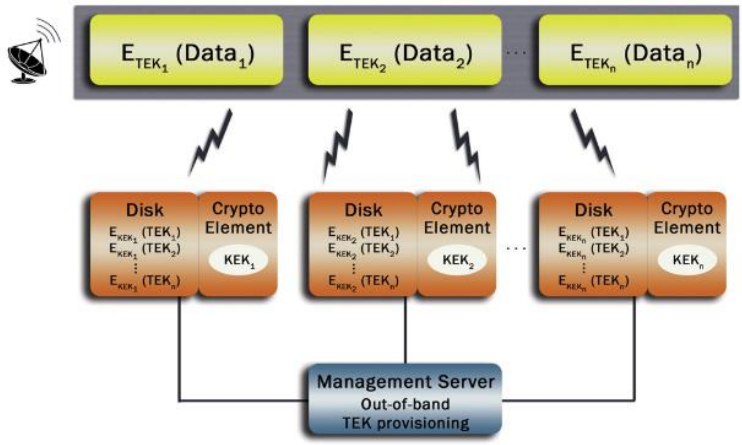
Dalam metode sebelumnya, kami mengasumsikan penyimpanan kunci berbasis elemen kriptografi yang dilindungi secara fisik dalam jumlah terbatas. Kendala ini sepertinya tidak akan berubah dalam waktu dekat. Namun, hal yang juga tidak mungkin terjadi dalam sistem tertanam modern adalah batasan ketat yang serupa dalam bentuk memori non-volatil lainnya.

Teknik ampuh untuk meningkatkan jumlah penyimpanan kunci yang dilindungi secara signifikan adalah dengan menggunakan kunci teks biasa yang dilindungi secara fisik untuk menggabungkan kunci lain yang disimpan dalam media penyimpanan yang banyak, seperti flash NAND atau hard disk.

Oleh karena itu, kami mengusulkan suatu pendekatan (lihat Gambar 5.18) yang hemat bandwidth, mudah dikelola, dan memanfaatkan ruang penyimpanan lokal yang signifikan. Setiap penerima memiliki KEK yang unik; Namun KEK tidak digunakan untuk menyiarkan TEK. Sebaliknya, sejumlah besar TEK dibungkus oleh KEK dan disediakan ke penerima di luar pita

[55] Wool A. Key Management for Encrypted Broadcast. ACM Transactions on Information and System Security (TISSEC) May 2000;3(2).

oleh penyedia layanan. TEK terenkripsi disimpan dalam memori non-volatil biasa. TEK yang sama disediakan untuk penyiar dan digunakan untuk mengenkripsi grup aliran konten tertentu, seperti yang dijelaskan di bagian "*TEK Grup*" sebelumnya. Kurangnya TEK yang disiarkan memberikan efisiensi bandwidth maksimum.



Gambar 5.18 KEK individual dengan TEK grup disediakan di luar pita dan dibungkus dalam disk.

Masa pakai TEK yang disimpan secara lokal bergantung pada jumlah penyimpanan, jumlah aliran data dengan kunci individual yang harus disiarkan secara bersamaan, dan nilai informasi yang dilindungi. Penyedia layanan harus menyediakan kembali penerima pada tingkat yang sepadan dengan habisnya periode kripto TEK.

Meskipun tingkat penyediaan out-of-band mungkin berbeda-beda di berbagai jenis sistem tertanam, beberapa bentuk akses untuk tujuan ini sering kali dimungkinkan. Pada Bab 4, kita membahas skenario kapal selam yang dikerahkan dalam waktu lama dan menerima perintah misi melalui siaran. Kapal selam harus menyimpan cukup TEK secara lokal hingga akhirnya kembali ke pelabuhan. Sebuah mobil harus dibawa masuk secara berkala untuk diservis. Meteran pintar kadang-kadang dapat diservis oleh teknisi utilitas atau mungkin dihubungkan ke jaringan rumah untuk komunikasi out-of-band. Jika penyediaan out-of-band tidak memungkinkan, maka KEK yang dikelompokkan dan TEK yang disiarkan akan diperlukan.

5.3 PROTOKOL DATA TIDAK AKTIF

Pada tahun 2010, jaringan CBS di Amerika menayangkan program yang menunjukkan bagaimana mesin fotokopi kantor yang dibuang adalah tambang emas bagi informasi pribadi, yang diambil dari disk drive di dalam mesin. Dari mesin fotokopi yang dipilih secara acak dari gudang mesin fotokopi bekas, penyelidik menemukan daftar buronan pelaku kejahatan seksual, target penggerebekan narkoba, rencana desain arsitektur, informasi identifikasi pribadi (nama, alamat, nomor Jaminan Sosial), dan catatan medis termasuk hasil tes darah dan diagnosis kanker.[56]

[56] Keteyian A. Digital Photocopiers Loaded With Secrets. http://www.cbsnews.com/stories/2010/04/19/eveningnews/main6412439.shtml?tag=currentVideoInfo;videoMetaInfo; April 20, 2010.

Ketika ditanya apakah situasi ini dapat dicegah, sebuah perusahaan mesin fotokopi mengatakan bahwa pelanggan dapat membeli opsi seharga Rp.750.000 yang akan menghapus gambar yang disalin dari hard drive setelah digunakan. Berikan bonus kepada orang yang menulis beberapa baris kode itu! Solusi lain yang jelas adalah topik bagian ini: enkripsi data tidak aktif. Banyak sistem tertanam dapat memperoleh manfaat dari perlindungan data tidak aktif.

Pada dasarnya, perangkat apa pun yang menyimpan informasi sensitif secara lokal berisiko diserang sehingga seseorang yang memperoleh akses fisik ke sistem tertanam dapat mengakses penyimpanan internal dan membaca informasi sensitif tersebut. Sistem tertanam mungkin menggunakan hard disk drive, memori flash, atau USB thumb drive yang terpasang untuk tujuan ini. Di pasar peralatan kantor, mari kita pertimbangkan jenis sistem tertanam berikut yang dapat dan memang menyimpan informasi sensitif selama pengoperasian normalnya:

- Mesin faks
- Pemindai
- Sistem perekaman pesan telepon
- Tandatangan digital
- Sistem konferensi video
- Sistem kamera keamanan
- Ponsel Pintar dan Gawai
- Pencetak

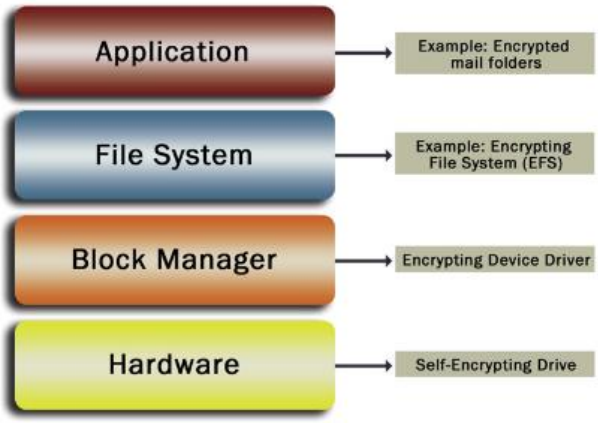
Banyak pengembang sistem tertanam tidak menyadari risiko yang terkait dengan data yang disimpan. Dan pelanggan produk ini memiliki pemikiran yang sama. Pengembang sistem tertanam dapat memimpin dengan terlebih dahulu menyediakan fitur enkripsi data tidak aktif untuk meningkatkan keamanan keseluruhan produk ini dan kemudian memastikan bahwa pengguna akhir memanfaatkan fasilitas ini.

Peraturan kepatuhan di industri tertentu mengharuskan data sensitif yang disimpan dilindungi dengan protokol perlindungan data yang sesuai yang mencakup enkripsi. Di sektor medis, Undang-Undang Portabilitas dan Akuntansi Asuransi Kesehatan (HIPAA) mewajibkan perlindungan data pasien yang disimpan dalam perangkat medis. Di sektor keuangan, Pembayaran

Standar keamanan data Industri Kartu (PCI DSS) memerlukan perlindungan informasi kartu kredit dalam sistem pemrosesan keuangan. Perlindungan data yang tidak disimpan dalam ponsel cerdas dan tablet merupakan persyaratan bagi perusahaan dan pemerintah yang sadar akan keamanan agar perangkat genggam tersebut dapat digunakan untuk memproses informasi sensitif.

5.3.1 Memilih Lapisan Penyimpanan untuk Keamanan

Sebelumnya dalam bab ini kita telah membahas beberapa lapisan dalam tumpukan komunikasi yang dapat dipilih oleh pengembang untuk menerapkan protokol perlindungan data-in-transit. Seperti yang ditunjukkan pada Gambar 5.19, ada pilihan serupa untuk perlindungan data saat tidak digunakan.



Gambar 5.19 Pilihan perlindungan data tidak aktif berdasarkan lapisan.

Dengan *full disk encryption* (FDE), seluruh media yang digunakan untuk penyimpanan dienkripsi. Keuntungannya adalah memastikan bahwa file tersembunyi, seperti file sementara sistem operasi dan ruang swap, tidak terekspos. Ketika FDE ditangani di dalam perangkat media itu sendiri, ini disebut sebagai *self-encrypting drive* (SED). Drive yang mengenkripsi sendiri adalah hal biasa di pasar laptop. Keuntungan SED bagi pengembang sistem tertanam adalah sedikit atau tidak ada perangkat lunak baru yang harus ditulis untuk memanfaatkan fasilitas perlindungan data. Enkripsi dilakukan dengan perangkat keras khusus di dalam perangkat penyimpanan, membongkar prosesor aplikasi utama yang tertanam.

Jika media penyimpanan yang mengenkripsi sendiri memungkinkan, maka ini adalah pilihan yang sangat baik karena kemudahan penggunaan, kinerja yang sangat baik, dan kemampuan untuk menyembunyikan kunci enkripsi penyimpanan dari prosesor dan memori aplikasi utama.

Sayangnya, banyak sistem tertanam tidak dapat menggunakan produk SDE mandiri yang tersedia karena keterbatasan faktor bentuk.

Enkripsi dapat dilakukan pada tingkat berikutnya, lapisan manajemen perangkat, biasanya driver berorientasi blok. Perlindungan pada tingkat ini akan mencakup seluruh perangkat media terkelola (FDE). Implikasi kinerja dari pendekatan ini bervariasi. Jika platform tertanam berisi akselerator enkripsi simetris, maka biaya overhead mungkin masuk akal. Implementasi kriptografi perangkat lunak murni dapat menyebabkan penurunan kinerja yang drastis. Apakah overhead ini dapat diterima tergantung pada tingkat permintaan akses media penyimpanan dan persyaratan kinerja tingkat sistem lainnya. Pengembang sistem tertanam dapat merancang fasilitas enkripsi sedemikian rupa sehingga driver perangkat memanggil rutinitas enkripsi blok media umum, memastikan bahwa perangkat lunak lebih mudah dipelihara di berbagai generasi produk tertanam yang mungkin menggunakan jenis penyimpanan berbeda.

Beberapa sistem operasi tertanam menyediakan paket dukungan papan (BSP) yang menyertakan driver perangkat penyimpanan yang mendukung enkripsi. Kandidat tingkat berikutnya untuk perlindungan data tidak aktif adalah sistem file.

Keuntungan utama penerapan perlindungan penyimpanan pada lapisan sistem file adalah memberikan perincian yang lebih baik dalam pemilihan informasi yang memerlukan kerahasiaan penyimpanan. Memiliki perincian ini sangat penting terutama jika enkripsi

dilakukan dalam perangkat lunak dengan akselerasi perangkat keras yang minimal atau tanpa akselerasi perangkat keras sama sekali. Tergantung pada implementasi sistem file, pengembang mungkin diberikan opsi untuk enkripsi pada tingkat volume atau pada tingkat file individual.

Terakhir, aplikasi dapat menambahkan perlindungan datanya sendiri, baik menggunakan fitur enkripsi sistem file yang mendasarinya atau implementasi khusus. Misalnya, aplikasi pencatatan audit dapat mengenkripsi catatan auditnya sebelum memanggil fungsi keluaran sistem file standar. Dalam kasus perlindungan data tingkat volume, file, atau aplikasi, pengembang dapat menggunakan kunci terpisah untuk kelompok data ini, bukan hanya menggunakan satu kunci untuk keseluruhan sistem. Ini adalah penerapan yang masuk akal dari prinsip-prinsip hak istimewa yang paling rendah.

Terkait dengan pemilihan lapisan keamanan jaringan, tidak ada satu jawaban yang tepat dalam memilih lapisan untuk perlindungan data tidak aktif; dalam beberapa kasus, pengembang mungkin ingin menggunakan lebih dari satu lapisan untuk pertahanan mendalam. Pengembang yang menggunakan pendekatan khusus pada tingkat aplikasi juga perlu mengambil tanggung jawab manajemen kunci, sedangkan pengguna sistem file enkripsi atau SED dapat memanfaatkan kerangka manajemen kunci yang disediakan oleh pemasok produk.

5.3.1 Memilih Lapisan Penyimpanan untuk Keamanan

Data yang tidak aktif menghadirkan beberapa tantangan unik untuk algoritma enkripsi dibandingkan dengan protokol keamanan jaringan. Meskipun pengembang mungkin dapat mengandalkan vendor sistem operasi atau sistem file untuk menangani detail enkripsi, penting untuk memahami apakah pemasok telah memilih pendekatan teknis yang kuat. Selain itu, pembaca yang menerapkan sistem perlindungan data mandiri perlu memahami rincian ini.

Bagi pembaca yang mungkin telah melewatkan Bab 4, sekarang adalah saat yang tepat untuk kembali dan memahami bagian bab yang membahas algoritma dan mode enkripsi simetris. Untuk tujuan diskusi ini, kami mengabaikan perlindungan integritas, karena sebagian besar aplikasi data tidak aktif hanya mementingkan kerahasiaan. Untuk perlindungan data saat tidak digunakan, umumnya kita menginginkan algoritme enkripsi yang dapat dilakukan tanpa menambahkan ruang penyimpanan tambahan: blok media teks biasa dienkripsi di tempatnya, menghasilkan blok teks tersandi dengan ukuran yang sama. Mode enkripsi paling dasar, ECB, akan menyediakan konservasi memori ini tetapi tidak cocok untuk enkripsi data-at-rest karena dua blok teks biasa yang sama akan mengenkripsi ke teks sandi yang sama, sehingga memudahkan penyerang menemukan pola dalam data dan berpotensi memperoleh informasi. Kita harus mempertimbangkan mode lain, yang sebagian besar memerlukan vektor inisialisasi (IV). Namun, untuk menghindari perluasan ruang, sistem perlindungan data harus menyertakan sarana untuk memperoleh IV ini secara implisit.

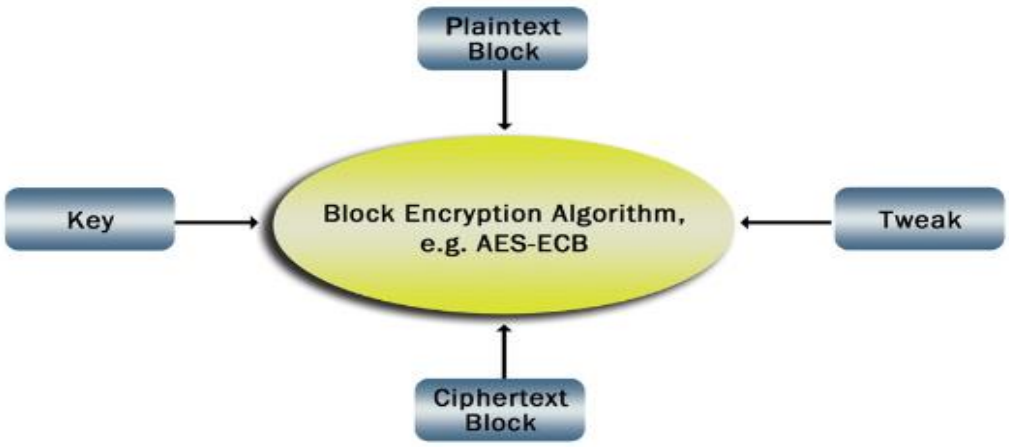
Derivasi IV implisit menimbulkan tantangan yang sangat sulit untuk mode enkripsi umum. Dari studi kita mengenai mode-mode ini di Bab 4, kita tahu bahwa mode umum memerlukan keunikan: IV yang sama tidak boleh digunakan kembali untuk kunci tertentu. Misalnya, dengan mode CTR, penghitung yang dapat diprediksi dapat digunakan, namun angka yang sama tidak akan pernah terulang untuk kunci tertentu. Untuk mode CBC, harus

menggunakan nomor unik dan tidak dapat diprediksi. Protokol keamanan jaringan memiliki kebebasan untuk menghasilkan IV dan mengirimkannya sebagai bagian dari data yang dikirimkan; untuk AES-CBC, setiap transmisi dapat menghasilkan nomor acak baru untuk IV dan mengirimkan IV ini ke penerima. Namun untuk data yang tidak digunakan, kami tidak memiliki ruang untuk menyimpan IV untuk dekripsi selanjutnya.

Sumber yang jelas untuk IV implisit adalah nomor sektor dan offset untuk blok data tertentu. Menggunakan kombinasi ini memberikan setiap blok disk nilai input yang relatif unik. Namun, karena blok data ditimpa selama operasi normal, sektor dan offset yang sama digunakan kembali untuk kunci yang sama. Hal ini menyiratkan kelemahan serius dalam penerapan mode enkripsi umum untuk perlindungan data saat tidak digunakan. Banyak kelemahan lain dari mode umum, khususnya CBC, telah diidentifikasi ketika diterapkan pada protokol perlindungan data-at-rest. Makalah bagus yang membahas kelemahan ini ditulis oleh Clemens Fruhwirth, "Metode Baru dalam Enkripsi Hard Disk."[57]

Sandi yang Dapat Diubah

Kabar baiknya adalah para kriptografer telah bekerja keras untuk mengatasi tantangan mode enkripsi ini. Liskov, Rivest, dan Wagner memperkenalkan konsep cipher blok yang dapat diubah pada tahun 2002.[58] Ide dasar dari cipher yang dapat diubah adalah untuk menerapkan konsep IV pada cipher blok tunggal itu sendiri, bukan pada mode rangkaian yang dibangun di atas cipher blok. Seperti yang ditunjukkan pada Gambar 5.20, cipher blok mengubah blok teks biasa menjadi blok teks tersandi, menggunakan kunci tradisional dan tweak sebagai masukan.



Gambar 5.20 Ikhtisar sandi blok yang dapat diubah.

Penerapan praktis sandi yang dapat diubah untuk masalah perlindungan data tidak aktif adalah properti yang keamanan sandinya tidak menghalangi penggunaan kembali vektor inisialisasi; dengan demikian, nomor sektor media dan blok offset dalam sektor tersebut sangat cocok untuk pemilihan penyesuaian.

[57] Fruhwirth C. New Methods in Hard Disk Encryption; Institute for Computer Languages Theory and Logic Group. Vienna University of Technology; July 18, 2005.

[58] Liskov M, Rivest R, Wagner D. Tweakable Block Ciphers. LNCS, Crypto: Santa Barbara, CA; 2002.

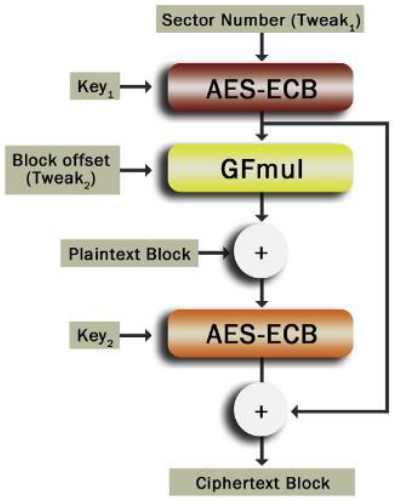
XTS-AES

Pada tahun 2007, Kelompok Kerja Keamanan dalam Penyimpanan (SISWG) IEEE menerbitkan standar P1619.[59]

Standar IEEE P1619 mendefinisikan mode sandi XTS-AES sebagai hasil studi menyeluruh terhadap berbagai algoritma berbasis tweak potensial untuk digunakan dalam aplikasi perlindungan data tidak aktif. Pilihan ini selanjutnya didukung oleh NIST dalam Publikasi Khusus 800-38E,[60] yang menyetujui mode sandi XTS-AES dan merujuk definisinya di IEEE P1619-2007. Terakhir, FIPS 140-2 telah diubah untuk menyertakan XTS-AES sebagai sandi yang disetujui untuk validasi.

Sebagai contoh adopsi komersial, perhatikan bahwa rilis terbaru Mac OSX pada saat penulisan ini, dengan nama kode Lion, adalah sistem operasi Mac pertama yang menyediakan fitur enkripsi disk lengkap asli; produk ini menggunakan XTS-AES. Algoritme penyesuaian yang ditemukan di XTS-AES didasarkan pada dan hampir identik dengan yang awalnya dibuat oleh kriptografer terkenal Phillip Rogaway, yang disebut XEX.[61] Selain keamanan yang kuat, XEX dan oleh karena itu XTS-AES juga dirancang untuk efisiensi ketika diterapkan pada penyimpanan dari banyak blok data berurutan (seperti yang umum terjadi pada penyimpanan file). Cipher blok XTS-AES digambarkan pada Gambar 5.21.

Aspek yang aneh dari sandi ini adalah persyaratan bahan kunci dua kali lipat; untuk keamanan 128-bit, kunci 256-bit harus digunakan. Bagian pertama dari kunci digunakan untuk memproses teks biasa; paruh kedua digunakan untuk mengenkripsi representasi 128-bit dari nomor sektor, yang bertindak sebagai penyesuaian utama, seperti yang ditunjukkan pada Gambar 5.21. Hasil enkripsi ini kemudian diumpankan ke fungsi yang melakukan perkalian bidang Galois (diimplementasikan sebagai urutan pergeseran dan XOR) dari hasil enkripsi dengan konstanta Galois yang berasal dari penyesuaian sekunder, indeks numerik dari blok data dalam sektor. Lihat dokumen IEEE P1619 untuk definisi rinci tentang operasi perkalian Galois.



Gambar 5.21 Sandi enkripsi data tidak aktif XTS-AES.

[59] Security in Storage Working Group of the IEEE Computer Society Committee. IEEE P1619. Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices; 2007.

[60] NIST Special Publication 800-38E. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices; January 2010.

[61] Rogaway P. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC, http://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf; September 24, 2004

Hasil perkalian Galois ini digunakan dua kali. Pertama, ditambahkan (XOR) ke blok teks biasa, yang kemudian dienkripsi dengan separuh kunci pertama. Kedua, hasil Galois ditambahkan (XOR) lagi ke hasil enkripsi blok plaintext untuk membuat blok ciphertext akhir. Dekripsi serupa; namun, meskipun algoritma dekripsi AES-ECB digunakan untuk memproses ciphertext, tweak ciphernya tetap sama, menggunakan algoritma enkripsi AES-ECB.

Dalam praktiknya, data disimpan ke media di sektor-sektor. Oleh karena itu, algoritma enkripsi blok sebelumnya harus dijalankan dalam satu lingkaran di seluruh sektor. Perhatikan bahwa meskipun XTS-AES menangani blok parsial, bagian algoritma tersebut sering kali tidak diperlukan. Misalnya, ukuran sektor umum sebesar 512 byte akan menghasilkan 32 blok enkripsi, dan sebagian besar lapisan manajemen media akan mengakses seluruh sektor dalam satu waktu. Untuk sistem seperti itu, dengan fungsi xts_encrypt, yang mengambil nomor dan ukuran sektor dalam byte, blok teks biasa, dan kunci enkripsi sebagai masukan, urutan kode sederhana berikut menangani enkripsi sektor:

```
Sector_encrypt (uint8_t *sector. Uint32_t sector_num. Uint32_t
      Sector_size. Uint8_t key [ ])
{
  Uint32_t i ;
  Assert((sector_size % AES_BLOCK_SIZE) – 0) ; /*512 % 16*/
  For (i=0; i < sector_size/AES_BLOCK_SIZE ; i++) /*32x */
      Xts_encrypt (sector+i*AES_BLOCK_SIZE, sector_num, i) ;
}
```

Juga mudah untuk melihat dari urutan kode ini bahwa XTS-AES dapat diparalelkan. Jika sistem tertanam berisi akselerator perangkat keras AES (terutama yang memiliki dukungan langsung untuk mode XTS), implementasi sebelumnya harus dimodifikasi untuk memanfaatkan kemampuan akselerator dalam memproses beberapa blok AES sekaligus. Selain itu, jika media mengizinkan konfigurasi ukuran sektor, pengembang mungkin ingin memvariasikan ukuran sektor untuk melihat apakah hasil yang lebih baik (berpotensi mengorbankan efisiensi ruang yang sedikit berkurang) dapat dicapai.

Pengembang sistem tertanam yang memilih produk perlindungan data tidak aktif disarankan untuk menghindari pendekatan lama yang menggunakan mode yang lebih lemah (banyak penerapan berbasis CBC telah dikomersialkan) dan sebagai gantinya menggunakan standar yang disetujui NIST dan FIPS. Mungkin mengejutkan, panduan Suite B NSA saat ini pada saat penulisan ini masih bersifat segan sehubungan dengan protokol dan mode data tidak aktif seperti XTS-AES. Namun NSA telah menyatakan keinginan yang kuat agar produk komersial menggunakan kriptografi yang sesuai dengan Suite B dan menunjuk pada validasi FIPS 140-2 sebagai persyaratan tingkat awal untuk solusi komersial. Jadi, Suite B membuat kasus tidak langsung untuk XTS-AES.

5.3.3 Mengelola Kunci Enkripsi Penyimpanan

Tujuan utama perlindungan data-at-rest adalah untuk memastikan bahwa informasi yang berada pada media yang hilang atau dicuri tidak dapat diakses oleh pihak yang tidak berwenang yang harus diasumsikan memiliki akses fisik penuh terhadap media tersebut.

Kunci enkripsi penyimpanan simetris tidak boleh disimpan secara jelas di media. Namun, seringkali salinan terenkripsi dari kunci simetris perlu disimpan pada media (atau mungkin Modul Platform Tepercaya yang terlampir, jika tersedia). Kuncinya dibuka untuk penggunaan aktif saat sistem dijalankan dengan cara yang sah. Untuk komputer pribadi seperti laptop dan ponsel pintar, pembukaan bungkus dipicu oleh keberhasilan otentikasi pengguna (misalnya, menggunakan kata sandi, kartu pintar, biometrik, dll., atau beberapa faktor).

Membuat Kunci Enkripsi Penyimpanan

Metode umum pembuatan kunci enkripsi penyimpanan, misalnya, adalah mengubah kredensial pengguna menjadi kunci menggunakan fungsi derivasi kunci (KDF). KDF populer yang digunakan untuk mengonversi kata sandi adalah fungsi derivasi kunci berbasis kata sandi, versi 2 (PBKDF2).

PBKDF2 didefinisikan dalam spesifikasi Laboratorium RSA PKCS #5;[62] diduplikasi dalam RFC 2898.[63] PBKDF2 menerapkan fungsi hash pada kata sandi yang digabungkan dengan garam (bitstring acak). Untuk mempersulit peretasan kata sandi, standar merekomendasikan agar keluaran hash diulang beberapa kali. Jumlah iterasi hash minimum yang disarankan adalah 1.000, meskipun jumlahnya diperkirakan akan meningkat seiring waktu. iOS 4.0 Apple menggunakan 10.000 iterasi. Pada bulan September 2010, layanan cadangan terenkripsi RIM BlackBerry dinyatakan rentan karena kesalahan penerapan PBKDF2: alih-alih mengikuti standar, perangkat lunak BlackBerry menggunakan jumlah iterasi satu. Kerentanan BlackBerry didokumentasikan dalam Database Kerentanan Nasional sebagai CVE-2010-3741.[64]

Ketika kata sandi digunakan untuk secara langsung menghasilkan kunci enkripsi penyimpanan, perubahan kata sandi akan mengubah kunci enkripsi dan karenanya memaksa enkripsi ulang seluruh media yang dilindungi. Untuk menghindari masalah ini, kunci enkripsi unik dan permanen dibuat saat media pertama kali disediakan, lalu kunci ini dibungkus (dienkripsi) dengan kunci turunan kata sandi. Dengan skema penguncian dua tingkat ini, perubahan kata sandi secara berkala (praktik keamanan yang baik) hanya memerlukan pembungkusan ulang kunci enkripsi.

Pendekatan otentikasi pengguna mungkin cukup untuk jenis sistem tertanam yang dihadiri terbatas yang dapat mentolerir intervensi pengguna setiap kali volume yang dilindungi harus dibuka kuncinya. Bayangkan sebuah printer perusahaan yang memiliki FDE (sayangnya, sebagian besar tidak memilikinya, namun itulah alasan kami ada di sini!). Dengan sistem ketersediaan tinggi di lingkungan tempat kerja, mungkin dapat diterima jika meminta administrator sistem untuk masuk ke printer guna membuka kunci volume yang dilindungi jika terjadi boot ulang sistem (direncanakan atau tidak direncanakan).

Namun demikian, pendekatan ini tidak cukup untuk kelas besar sistem tertanam tanpa pengawasan. Jika sistem tertanam mengalami kesalahan dan melakukan boot ulang secara otomatis, volume terenkripsi harus dapat kembali online tanpa input kredensial manual.

---

[62] RSA Laboratories. PKCS #5 v2.0: Password-Based Cryptography Standard; March 25, 1999.

[63] PKCS #5: Password-Based Cryptography Specification Version 2.0. Internet Engineering Task Force, Request for Comments: 2898; September 2000.

[64] National Institute of Standards and Technology National Vulnerability Database. http://web.nvd.nist.gov/view/vuln/detail?vulnId¼CVE-2010-3741.

Penyediaan Kunci Jarak Jauh

Kita dapat mempertimbangkan dua kelas sistem tertanam tanpa pengawasan: sistem yang memiliki antarmuka jaringan manajemen jarak jauh dan sistem yang tidak. Untuk yang terakhir, sistem tertanam tidak memiliki mekanisme interaksi dinamis yang dapat membuka kunci enkripsi. Dalam hal ini, jika nilai informasi memerlukan perlindungan data saat tidak digunakan, maka perancang disarankan untuk menggabungkan koprosesor kriptografi yang menyediakan penyimpanan kunci tahan kerusakan fisik dan eksekusi internal algoritma enkripsi data. Driver perangkat mengirimkan teks biasa ke enkripsi ini dan menerima teks sandi untuk disimpan di disk dan juga meminta dekripsi blok disk sesuai kebutuhan.

Untuk sistem tertanam yang mendukung jaringan, server manajemen jarak jauh menyimpan database kunci enkripsi data yang disediakan. Koneksi server dimulai oleh sistem tertanam setiap kali kunci enkripsi data harus dibuka (misalnya, pada saat boot). Sistem tertanam dan server saling mengautentikasi (menggunakan beberapa pilihan protokol keamanan jaringan yang dijelaskan sebelumnya dalam bab ini), dan kemudian server menyediakan salinan kunci enkripsi data yang disediakan sistem tertanam melalui saluran aman. Pendekatan ini jelas bergantung pada konektivitas jaringan yang tersedia untuk mengaktifkan layanan perlindungan data. Perancang sistem harus mengadopsi kebijakan untuk menangani kasus di mana jaringan tidak aktif ketika operasi pembukaan kunci diperlukan. Kebijakan ini dapat menonaktifkan operasi penyimpanan data sensitif tertentu, mengizinkan penyimpanan yang tidak dilindungi, atau mungkin memaksa perangkat memasuki kondisi non-operasional.

Penampungan Kunci

Saat kita lupa kata sandi situs web, selalu ada pertanyaan “lupa kata sandi Anda?” link yang bisa kita klik untuk meresetnya.

Saat menerapkan sistem perlindungan data tidak aktif, pengembang harus mempertimbangkan escrow kunci untuk mencegah kemungkinan hilangnya informasi autentikasi yang digunakan untuk membuka kunci enkripsi penyimpanan. Selain itu, ada situasi di mana pemilik sistem mungkin perlu mengekstrak data dari penyimpanan, misalnya setelah terjadi kegagalan sistem. Menyimpan salinan kunci enkripsi data di lokasi aman di luar lokasi disarankan di sebagian besar desain sistem untuk mencegah hilangnya data ketika kunci enkripsi data tidak lagi dapat diakses. Jika sistem tertanam tidak memiliki antarmuka manajemen jaringan, maka kunci yang disimpan secara internal harus dapat diekspor ke media untuk penyimpanan escrow di luar lokasi (misalnya, dalam brankas yang aman). Jika sistem mendukung manajemen jaringan dan penyediaan kunci jarak jauh yang dijelaskan di bagian sebelumnya, maka pengembang hanya perlu memastikan bahwa kunci yang disediakan dari jarak jauh disimpan di server yang aman atau disalin ke media offline yang dilindungi.

Uraian ini mulai menyentuh subjek umum manajemen siklus hidup utama. Banyak pengembang sistem tertanam yang menambahkan fungsi keamanan, seperti perlindungan data saat tidak digunakan, untuk pertama kalinya mungkin tidak siap untuk menangani masalah pengelolaan kunci pada penerapan yang besar. Selain itu, bagi produsen sistem tertanam, pelanggan mungkin berharap bahwa sistem manajemen siklus hidup utama disediakan sebagai bagian dari penawarannya. Misalnya, pengembang smart meter tidak

hanya dapat menjual smart meter, namun juga sistem manajemen kunci back-end untuk membuat, mendistribusikan, mengaudit, dan menyimpan kunci yang digunakan untuk perlindungan data yang tidak aktif dalam smart meter. Meskipun diskusi komprehensif tentang manajemen siklus hidup utama perusahaan berada di luar cakupan buku ini, banyak pemasok perangkat lunak tertanam yang menyediakan solusi keamanan jaringan dan perlindungan data tidak aktif juga menawarkan sistem manajemen siklus hidup utama. Pembaca didorong untuk berkonsultasi dengan vendor untuk melihat opsi apa yang tersedia.

5.3.4 Ancaman Tingkat Lanjut terhadap Solusi Enkripsi Data

Perangkat lunak autentikasi yang dijalankan untuk membuka kunci media terenkripsi harus dapat dipercaya dan dilindungi dari kerusakan. Misalnya, sistem operasi tertanam dapat menggabungkan fungsi otentikasi secara langsung. Citra sistem operasi yang tertanam (dan boot loader sebelumnya) tidak dienkripsi; hanya media lainnya, yang berisi file sensitif, yang dilindungi. Jika sistem operasi yang tertanam tidak dipercaya (misalnya, berisiko mengandung malware atau kerentanan yang memungkinkan pemuatan malware), maka proses autentikasi dapat digagalkan. Misalnya, pencatat kunci dapat mencatat kata sandi pengguna, memungkinkan pemulihan kunci enkripsi penyimpanan dan semua data terenkripsi.

Jika kami berasumsi bahwa sistem operasi yang tertanam dapat dipercaya, maka kami tetap harus memastikan bahwa apa pun yang dijalankan sebelum peluncuran sistem operasi tersebut dapat dipercaya. Ini adalah contoh bagus lainnya mengenai perlunya boot aman (dibahas di Bab 2).

Dalam beberapa kasus, perancang mungkin ingin sistem operasi yang tertanam juga dienkripsi. Ketika FDE sedang digunakan dan sistem operasi canggih (misalnya Linux) berada pada disk terenkripsi, otentikasi pra-boot dapat digunakan: sebagian kecil dari disk terenkripsi berisi sistem operasi mini yang di-boot hanya untuk tujuan melakukan otentikasi dan membuka kunci media sebelum mem-boot sistem operasi penuh. Jika sistem operasi tertanam adalah mikrokernel aman (Bab 2), maka modul otentikasi pra-boot terpisah tidak diperlukan.

Serangan terhadap pengautentikasi pra-boot telah berhasil dilakukan. Misalnya, sistem di-boot ke sistem operasi berbahaya (misalnya, booting alternatif dari drive USB eksternal) yang merusak kode pra-boot untuk mencuri kredensial autentikasi yang dimasukkan.[65]65 Boot aman juga dapat mencegah serangan ini ; tanda tangan pengautentikasi yang dimodifikasi akan gagal mencocokkan versi yang dikenal baik, sehingga membatalkan proses booting.

Contoh lain dari ancaman tingkat lanjut adalah apa yang disebut serangan cold-boot. Kecuali jika sistem tertanam menggunakan hard drive yang dapat mengenkripsi sendiri yang kuncinya disimpan di dalam media dan tidak pernah diekspos ke prosesor utama, enkripsi disk mengharuskan kunci enkripsi penyimpanan disimpan di memori (di tempat yang jelas) saat sistem sedang beroperasi. , menjalankan algoritma enkripsi dan dekripsi untuk mengakses data. Ketika sistem dimatikan, RAM tidak tersedia, dan satu-satunya salinan kunci enkripsi dienkripsi sendiri. Atau itu? Di beberapa sistem, RAM tidak segera dibersihkan. Penyerang

[65] Turpe S, et al. Attacking the BitLocker Boot Process. Proceedings of the 2nd International Conference on Trusted Computing. TRUST; 2009. Oxford, UK, April 6e8, 2009; LNCS 5471, Springer, 2009. DOI: 10.1007/ 978-3-642-00587-9_12.

mem-boot sistem menggunakan sistem operasi berbahaya yang mengambil kunci teks biasa di RAM. Serangan ini telah berhasil dilakukan.[66]

Perlindungan data saat tidak digunakan dalam sistem tertanam yang dilengkapi dengan boot aman dan sistem operasi tepercaya yang tahan terhadap serangan jarak jauh masih dapat dikalahkan dengan menghapus media yang dilindungi dan mem-boot-nya pada komputer lain yang tidak memiliki lingkungan aman ini. Mengikat kunci enkripsi penyimpanan ke platform sistem tertanam yang dimaksudkan dapat mencegah serangan ini. Dalam hal ini, kunci enkripsi penyimpanan permanen diperoleh (secara keseluruhan atau dikombinasikan dengan kredensial pengguna) dari kunci khusus platform, seperti kunci yang dapat diprogram satu kali atau kunci TPM (jika berlaku). Bahkan jika kredensial pengguna dicuri, kunci enkripsi penyimpanan tidak dapat diperoleh di luar platform tertanam yang ditargetkan. Kelemahan dari tingkat pertahanan ekstra ini adalah kegagalan perangkat keras yang mencegah akses ke kredensial platform akan membuat data tidak dapat diakses secara permanen (kecuali kunci enkripsi penyimpanan turunan itu sendiri disimpan dengan aman).

Pengembang sistem tertanam yang ingin menggabungkan keamanan jaringan dan/atau perlindungan data tidak aktif ke dalam desain berikutnya dihadapkan pada banyak pilihan dan kendala desain. Bab ini berupaya memberikan gambaran umum kepada para desainer tentang isu-isu utama yang perlu dipertimbangkan. Pada tingkat arsitektur perangkat lunak, pilihan protokol keamanan jaringan utama mencakup IPsec dan TLS; Meskipun revisi terbaru memecahkan masalah keamanan yang penting, perancang juga harus mempertimbangkan persyaratan untuk interoperabilitas dengan rekan komunikasi yang menggunakan versi yang lebih lama. Dari perspektif perangkat keras dan sistem, perancang perlu mempertimbangkan kemampuan keamanan perangkat keras, termasuk pelepasan protokol penuh serta akselerasi algoritma kriptografi dan penyimpanan kunci yang aman, dari prosesor yang tertanam. Kendala jejak kaki dan latensi serta persyaratan sertifikasi selanjutnya dapat mempengaruhi jalan yang diambil. Pertimbangan khusus untuk perlindungan data tidak aktif mencakup penggunaan algoritma enkripsi simetris yang disetujui pemerintah yang dirancang khusus untuk aplikasi tersebut dan pengelolaan yang tepat atas kunci jangka panjang yang biasanya digunakan untuk tujuan ini.

[66] Halderman AJ, Schoen SD, Heninger N, Clarkson W, Paul W, Candrino JA, Feldman AJ, Appelbaum J, Felten EW. Lest We Remember: Cold Boot Attacks on Encryption Keys. Proceedings of the 17th USENIX Security Symposium; August 2008. 45e60.

BAB 6
APLIKASI YANG MUNCUL

Selamat datang di bab terakhir buku ini dan mungkin bab yang paling menyenangkan bagi penulis untuk ditulis dan semoga sama menyenangkannya untuk Anda baca. Meskipun buku ini mencakup banyak studi kasus, bab ini memberikan sejumlah kecil studi kasus yang lebih rinci mengenai permasalahan, kekhawatiran, dan panduan mengenai pembuatan sistem tertanam yang aman. Secara khusus, kami fokus pada beberapa topik modern yang mungkin akan segera menghadirkan tantangan unik untuk proyek tertanam Anda berikutnya: transaksi jaringan tertanam, keamanan otomotif, dan cara menggabungkan sistem operasi Android dan Linux yang populer ke dalam sistem seluler dan sistem tertanam dengan aman.

6.1 TRANSAKSI JARINGAN TERTANAM

Sebagian besar perdagangan dunia dan pengendalian infrastruktur penting bergantung pada keamanan transaksi jaringan. Oleh karena itu, kemampuan untuk memberikan keamanan transaksi end-to-end, yang mencakup klien dan server, sangat penting bagi perusahaan, pemerintah, dan konsumen. Semakin banyak sistem tertanam yang bertindak sebagai klien (dan terkadang server) dalam transaksi tersebut. Namun postur keamanan saat ini untuk transaksi jaringan tertanam tidak memadai. Infrastruktur penting dipenuhi dengan kerentanan baik pada sistem operasi maupun aplikasi, baik pada titik akhir klien maupun pada server. Kami menerapkan filter, pemindai, dan Patch Tuesday, namun selalu ada kerentanan baru yang membuat sumber daya penting kami terekspos. Dan seiring dengan semakin bergantungnya kita pada transaksi jaringan, tekad dan kecanggihan peretas pun meningkat secara bersamaan.

Hanya dengan Rp.375.000.000, seorang peretas canggih dapat membobol hampir semua jaringan, bahkan pusat data perusahaan yang paling dilindungi sekalipun. Peretas dapat melewati kontrol keamanan server, atau lebih sering, menumbangkan titik akhir klien yang relatif lemah untuk mengakses permata utama server. Semakin banyak titik akhir klien yang merupakan sistem tertanam.

Meskipun ada banyak produk keamanan dan proyek penelitian yang berupaya meningkatkan keamanan transaksi jaringan, belum pernah ada solusi yang sangat terjamin (yaitu, tersertifikasi untuk memberikan pengguna keyakinan yang sangat tinggi bahwa sistem akan melindungi informasi bernilai tinggi dari serangan). penyerang canggih) dan dapat diterapkan (efektif biaya dan mudah digunakan). Bagian ini menjelaskan ancaman keamanan modern utama yang dihadapi transaksi jaringan dan mensurvei beberapa upaya terbaru (sebagian lemah, sebagian menjanjikan) untuk mengatasi masalah tersebut. Secara khusus, kami mengidentifikasi pendekatan keamanan yang dapat diskalakan di semua platform komputasi klien yang melakukan transaksi komersial melalui Internet, termasuk telepon seluler dan perangkat yang tertanam dalam.

6.1.1 Anatomi Transaksi Jaringan

Dalam transaksi jaringan, klien membuat permintaan ke server jauh. Klien mungkin melakukan pembayaran (atas nama pengguna ponsel), mengubah status sumber daya jarak

jauh, atau meminta berbagai macam layanan lainnya. Server memenuhi permintaan dengan mentransfer dana, mengubah status akun, dan sebagainya, dan sering kali mengirimkan kembali respons ke klien, menyelesaikan transaksi.

Keamanan transaksi ditentukan oleh kemampuan klien dan server untuk mempercayai keaslian, integritas, dan kerahasiaan pesan yang dikirim dan diterima oleh kedua belah pihak selama transaksi. Serangan terhadap transaksi dapat menargetkan data transaksi baik di tempat, di dalam browser, atau selama transmisi, atau pada komputer titik akhir dan sumber daya yang terhubung dengannya.

6.1.2 Keadaan Ketidakamanan

Cara paling umum bagi peretas untuk mendapatkan akses ke informasi penting di server adalah dengan membajak titik akhir klien. Identitas klien dapat dicuri melalui berbagai macam teknik, beberapa di antaranya akan kita bahas nanti. Peretas menggunakan kredensial dan/atau sesi web yang disita untuk melakukan transaksi terlarang dan mendapatkan akses tidak sah ke server. Meskipun server biasanya dilindungi oleh peralatan keamanan yang mahal seperti firewall, sistem deteksi intrusi, sistem perlindungan intrusi, dan perangkat manajemen ancaman terpadu, sistem tertanam umumnya memiliki perlindungan yang jauh lebih lemah dan sering kali terletak di luar zona keamanan server, sehingga memberikan profil serangan yang lebih lembut. untuk peretas.

Seperti dilansir firma analis IDC, Sarana transmisi kode berbahaya (malware) telah berkembang selama dekade terakhir dari floppy disk, email, hingga yang terbaru, jaringan itu sendiri. Hal ini sangat mengancam karena pembuat malware telah menemukan bahwa klien yang terhubung ke jaringan nirkabel dan kabel berkecepatan tinggi merupakan sarana ampuh untuk menyebarkan infeksi. Klien telah menjadi sumber serangan jaringan terdistribusi pada server dan yang lebih penting, aplikasi.[1]

Dampak buruk dari serangan terhadap transaksi ini telah dipublikasikan dengan baik. Kami hanya membahas beberapa peristiwa terkini yang menunjukkan semakin canggihnya serangan-serangan ini. Menurut laporan Businessweek, para pejabat tinggi NASA direkayasa secara sosial melalui email palsu, yang mengakibatkan komputer di kantor pusat badan tersebut di Washington dibobol pada tahun 2006. Salah satu temuan penting dari penyelidikan selanjutnya, seperti dilansir Inspektur Jenderal NASA, adalah bahwa "cakupan, kecanggihan, waktu, dan karakteristik permusuhan dari beberapa intrusi menunjukkan bahwa intrusi tersebut terkoordinasi atau dikelola secara terpusat."[2]

Selama kampanye kepresidenan AS tahun 2008, CNN melaporkan intrusi malware komputer titik akhir yang canggih yang bertujuan untuk membahayakan informasi kebijakan kepresidenan. Agen dari FBI dan Dinas Rahasia melaporkan ke markas kampanye Obama bahwa "Anda mempunyai masalah yang jauh lebih besar dari apa yang Anda pahami. Anda telah disusupi, dan sejumlah besar file telah dimuat dari sistem Anda."[3]

InformationWeek melaporkan bahwa pelanggaran TJX tahun 2007 yang dipublikasikan secara luas yang mengakibatkan pencurian 45 juta catatan pelanggan, setidaknya sebagian,

---

[1] What is Endpoint Security? http://www.endpointsecurity.org/Documents/What_is_endpointsecurity.pdf.

[2] Epstein K, Elgin B. Network Security Breaches Plague NASA. http://www.businessweek.com/print/magazine/content/08_48/b4110072404167.htm; November 20, 2008.

[3] Bohn K, Todd B. Obama, McCain Campaigns' Computers Hacked for Policy Data. http://www.cnn.com/2008/TECH/11/06/campaign.computers.hacked; November 6, 2008

berasal dari kios di dalam toko yang digunakan oleh peretas untuk mendapatkan akses ke LAN perusahaan.[4] Juga pada tahun 2008, Wired.com melaporkan bagaimana email ratu remaja Miley Cyrus (yang berada di Google cloud), bersama dengan beberapa foto pribadi, dicuri ketika seorang peretas menipu pekerja MySpace untuk menyerahkan kredensial yang memungkinkan akses administrator ke server MySpace.[5]

Tentu saja, ini hanyalah segelintir dari serangan yang dilaporkan terhadap komputasi berbasis transaksi. Namun, banyaknya cerita semacam ini yang terjadi setiap hari memberikan bukti ketidakamanan saat kita bertransaksi bisnis, mengendalikan infrastruktur, dan bersosialisasi di Internet. Pada bagian berikut, kami mensurvei ancaman utama yang dihadapi endpoint jaringan saat ini.

6.1.3 Ancaman Transaksi Berbasis Jaringan

Man-in-the-Middle dan/atau Penyadapan

Serangan Man-in-the-middle (MITM) sangat diinginkan oleh penyerang karena tidak memerlukan intrusi titik akhir klien yang berhasil seperti malware yang diunduh. Sebaliknya, penyerang mencegat pesan klien dan memasukkan pesan kembali ke klien (dan dalam beberapa kasus, juga kembali ke server yang dituju pengguna) untuk mengarahkan klien ke server palsu atau meniru koneksi yang valid untuk tujuan jahat.

Serangan man-in-the-middle mungkin terjadi tidak hanya pada tautan yang tidak terenkripsi, namun juga pada tautan terenkripsi, seperti koneksi HTTP yang dilindungi SSL, dengan mengeksploitasi kelemahan dalam infrastruktur kriptografi atau sistem operasi yang mendasarinya.

Salah satu contoh ancaman MITM dapat ditemukan pada apa yang disebut titik akses Wi-Fi "kembaran jahat" di mana penyerang menyamar sebagai tautan Internet publik yang tidak terenkripsi untuk mendapatkan akses ke perangkat seluler, bahkan tanpa pengguna sadari. koneksi terlarang.[6] Serangan MITM dapat digunakan untuk mencuri informasi pribadi langsung dari komputer pengguna (yang kemudian dapat digunakan untuk meluncurkan serangan tambahan) serta untuk menguping.

Phishing dan Serangan Rekayasa Sosial Lainnya

Salah satu kritik paling umum terhadap solusi teknologi baru terhadap masalah keamanan adalah bahwa solusi tersebut tidak mampu melawan faktor manusia. Meskipun benar bahwa kesalahan manusia dapat mengakibatkan pengendalian keamanan teknologi menjadi tidak berdaya, seperti yang akan kita bahas nanti, masalah keamanan umum yang disebabkan oleh sosial memang dapat diatasi dengan teknologi.

Serangan phishing adalah upaya yang dilakukan oleh situs web atau email jahat untuk menipu pengguna agar memberikan informasi sensitif. Secara umum, rekayasa sosial mengacu pada manipulasi orang yang menyebabkan mereka melakukan tindakan atau membocorkan informasi.

---

[4] Greenemeier L. The TJX Effect. http://www.informationweek.com/news/security/cybercrime/showArticle. jhtml?articleID¼201400171; August 11, 2007.

[5] Zetter K. Miley Cyrus Hacker Raided by FBI. http://blog.wired.com/27bstroke6/2008/10/miley-cyrus-hac.html; October 20, 2008.

[6] Kirk J. 'Evil Twin' Wi-Fi Access Points Proliferate. http://www.networkworld.com/news/2007/042507- infosec-evil-twin-wi-fi-access.html; April 25, 2007.

Serangan rekayasa sosial terkenal lainnya memancing pengguna untuk memasang media yang terinfeksi, misalnya, flash drive USB yang dijatuhkan ke tanah di tempat parkir perusahaan. USB flash drive yang terinfeksi diyakini sebagai salah satu vektor infiltrasi utama Stuxnet, yang dibahas di Bab 1.

Secara umum, serangan rekayasa sosial semakin lazim dan efektif, hal ini sebagian disebabkan oleh semakin canggihnya peretas yang ingin menambang data pribadi di Internet yang dapat digunakan untuk melancarkan serangan dan meningkatnya penggunaan situs jejaring sosial seperti Facebook. lebih banyak data pribadi ke dalam domain publik dan menjadikan data tersebut lebih mudah diakses.

Serangan Malware

Malware pencuri data pada perangkat titik akhir dapat mencatat penekanan tombol atau mengikis konten layar untuk mendapatkan informasi transaksi yang dapat digunakan untuk merusak transaksi di masa depan atau menyediakan sarana untuk menyerang server. Malware dapat menonaktifkan perangkat lunak anti-malware, meluncurkan bentuk serangan lain menggunakan kemampuan jaringan komputer yang dibajak, mengubah apa yang dilihat pengguna di layar, atau melakukan subversi yang hampir tak terbatas yang dimaksudkan untuk mencuri informasi atau menyebabkan pengguna melakukan aktivitas yang dapat mengakibatkan dalam mencuri informasi.

Serangan Aplikasi Web

Kerentanan dalam aplikasi web sisi server mungkin melanggar keamanan transaksi. Misalnya, kerentanan cross-side scripting (XSS) dapat memungkinkan penyerang jarak jauh membajak transaksi web dengan mencuri kredensial autentikasi atau informasi penagihan saat pengguna menelusuri situs web yang rentan. Serangan umum lainnya melibatkan eksploitasi kerentanan injeksi SQL dalam database untuk mengubah situs web yang berinteraksi dengan database secara jahat.

Serangan Pharming

Serangan farmasi mengarahkan klien ke server jahat. Misalnya, dalam serangan Sistem Nama Domain (DNS), pengalihan dilakukan dengan mengubah alamat IP server, dalam server DNS klien yang berlaku.

Mirip dengan serangan man-in-the-middle, serangan pharming berbasis DNS sangat berbahaya karena baik klien maupun server tidak perlu ditumbangkan agar serangan berhasil. Keracunan cache DNS adalah salah satu contoh serangan DNS di mana penyerang menyuntikkan informasi pengalamatan palsu ke dalam server DNS, menipunya agar mengarahkan klien ke situs web berbahaya. Versi DNS yang ditingkatkan dapat menggagalkan upaya peracunan tetapi tidak selalu diterapkan.

Sertifikat digital dimaksudkan untuk melindungi pengguna dari serangan farmasi. Sertifikat digital adalah mata uang Infrastruktur Kunci Publik (PKI) Internet, yang memungkinkan titik akhir klien mengautentikasi dan berkomunikasi secara aman dengan server dengan mengandalkan otoritas sertifikasi (CA) yang tepercaya. Situs web palsu seharusnya tidak dapat menunjukkan sertifikat valid yang diterima oleh browser pengguna. Namun, pengguna dan aplikasi yang tertanam tidak selalu memperhatikan peringatan browser untuk sertifikat yang tidak valid, dan terdapat kelemahan dalam PKI Internet yang memungkinkan penyerang memperoleh sertifikat palsu yang tampaknya ditandatangani oleh

otoritas sertifikasi dan karenanya diterima oleh browser. Sebuah penelitian mendemonstrasikan bentuk serangan ini dengan memanfaatkan kelemahan algoritma MD5 yang terkadang digunakan untuk tanda tangan digital sertifikat.[7]

Serangan DNS terakhir melibatkan menumbangkan otoritas sertifikasi Internet yang tepercaya dan pendaftar nama domain, misalnya Network Solutions. Serangan seperti itu dapat menyebabkan semua komunikasi Internet ke server dialihkan. Hal itulah yang terjadi pada CheckFree, salah satu perusahaan pembayaran tagihan online terbesar di dunia. Menurut laporan, kredensial perusahaan, yang diperlukan untuk mengubah identitas Internetnya melalui situs web Network Solutions, telah dicuri. Para ahli yakin kredensial tersebut dicuri menggunakan serangan phishing terhadap karyawan CheckFree. Empat puluh dua juta pelanggan mengakses situs pembayaran tagihan CheckFree. Alamat Internet perusahaan dialihkan selama beberapa jam. CheckFree memperingatkan lebih dari lima juta pelanggan tentang pelanggaran tersebut.[8]

Serangan CheckFree menunjukkan kerapuhan keamanan transaksi jaringan: pengembang sistem tertanam dan penyedia layanan yang bekerja keras untuk selalu mengikuti perkembangan patch keamanan terbaru, teknologi anti-malware, dan kesadaran perilaku rekayasa sosial masih sepenuhnya terekspos.

Kombinasi Serangan

Kombinasi serangan yang jelas melibatkan penggunaan phishing untuk memasang malware. Misalnya, tautan web phishing dalam email dapat menyebabkan browser pengguna terhubung ke situs web yang telah dipenuhi kerentanan zero-day flash yang menyebabkan malware diunduh secara diam-diam ke komputer pengguna.

Serangan terhadap layanan Citibusiness Citibank menggunakan kombinasi phishing dan MITM untuk mengalahkan skema otentikasi dua faktor Citibank. Situs phishing meminta kredensial aplikasi klien tetapi kemudian bertindak sebagai MITM untuk meneruskan kredensial tersebut ke situs Citibusiness yang valid. Dengan demikian, seseorang yang mencoba menguji keaslian situs akan menemukan bahwa situs tersebut merespons dengan benar terhadap login yang valid dan tidak valid.[9]

Contoh umum lainnya adalah penggunaan kelemahan aplikasi web, seperti skrip lintas situs atau kerentanan injeksi SQL, untuk meluncurkan serangan phishing dalam halaman web yang disusupi.

Seperti yang telah disinggung sebelumnya, kegagalan CheckFree kemungkinan besar disebabkan oleh kombinasi serangan phishing untuk mendapatkan kredensial yang diikuti oleh subversi DNS.

6.1.4 Upaya Modern untuk Meningkatkan Keamanan Transaksi Jaringan

Jika ada satu hal yang pembaca dapat simpulkan dari luas dan frekuensi kerentanan dan vektor serangan yang bertujuan untuk membahayakan transaksi jaringan, itu adalah fakta bahwa pengembang dan penyedia layanan telah berjuang keras untuk tetap menjadi yang terdepan dalam persaingan topi hitam. Terlalu banyak peluang untuk menyerang. Sistem

---

[7] Sotirov A, et al. MD5 Considered Harmful Today. http://www.win.tue.nl/hashclash/rogue-ca; December 30, 2008.

[8] McMillan R. CheckFree Warns 5 Million Customers After Hack. http://csoonline.com/article/474365/CheckFree_Warns_Million_Customers_After_Hack; January 7, 2009.

[9] Krebs B. Citibank Phish Spoofs 2-Factor Authentication. http://blog.washingtonpost.com/securityfix/2006/07/citibank_phish_spoofs_2factor_1.html; July 10, 2006.

operasi, middleware, dan aplikasi titik akhir saat ini tidak pernah dirancang untuk tingkat jaminan yang tinggi. Perangkat lunak ini juga sangat kompleks dan semakin kompleks setiap saat. Seperti yang telah kita bahas di Bab 1, kompleksitas menyebabkan kelemahan yang dapat dieksploitasi. Kombinasi kompleksitas dan kurangnya proses pengembangan dengan jaminan tinggi dapat dianggap sebagai dampak ganda dari perangkat lunak yang tidak aman.

Tanpa peningkatan dramatis dalam desain dan jaminan keamanan, tren ganda yaitu meningkatnya cacat perangkat lunak dan kecanggihan penyerang akan menyebabkan hilangnya kepercayaan total pada jaringan bersama seperti Internet sebagai media perdagangan atau komputasi berbasis jaringan penting lainnya. Internet sendiri tidak dirancang untuk menyediakan transaksi pribadi dan aman; sebaliknya, hal ini dirancang untuk hal yang sebaliknya: kolaborasi dan berbagi informasi. Mari kita lihat perkembangan terkini sehubungan dengan mitigasi ancaman keamanan transaksi jaringan.

Anti-malware

Banyak sistem titik akhir transaksi menjalankan beberapa bentuk perangkat lunak anti-malware. Contohnya termasuk anti-virus, anti-spam, anti-phishing, dan firewall. Anti-malware dibangun ke dalam sistem operasi, browser, klien email, dan dapat dibeli sebagai add-on. Perangkat lunak ini berharga karena dapat mendeteksi berbagai bentuk malware sebelum menyebabkan kerusakan.

Masalah dengan perangkat lunak anti-malware adalah bahwa perangkat lunak tersebut pada dasarnya merupakan teknologi reaktif; ketika kerentanan dan metode serangan baru ditemukan, protokol anti-malware dengan cepat menjadi usang.

Selain itu, malware yang diciptakan oleh topi hitam mungkin sudah beroperasi jauh sebelum terdeteksi oleh komunitas yang mencoba menggagalkannya. Sejumlah pengujian independen terhadap produk anti-malware terkemuka menunjukkan bahwa semuanya tidak mampu mengatasi sejumlah besar spesimen malware modern.

Secunia, sebuah perusahaan manajemen kerentanan, melakukan satu pengujian dan menemukan bahwa Symantec adalah yang terbaik dalam mendeteksi serangkaian file dan halaman web berbahaya. Kabar buruknya adalah Symantec hanya mendeteksi 21% (64 dari 300) spesimen malware.[10] Majalah Keamanan Informasi melakukan pengujian terhadap 8.114 spesimen malware terhadap tujuh produk vendor anti-malware yang berbeda. Produk dengan kinerja terbaik, dari Trend Micro, tidak mampu mendeteksi 7,9%, atau sekitar 640, spesimen.[11]

Terakhir, produk anti-malware ternyata rentan. Vendor anti-malware terpaksa mengubah produk mereka dengan cepat agar bisa mengikuti contoh terbaru. Pengembang anti-malware tidak mengikuti proses pengembangan ketat dengan jaminan tinggi yang memerlukan evaluasi atau sertifikasi independen. Basis Data Kerentanan Nasional CERT AS telah mencatat sekitar ratusan kelemahan keamanan pada perangkat lunak Symantec.[12] Sejumlah penelitian menemukan bahwa kerentanan anti-malware dapat dieksploitasi.[13] Anti-

[10] Kristensen T. Symantec Beats the Competition.. http://secunia.com/blog/29; October 13, 2008.

[11] Skoudis Ed, Carpenter M. Seven Desktop Security Suites Reviewed. Information Security Magazine; November 2007; http://searchsecurity.techtarget.com/magazineContent/Product-review-Seven-integrated- endpoint-security-products.

[12] National Vulnerability Database, search on vendors Symantec. http://web.nvd.nist.gov/view/vuln/search?execution¼e2s1

[13] McAfee Antivirus Vulnerability Published.http://news.zdnet.co.uk/security/0,1000000189,39191831,00.htm; March 18, 2005.

malware adalah komponen penting dari strategi keamanan transaksi modern. Namun, hal tersebut sama sekali bukan sesuatu yang dapat menjamin keamanan transaksi.

Peramban Aman

Pendekatan yang menggembirakan terhadap keamanan transaksi melibatkan sandboxing klien web tertanam dari lingkungan komputasi lainnya; teorinya adalah bahwa browser dengan sandbox lebih kecil kemungkinannya untuk terinfeksi malware dan oleh karena itu lebih mungkin dipercaya untuk transaksi yang aman. Ada sejumlah pendekatan untuk sandboxing browser. Salah satu pendekatan sandboxing menggunakan sistem operasi yang di-boot secara terpisah (versi Linux) yang dipanggil dengan tombol khusus di komputer. Kelemahannya adalah lingkungan reguler pengguna tidak dapat diakses saat lingkungan penjelajahan sandbox sedang berjalan. Reboot harus dilakukan untuk berpindah di antara dua lingkungan. Tentu saja, karena lingkungan penjelajahan alternatif berjalan pada sistem operasi tujuan umum yang menggunakan browser tujuan umum, dapat dipastikan bahwa peretas akan menargetkan lingkungan alternatif tersebut jika ingin diadopsi secara luas. Pengguna mengandalkan keamanan melalui ketidakjelasan.

Beberapa vendor produk penyimpanan portabel yang aman, seperti USB thumb drive, menawarkan lingkungan browser Linux yang dikemas secara khusus.[14] Browser diluncurkan dari dan menyimpan data sensitif ke perangkat penyimpanan yang terpasang. Kerugiannya, tentu saja, adalah biaya tambahan dan ketidaknyamanan dalam menggabungkan perangkat fisik terpisah. Selain mobilitas antar titik akhir yang disediakan oleh drive portabel, pendekatan ini memiliki satu keunggulan utama dibandingkan solusi berbasis PC seperti boot alternatif: keamanan fisik. Beberapa dari drive portabel ini disegel terhadap gangguan dan akan menghancurkan sendiri kunci kriptografi internalnya jika ada upaya untuk menembus casing perangkat secara fisik terdeteksi.

Pendekatan lain untuk mengamankan penjelajahan adalah dengan memperkuat browser itu sendiri. Browser Chrome Google terkenal karena upayanya untuk menyediakan browser yang lebih aman sehingga laman web atau plugin berbahaya akan lebih kecil kemungkinannya menyebabkan kerusakan pada komponen browser web yang tidak terkait. Namun, teknisi Google tidak menggunakan praktik pengembangan dengan jaminan tinggi untuk mengimplementasikan Chrome. Meskipun Chrome mungkin memiliki beberapa konsep isolasi yang baik secara fundamental, kurangnya komitmen terhadap rekayasa jaminan tinggi memastikan produk tidak dapat dipercaya untuk transaksi bernilai tinggi. Tidak mengherankan jika pada bulan September 2008, hanya beberapa bulan setelah rilis Chrome, peneliti keamanan menemukan kerentanan buffer overflow di browser yang memungkinkan peretas mengambil kendali penuh atas titik akhir.[15] Sejak saat itu, ratusan kerentanan telah diposting ke Database Kerentanan Nasional CERT AS.

Ada beberapa upaya akademis untuk menciptakan browser web dengan jaminan lebih tinggi. Upaya terbaru yang menjanjikan dari Samuel King di Universitas Illinois di Urbana-Champaign, yang disebut browser web OP, juga mengklaim adanya isolasi antar komponen tingkat browser. Namun, para peneliti OP telah melangkah lebih jauh dan menggunakan

[14] http://www.ironkey.com

[15] Claburn T. Another Google Chrome Security Flaw Identified. http://www.informationweek.com/news/internet/google/showArticle.jhtml?articleID¼210500290; September 5, 2008.

beberapa metode formal untuk memverifikasi perilaku browser.[16] Peningkatan tingkat jaminan dapat memberikan keyakinan kepada pengguna OP bahwa kemampuan sandboxing yang diklaim benar-benar berfungsi. Selain hambatan teknis dalam menyelesaikan produk dan mendapatkan sertifikasi independen pada tingkat jaminan tinggi (keduanya merupakan tujuan yang terhormat), hambatan utama untuk browser OP terletak pada pencapaian penerimaan pasar. Pengembang dan pengguna adalah makhluk yang nyaman; kecuali OP dapat dikonfigurasi agar terlihat dan terasa persis seperti browser paling populer (misalnya, Internet Explorer, Firefox, Safari) dan dapat mengikuti laju fungsi browser yang terus berubah (scripting, plug-in, tampilan-dan-rasa), dampak global dari penawaran semacam ini masih dipertanyakan.

Pendekatan lain terhadap keamanan browser adalah dengan menjalankan lingkungan browser di bawah mesin virtual yang berjalan secara bersamaan, bukan dengan boot terpisah. Dengan asumsi kinerja yang memadai, pendekatan alat browser virtual ini lebih baik dibandingkan dengan boot sekunder. Namun, perangkat lunak mesin virtual harus dipercaya untuk menyediakan kotak pasir browser. Hypervisor komersial pada umumnya, seperti VMware dan Xen, tidak dikembangkan menggunakan metodologi jaminan tinggi, telah melaporkan banyak kerentanan keamanan,[17] dan oleh karena itu tidak dapat dipercaya untuk menahan upaya yang dilakukan oleh penyerang canggih untuk memecahkan sandbox.

Otentikasi Dua Faktor

Otentikasi dua faktor telah menjadi respons keamanan utama dari bank dan perusahaan transaksi jaringan penting lainnya. Salah satu pendekatannya adalah dengan meminta pengguna memasukkan nomor identifikasi selain nama pengguna dan kata sandi biasa. Nomor identifikasi dapat ditampilkan pada perangkat khusus dalam faktor bentuk key fob. Nomor tersebut dihasilkan secara kriptografis dan berubah setelah periode waktu singkat. Bank akan menyediakan perangkat identifikasi ini dan menjalankan server pelengkap yang menghasilkan nomor yang cocok. SecurID RSA adalah salah satu solusi otentikasi dua faktor yang populer.

Otentikasi dua faktor bermanfaat karena meningkatkan kemungkinan klien dan server dapat saling mengautentikasi; peretas yang hanya memiliki kata sandi yang valid tidak dapat mengakses akun yang dilindungi dua faktor. Otentikasi dua faktor melindungi terhadap beberapa serangan phishing yang mencoba mencuri kata sandi. Namun, ini tidak melindungi terhadap MITM: MITM hanya dapat meneruskan kredensial pertama dan kedua ke server sebenarnya. Otentikasi multi-faktor juga tidak melindungi terhadap malware yang dapat mendukung koneksi yang diautentikasi.

Karena kemampuan malware dan serangan lain untuk melintasi koneksi yang diautentikasi, kami memandang pendekatan autentikasi multi-faktor, setidaknya dengan sendirinya, sebagai respons yang lemah terhadap iklim ketidakamanan transaksi jaringan saat ini.

---

[16] King ST, et al. Secure Web Browsing with the OP Web Browser. http://www.cs.uiuc.edu/homes/kingst/Research_files/grier08.pdf; 2008.

[17] National Vulnerability Database, search for VMware and/or Xen. http://web.nvd.nist.gov/view/vuln/search?execution¼e2s1

Seperti yang dinyatakan oleh Bruce Schneier, chief security technology officer dari raksasa telekomunikasi BT dan pakar keamanan terkenal, ".itu tidak akan berfungsi untuk otentikasi jarak jauh melalui Internet pada akhirnya akan terjadi sedikit penurunan jumlah penipuan dan pencurian identitas."[18]

Komentar Schneier mungkin sudah ada sebelumnya; pada bulan Maret 2011, sistem SecurID RSA disusupi oleh peretas yang diduga menggunakan serangan phishing terhadap karyawan RSA un tuk memasang malware Adobe Flash yang membuka server RSA bagi para penyerang. Para penyerang diduga mampu mencuri benih SecurID dan menggunakannya untuk menembus jaringan pengguna RSA SecurID, termasuk Lockheed Martin Corporation. Kita harus bertanya-tanya, jika laporan ini benar, mengapa perusahaan keamanan seperti RSA meninggalkan kunci kerajaan di server yang dapat diakses ke Internet. Ini tidak seperti meretas firewall, UTM, dan DMZ yang bukan kejadian sehari-hari di dunia Internet.

Kontrol Akses Jaringan

Kontrol Akses Jaringan (NAC) berupaya melindungi jaringan dari akses klien yang tidak memenuhi profil yang dapat dikonfigurasi. Misalnya, server NAC dapat menolak atau mengkarantina klien mana pun dalam jaringan yang terpisah secara logis yang tidak menerapkan patch sistem operasi yang tepat dan menginstal perangkat lunak anti-malware. Bentuk paling dasar dari NAC pada jaringan Ethernet adalah penggunaan otentikasi 802.1X yang dijelaskan pada Bab 5.

NAC adalah alat perusahaan yang berguna karena memungkinkan administrator sistem untuk menerapkan kebijakan kesehatan pada klien yang terhubung ke jaringan perusahaan. NAC paling berguna ketika klien jaringan adalah populasi yang terikat dan terkendali. NAC tidak sesuai untuk jaringan terbuka (yaitu Internet) di mana server tidak memiliki pengaruh terhadap perangkat keras dan perangkat lunak klien. Namun masalah terbesar NAC adalah kesulitan dalam menilai kesehatan. Mirip dengan teknologi anti-malware, NAC bersifat reaktif. Setelah klien NAC lulus uji status yang telah diprogram, malware dan risiko rekayasa sosial yang tidak terdeteksi pada klien tetap ada.

Verifikasi Perangkat Sekunder

Perangkat seluler semakin banyak mengambil bagian dalam transaksi jaringan penting. Misalnya, terminal nirsentuh komunikasi jarak dekat (NFC) yang terpasang pada telepon dapat digunakan sebagai titik akhir antarmuka manusia transaksi jaringan. Dalam kasus lain, perangkat seluler dapat digunakan sebagai mekanisme otentikasi dua faktor: alih-alih key fob terpisah, bank dapat mengirimkan nomor identifikasi kepada pengguna, yang disebut nomor transaksi seluler (mTAN) dalam pesan SMS. Dalam kasus lain, perangkat seluler digunakan sebagai verifikasi transaksi out-of-band: ketika pengguna melakukan transaksi jaringan menggunakan PC, penyedia layanan dapat mengirimkan pesan SMS untuk mendapatkan konfirmasi akhir dari pengguna sebelum melanjutkan.

Verifikasi transaksi out-of-band adalah pendekatan yang menarik karena memungkinkan klien dan server memperoleh tingkat kepercayaan yang lebih tinggi terhadap keaslian identitas masing-masing serta informasi aktual yang ditransaksikan. Teori ini

[18] Schneier B. Two-Factor Authentication: Too Little, Too Late. http://www.schneier.com/essay-083.html; April2005.

didasarkan pada asumsi bahwa penyerang akan lebih sulit menciptakan serangan terkoordinasi pada perangkat endpoint dan out-of-band.

Kritik terhadap verifikasi perangkat sekunder mencakup persyaratan untuk menggunakan perangkat seluler terpisah, potensi kurangnya jangkauan seluler, dan biaya tambahan untuk pesan SMS. Masalah lainnya adalah keamanan perangkat seluler itu sendiri. Seperti yang akan kita bahas nanti di bab ini, dengan adanya kerentanan di lingkungan operasi Google Android iPhone, BlackBerry, dan Windows Mobile, tidak ada keraguan bahwa penyerang canggih dapat dan akan merancang metode untuk menumbangkan sistem operasi telepon seluler jika sistem operasi tersebut digunakan secara luas untuk jaringan. transaksi. Server juga merupakan kelemahan potensial, karena server jahat atau rusak dapat mengikuti protokol pengiriman verifikasi out-of-band yang akan membuat pengguna percaya bahwa transaksi tersebut aman.

Zona Saluran Informasi Terpercaya

Laboratorium penelitian IBM di Zurich menciptakan Zone Trusted Information Channel (ZTIC), sebuah perangkat USB yang menyediakan protokol keamanan jaringan dan tampilan minimalnya sendiri (SSL/TLS).[19] ZTIC menyediakan saluran yang diautentikasi dan terenkripsi antara klien dan server dan sarana bagi klien untuk memverifikasi rincian transaksi dari server. Namun, alih-alih pesan SMS yang disebutkan di atas, ZTIC menggunakan tampilan sederhana dan tombol on-boardnya sendiri. Verifikasi out-of-band ZTIC, secara teori, lebih unggul daripada pesan SMS karena saluran informasi yang lebih terjamin di mana pesan verifikasi dikirimkan.

ZTIC belum disertifikasi pada tingkat jaminan yang tinggi, dan oleh karena itu kita harus berasumsi hingga terbukti sebaliknya bahwa perangkat lunak tersebut, meskipun jauh lebih sederhana daripada perangkat titik akhir pada umumnya, tidak dapat diharapkan mampu menahan penyerang yang gigih dan canggih.

Meskipun demikian, ZTIC mempunyai kelemahan yang sama dengan pendekatan telepon seluler: pengguna harus belajar menggunakan dan mengelola perangkat terpisah. Selain itu, tidak diketahui apakah biaya produksi, distribusi, dan pemeliharaan perangkat ini dapat diterima oleh sebagian besar perancang sistem. ZTIC dihubungkan ke titik akhir melalui USB dan oleh karena itu tidak cocok untuk kelas perangkat tertanam yang tidak memiliki port USB.

Kritik lain terhadap ZTIC adalah, meskipun memegang kunci pribadi yang digunakan untuk komunikasi aman, perangkat itu sendiri tidak memiliki antarmuka untuk otentikasi pengguna. Oleh karena itu, perangkat yang hilang atau dicuri lebih rentan terhadap penggunaan jahat. Perangkat diaktifkan melalui otentikasi pengguna pada PC yang dicolokkan. Namun ketidakamanan pada PC adalah alasan utama penggunaan ZTIC!

Klien Web Tervirtualisasi

Virtualisasi memungkinkan versi yang lebih baik dari konsep browser aman yang dibahas sebelumnya di mana alat klien web virtual khusus berjalan di satu mesin virtual (atau dalam proses mikrokernel asli yang tertanam saat menggunakan hypervisor mikrokernel Tipe 1), sementara lingkungan utama sistem berjalan di mesin virtual lain. mesin virtual. Hal

[19] Weigold T, et al. The Zurich Trusted Information ChanneldAn Efficient Defence against Man-in-the-Middle and Malicious Software Attacks; TRUST 2008, LNCS 4968; 2008. 75e91.

penting dalam pendekatan ini adalah penggunaan pemisahan dengan jaminan tinggi antara browser sandbox dan lingkungan pengguna utama. Gambar 6.1 menunjukkan contoh implementasi di mana lingkungan utama sistem tertanam adalah Linux, dan alat web virtual terdiri dari browser Firefox yang di-porting. Seseorang dapat dengan mudah membayangkan banyak aplikasi lain dari arsitektur ini, termasuk penggunaan beberapa peralatan virtual dan lingkungan pengguna. Misalnya, beberapa bentuk perangkat lunak pemfilteran anti-malware dapat digunakan sebagai alat virtual, sehingga mencegah malware yang dipenuhi tamu atau kerentanan sistem operasi tamu berdampak buruk pada agen anti-malware.

Pendekatan Modern terhadap Keamanan Transaksi

Meskipun kenyamanan Internet untuk transaksi keuangan dan transaksi bernilai tinggi lainnya sangat menarik, pendekatan utama terhadap keamanan tidak mampu memberikan tingkat jaminan yang memadai terhadap kerugian. Beberapa perusahaan dan organisasi pemerintah telah mencabut sambungan Internet perusahaan mereka karena takut akan serangan. Militer AS melarang USB thumb drive karena alasan yang sama.[20]

Gambar 6.1 Alat virtual browser web.

Selain kurangnya jaminan dan sertifikasi, sebagian besar pendekatan terhadap keamanan transaksi tidak cukup alami dan mudah digunakan oleh masyarakat umum. Jelas, diperlukan pendekatan dengan jaminan tinggi yang masih memungkinkan penggunaan kembali secara maksimal lingkungan lama yang diharapkan.

6.1.5 Arsitektur Transaksi Tertanam yang Dapat Dipercaya

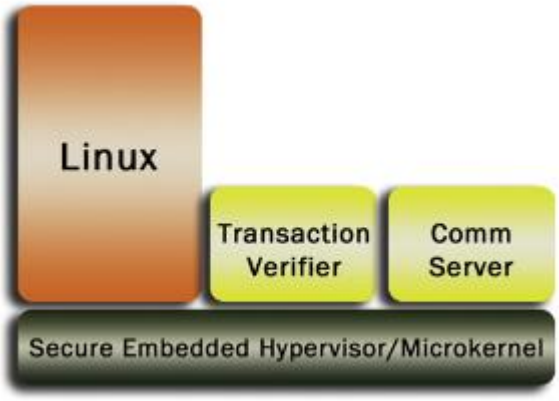
Mari kita lakukan pendekatan sandboxing ini selangkah lebih maju. Virtualisasi dapat digunakan untuk menghosting lingkungan Linux tamu sistem sambil menggunakan dua subsistem keamanan penting yang berjalan di mesin virtual terpisah atau dalam proses mikrokernel asli (jika berlaku). Subsistem pertama, disebut server komunikasi, menyediakan koneksi komunikasi yang dapat dipercaya antara sistem tertanam dan server jarak jauh. Subsistem kedua, disebut pemverifikasi transaksi, menyediakan antarmuka verifikasi terpercaya untuk klien (lihat Gambar 6.2).

[20] Shachtman N. Under Worm Assault, Military Bans Disks, USB Drives. http://blog.wired.com/defense/2008/ 11/army-bans-usb-d.html; November 19, 2008.

Proksi Komunikasi

Saat klien web membuat koneksi HTTPS ke server, lalu lintas web disaring dan dialihkan ke proxy komunikasi yang membuat koneksi kedua yang diamankan secara kriptografis dengan server. Ide utamanya di sini adalah bahwa solusi ini menciptakan koneksi yang terlindungi, secara logis out-of-band dari sistem Linux, namun masih in-band sehubungan dengan sistem tertanam. Oleh karena itu, perangkat fisik terpisah tidak diperlukan. Karena kumpulan kunci enkripsi, sertifikat server, dan perangkat lunak protokol kedua berjalan dalam proses asli yang tepercaya, data penting ini tidak dapat dicuri atau dirusak oleh sistem operasi tamu, terlepas dari infiltrasi malware. Selain itu, subsistem keamanan asli dapat memanfaatkan Trusted Platform Module (TPM) atau kemampuan ko-prosesor perangkat keras yang setara, jika ada, untuk penyimpanan kunci yang lebih kuat dan pengesahan platform.

Proksi aman mengalahkan serangan MITM serta serangan malware yang mencoba mengambil alih kunci kriptografi yang digunakan untuk komunikasi aman. Koneksi aman juga melindungi terhadap serangan pharming karena klien dan server mampu mencapai jaminan tinggi dalam protokol otentikasi bersama mereka. Menerapkan pendekatan yang sama dalam titik akhir server dapat semakin meningkatkan keamanan transaksi secara keseluruhan.



Gambar 6.2 Arsitektur transaksi tertanam yang aman.

Pemverifikasi Transaksi

Ketika transaksi akan dieksekusi, server mengirimkan permintaan verifikasi ke klien. Permintaan ini juga dikirim melalui koneksi aman dan diterima oleh hypervisor/sistem operasi tepercaya sehingga tahan terhadap serangan keamanan yang menimpa perangkat lunak operasi tujuan umum dalam sistem titik akhir klien umum. Perhatikan bahwa, bergantung pada situasinya, aplikasi server mungkin perlu diadaptasi untuk menghasilkan siklus verifikasi akhir ini. Namun keberadaan pendekatan SMS membuktikan bahwa penyedia layanan bersedia melakukan modifikasi kecil tersebut. Faktanya, pesan dengan format yang sama yang digunakan dalam verifikasi SMS dapat digunakan untuk sambungan aman.

Dalam sistem tertanam yang otonom, pemverifikasi transaksi mungkin menjalankan pemeriksaan aturan yang telah diprogram untuk memverifikasi respons transaksi dan mengirimkan kembali konfirmasinya ke server. Dalam titik akhir klien yang dioperasikan oleh pengguna manusia, verifikasi sering kali memerlukan konfirmasi manusia. Pemverifikasi dapat

memanfaatkan tampilan komputer untuk menyediakan kotak dialog verifikasi yang tidak dapat dipalsukan atau ditimpa oleh sistem operasi tamu. Pemverifikasi transaksi adalah klien dari manajer tampilan grafis tepercaya yang dihosting oleh hypervisor/sistem operasi tepercaya.

Saat manusia berada dalam lingkaran, perangkat LED atau audio khusus dapat digunakan untuk mengumumkan respons transaksi masuk, namun hal ini mungkin memerlukan modifikasi perangkat keras titik akhir. Pendekatan lain adalah dengan menggunakan area kecil pada tampilan layar, yang disebut sprite verifikasi, yang dicadangkan untuk pemverifikasi transaksi. Sistem operasi tamu tidak memiliki akses langsung ke perangkat tampilan fisik. Sebaliknya, akses tamu ke layar divirtualisasikan. Manajer tampilan grafis tepercaya akan memperbarui tampilan fisik dengan konten grafis tervirtualisasi tamu, dengan pengecualian sprite verifikasi yang tidak tumpang tindih yang isinya dikontrol oleh pemverifikasi transaksi. Ketika permintaan verifikasi transaksi masuk dari server, sprite berubah untuk memberi tahu pengguna bahwa permintaan tersebut tertunda. Misalnya sprite yang bisa berubah warna.

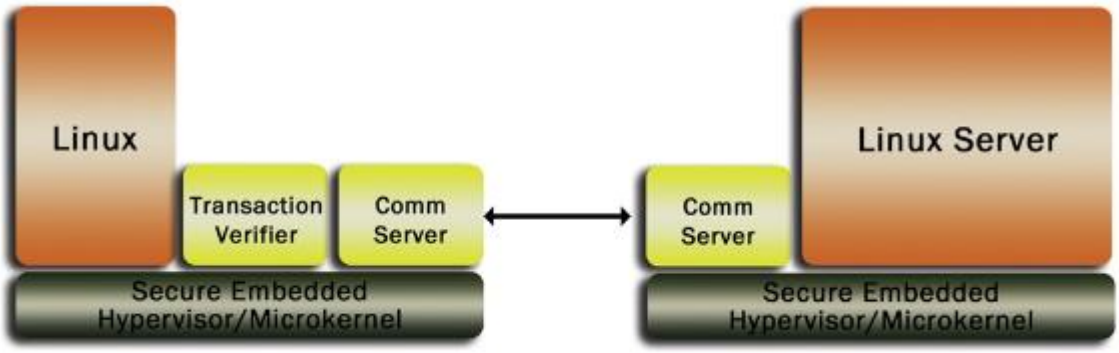
Pengguna sekarang dapat mengklik sprite (atau menggunakan urutan keyboard khusus) untuk menjalankan dialog konfirmasi aman yang dapat menggunakan kekuatan penuh layar. Tindakan fisik mengklik sprite atau menggunakan urutan keyboard yang dicadangkan memberi pengguna jalur tepercaya ke layar verifikasi, contoh implementasinya ditunjukkan pada Gambar 6.3.

Gambar 6.3 Dialog verifikasi transaksi tepercaya.

Ada sejumlah keuntungan penting dari pendekatan ini yang tidak dimiliki solusi keamanan transaksi yang ada. Manfaat yang paling penting adalah kemampuan untuk menggunakan kembali platform perangkat keras tertanam yang sama dan tidak memaksa pengguna untuk mengadopsi perangkat asing eksternal. Beberapa sistem tertanam sudah terisi penuh serta keterbatasan ruang dan biaya, sehingga tidak praktis untuk menambahkan perangkat keras tambahan.

Saat manusia terlibat, penggunaan kembali antarmuka grafis akan meningkatkan kegunaan alami. Manajer tampilan tepercaya memiliki definisi lengkap tentang grafik perangkat titik akhir untuk menyusun dialog verifikasinya. Dengan memanfaatkan tampilan secara maksimal, pengguna tidak hanya dijamin menerima dan memperhatikan permintaan verifikasi dari server yang dikonfigurasi dengan benar, namun pengalaman pengguna juga dijamin jauh lebih baik daripada yang dapat dicapai pada perangkat kecil yang terpasang dengan faktor bentuk.

Keamanan transaksi jaringan end-to-end akan ditingkatkan dengan menerapkan metodologi jaminan tinggi yang sama di sisi server. Hal ini dapat dilakukan dengan menggunakan alat transaksi jaringan (berisi hypervisor/sistem operasi dan server komunikasi tepercaya) di jaringan server atau dengan mengadopsi arsitektur virtualisasi serupa yang digunakan di titik akhir klien. Seperti terlihat pada Gambar 6.4, subsistem komunikasi aman out-of-band di kedua ujung transaksi menyediakan koneksi yang dapat dipercaya oleh klien dan server.



Gambar 6.4 Menerapkan arsitektur hypervisor/mikrokernel yang aman ke kedua titik akhir.

Menggunakan sistem operasi tepercaya untuk keamanan sisi server melindungi terhadap MITM dan malware dengan melakukan sandboxing sepenuhnya pada komponen komunikasi aman ujung ke ujung.

Penegakan Kebijakan Kriptosistem dan PKI

Kontrol end-to-end dari koneksi transaksi juga dapat melindungi terhadap kelemahan PKI dan kriptosistem dengan menerapkan penggunaan kunci publik yang kuat, enkripsi massal, dan algoritma serta panjang kunci Message Authentication Code (MAC) daripada mengandalkan web asli. server dan implementasi TLS defaultnya.

Misalnya, server komunikasi mungkin melarang penggunaan algoritma hash MD5 yang lemah dan memerlukan penggunaan kriptografi kurva elips (ECC) dengan panjang kunci 384-bit atau lebih besar untuk sistem kriptografi kunci publik. Selain itu, karena kriptografi kunci publik digunakan untuk menegosiasikan dan mendapatkan kunci simetris yang digunakan untuk enkripsi massal, kebijakan ini dapat memastikan bahwa panjang kunci publik memberikan kekuatan yang sepadan dengan sistem kriptografi simetris yang mendasarinya.

Infrastruktur berbasis hypervisor/sistem operasi yang tepercaya bahkan dapat digunakan sebagai otoritas sertifikatnya sendiri. Serangan farmasi tidak dapat menggagalkan

jaminan transaksi karena klien dan server yang sah selalu mengetahui kapan mereka berkomunikasi dengan rekan yang sah di ujung sana.

TLS dirancang agar fleksibel, menangani kompatibilitas ke belakang dan menangani realitas dunia web yang kompleks dengan banyak jenis server dan klien. Namun, fleksibilitas ini menyebabkan kerentanan, seperti kelemahan TLS, yang dibahas di Bab 5, yang memungkinkan MITM menurunkan kekuatan kriptografi yang digunakan dalam protokol. Kontrol kedua ujung koneksi melalui server komunikasi aman memungkinkan kebijakan diterapkan di tingkat TLS. Kebijakan ini secara drastis dapat meningkatkan keamanan sistem secara keseluruhan. Misalnya, kebijakan dapat mengharuskan klien mengautentikasi dirinya ke server (otentikasi klien bersifat opsional dalam standar TLS).

Masyarakat modern menjadi semakin bergantung pada keamanan transaksi jaringan melalui jaringan area luas. Transaksi jaringan digunakan untuk perdagangan dan perbankan, dan untuk mengontrol infrastruktur penting dari jarak jauh. Sistem tertanam yang menggunakan sistem operasi dan aplikasi titik akhir yang tidak aman tidak mampu memberikan keamanan yang diperlukan untuk transaksi bernilai tinggi. Sebagaimana dibuktikan oleh pertumbuhan tren seperti komputasi awan dan Perangkat Lunak sebagai Layanan (SaaS), dampak keamanan dari serangan siber yang berhasil kemungkinan besar akan meningkat. Kabar baiknya adalah metodologi dengan jaminan tinggi dan teknologi bersertifikat dengan ketahanan tinggi dapat menembus status quo ketidakamanan dan melakukannya tanpa mengorbankan sistem operasi dan aplikasi warisan dunia. Satu-satunya hal yang kita butuhkan adalah tekad untuk berpikir out of the box dan menerapkan teknik-teknik baru ini secara kreatif dan efektif.

6.2 KEAMANAN OTOMOTIF

Pada Bab 1, kita membahas proyek penelitian universitas yang meresahkan pada tahun 2010 di mana tim peneliti berhasil meretas sistem penting yang berhubungan dengan keselamatan (mesin, kerusakan) pada jaringan dalam mobil dengan keamanan rendah dan bagaimana mobil modern kini secara teratur terhubung ke jaringan luas. jaringan area menggunakan teknologi nirkabel. Kombinasi dari situasi ini menghasilkan vektor serangan terhadap infrastruktur nasional yang dilakukan oleh agen ancaman yang canggih. Seperti yang juga kita bahas di Bab 1, tantangan yang lebih besar bagi pengembang sistem tertanam adalah ledakan kompleksitas unit kontrol elektronik (ECU) yang luar biasa pada kendaraan modern, dengan lebih dari 200 mikroprosesor ditemukan di beberapa model kelas atas.

Selain jumlah dan kompleksitas sistem tertanam, beberapa jaringan dengan berbagai jenis, termasuk Controller Area Network (CAN), FlexRay, Local Interconnect Network (LIN), dan Media Oriented Systems Transport (MOST), menghubungkan ECU ini. OEM mobil mengintegrasikan komponen dan perangkat lunak ECU jaringan dari puluhan pemasok Tier-1 dan Tier-2. Meskipun OEM sering kali menetapkan persyaratan untuk ECU ini, OEM tidak secara ketat mengontrol konten aktual atau proses pengembangannya.

Tidak mengherankan jika situasi ini menjadi tidak dapat dipertahankan. OEM menderita sindrom tiang terpanjang di tenda: satu ECU, yang dikirimkan terlambat atau dengan masalah keandalan yang serius, mungkin merupakan satu-satunya hal yang diperlukan untuk menyebabkan penundaan pengiriman atau kegagalan yang terlihat oleh pelanggan yang

menyebabkan penarikan kembali dan reputasi buruk. Ditambah lagi dengan tantangan keamanan yang baru:

satu kerentanan dalam komponen penting, seperti gerbang ke jaringan dan fungsi yang penting bagi keselamatan, dapat memungkinkan masuknya penyerang jarak jauh. Meskipun kami telah menyebutkan beberapa bidang utama untuk meningkatkan keamanan di lingkungan ini, kami meninjau kembali keamanan otomotif secara lebih rinci di sini.

Ancaman dan Mitigasi Keamanan Kendaraan

Ranah ancaman keamanan terhadap mobil dapat diklasifikasikan secara kasar dalam tiga domain: fisik lokal, jarak jauh, dan internal elektronik. Kombinasi dari domain ancaman ini sering kali diperlukan untuk menimbulkan kerusakan.

Lokal-Fisik

Contoh ancaman fisik lokal adalah seseorang secara fisik memasuki jaringan CAN drivetrain dan mengganggu komunikasi atau merusak ECU melalui lonjakan listrik atau penggunaan panas yang berlebihan. Serangan invasif seperti itu dapat dengan mudah menonaktifkan fungsi penting mobil. Namun, penyerang lokal, seperti mekanik yang tidak puas atau pasangannya yang memfitnah, hanya dapat melukai satu mobil dan oleh karena itu kemungkinan besar tidak akan mendapat perhatian tim keamanan. Selain itu, sistem elektronik terdistribusi dan sangat rumit pada mobil tidak praktis untuk melindungi dari serangan fisik. Jadi kami biasanya menyerang kelompok ancaman ini.

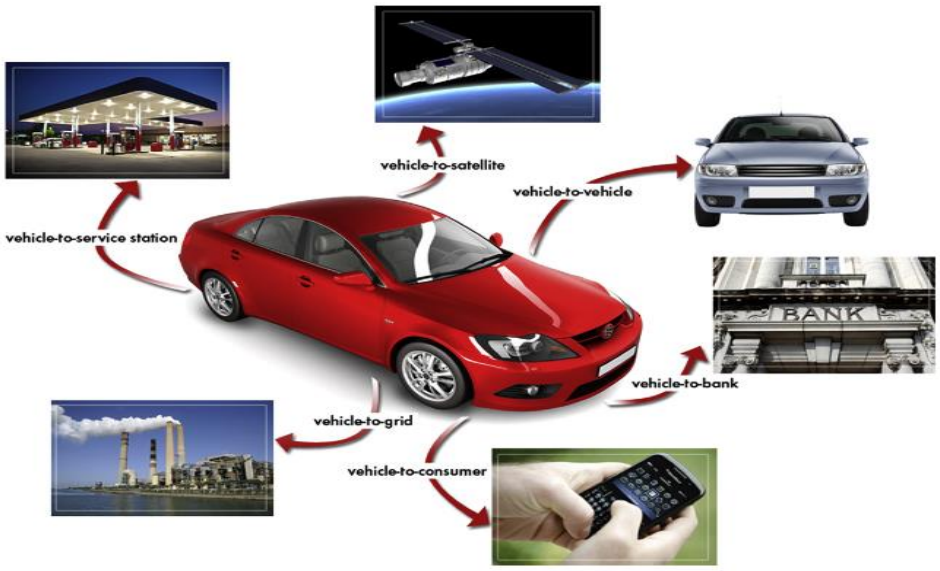
Namun, ada satu pengecualian, dan ini merupakan pengecualian yang penting. Di suatu tempat dalam satu atau lebih ECU, kunci kriptografi pribadi disimpan untuk digunakan dalam menciptakan saluran komunikasi yang dilindungi dan untuk menyediakan layanan perlindungan data lokal. Komunikasi dapat mencakup mobil ke pusat layanan atau infrastruktur OEM lainnya, mobil ke penyedia multimedia, mobil ke mobil, mobil ke jaringan listrik (pada kendaraan listrik), mobil ke ponsel pintar, atau bahkan mobil ke bank. Gambar 6.5 menunjukkan beberapa contoh sambungan radio jarak jauh pada kendaraan generasi mendatang. Perlindungan data saat tidak digunakan mungkin diperlukan untuk algoritme otomotif, konten multimedia, dan materi kriptografi.

Kunci pribadi harus disimpan dalam penyimpanan yang tahan terhadap serangan fisik yang canggih, baik invasif maupun non-invasif, karena hilangnya satu kunci pribadi saja dapat memungkinkan penyerang membuat koneksi ke infrastruktur jarak jauh yang dapat menyebabkan kerusakan luas dan kerugian properti. OEM harus mampu mencapai jaminan perlindungan utama yang tinggi di seluruh siklus hidup, mulai dari pembuatan hingga penanaman ke dalam ECU, pengiriman dan integrasi di dalam mobil, dan di lapangan. Pakar kriptografi sistem tertanam dapat membantu OEM dan pemasok mereka dengan panduan dan pengawasan di bidang ini.

Jarak Jauh

Ancaman jarak jauh adalah ancaman klasik: seorang peretas mencoba menyelidiki antarmuka radio jarak jauh mobil untuk mencari kerentanan dalam protokol keamanan jaringan, layanan web, dan aplikasi untuk menemukan jalan ke dalam kompleks elektronik internal. Tidak seperti pusat data kelas atas, mobil ini kemungkinan tidak dilengkapi dengan IDS, IPS, firewall, dan UTM yang lengkap. Terlepas dari itu, intrusi yang terjadi baru-baru ini terhadap Sony, Citigroup, Amazon, Google, dan RSA dengan jelas menunjukkan bahwa

mekanisme pertahanan ini adalah tipuan Swiss dalam melawan penyerang yang canggih. Jelasnya, sistem penting mobil harus diisolasi secara kuat dari ECU dan jaringan yang tidak penting untuk pengoperasian yang aman.



Gambar 6.5 Contoh komunikasi jaringan ekstra-kendaraan generasi berikutnya.

Jika memungkinkan, isolasi fisik lengkap ECU otomotif penting dari infrastruktur kendaraan yang terhubung ke jaringan harus diterapkan.

Internal-Elektronik

Meskipun isolasi jaringan fisik diinginkan, titik kontak pasti akan ada. Misalnya, sistem navigasi mobil, di beberapa pasar, harus dinonaktifkan saat mobil sedang bergerak, yang berarti komunikasi antar sistem dengan tingkat kepentingan keselamatan yang sangat berbeda. Selain itu, tren masa depan yang kuat menuju konsolidasi dimana mikroprosesor multi-core yang lebih kuat digunakan untuk menghosting sistem yang berbeda, mengubah banyak ECU menjadi ECU virtual, meningkatkan risiko ancaman yang disebabkan oleh perangkat lunak seperti peningkatan hak istimewa karena kerentanan sistem operasi, serangan saluran samping pada kriptografi. , dan penolakan layanan.

Oleh karena itu, arsitektur elektronik internal mobil harus dirancang dari awal demi keamanan. Titik kontak antara sistem dan jaringan kritis dan non-kritis harus dibenarkan pada tingkat manajemen tertinggi, dan titik kontak elektronik ini harus dianalisis dan disertifikasi tanpa kerentanan pada tingkat jaminan tertinggi, seperti jaminan yang dievaluasi ISO 15408 (Kriteria Umum) tingkat (EAL) 6+.

PHASEdPrinciples of High-Assurance Software Engineering, dijelaskan dalam Bab 3, yang mendukung minimalisasi kompleksitas, arsitektur komponen perangkat lunak, prinsip paling sedikit hak istimewa, proses pengembangan perangkat lunak dan sistem yang aman, dan validasi keamanan ahli independen harus dipelajari dan diadopsi oleh OEM dan disebarluaskan ke ECU pemasok.

Produsen mobil dan perusahaan kelas 1 mungkin tidak terlalu memikirkan keselamatan saat mereka merancang mobil yang akan digunakan di jalanan saat ini, namun hal tersebut jelas harus diubah. Produsen harus bekerja sama dengan spesialis keamanan

tertanam sejak awal dalam desain dan arsitektur perangkat elektronik dan jaringan di dalam mobil, serta harus meningkatkan standar rekayasa dan jaminan perangkat lunak yang berbasis keamanan.

6.3 MENGAMANKAN ANDROID

Pada Konferensi Sistem Tertanam baru-baru ini, kursus terkait sistem operasi Android mendapatkan popularitas yang mengejutkan. Perancang sistem tertanam sedang mempertimbangkan penggunaan Android untuk semua bentuk antarmuka manusia-mesin (HMI) di hampir semua industri besar yang terkait dengan sistem tertanam: unit kepala otomotif, antarmuka grafis perangkat medis, dan panel manajemen energi pintar rumah, dan sebagainya. sedikit. Android menghadirkan kepada komunitas sistem tertanam kekuatan Linux open source yang ditambah dengan antarmuka grafis dan infrastruktur toko aplikasi dari salah satu sistem operasi seluler paling populer di dunia. Selain itu, pasar yang berkembang pesat untuk solusi Manajemen Perangkat Seluler Android (MDM) memberi pengembang sistem tertanam janji infrastruktur manajemen perangkat jarak jauh kelas dunia yang dapat terhubung secara mulus dengan sistem TI back-end tradisional. Fungsi MDM mencakup pemantauan dan audit jarak jauh, pembaruan firmware, manajemen dan kontrol konfigurasi aplikasi, enkripsi data saat tidak aktif, layanan VPN, penghapusan jarak jauh (misalnya, ketika perangkat tertanam diyakini telah disusupi), dan banyak lagi. Tidak mengherankan jika para pengembang sistem tertanam tertarik dan bertanya-tanya bagaimana Android dapat membuat desain berikutnya jauh lebih fungsional dengan memanfaatkan pergerakan Android open source yang kuat.

Sebagian besar instruksi ahli di pusat Android tertanam mengenai cara mencapai persyaratan umum sistem tertanam yang penting untuk respons real-time dan keselamatan dan/atau operasi penting keamanan menggunakan sistem operasi seluler tidak dirancang untuk hal ini. Karena buku ini membahas tentang keamanan tertanam, mari kita fokus pada tantangan keamanan.

6.3.1 Retrospektif Keamanan Android

Sebagai bagian dari pengenalan awal Android pada tahun 2008, Google menggembar-gemborkan peningkatan keamanan di ponsel pintarnya. Situs web Google memuji keamanan platform tersebut: “Titik desain utama arsitektur keamanan Android adalah bahwa tidak ada aplikasi, secara default, yang memiliki izin untuk melakukan operasi apa pun yang akan berdampak buruk pada aplikasi lain, sistem operasi, atau pengguna.”[21]

Beberapa hari setelahnya peluncuran ponsel Android pertama, G1, sebuah kerentanan parah yang dipublikasikan secara luas ditemukan di browser web ponsel tersebut. Namun permasalahan keamanan G1 tidak berakhir di situ. Pada bulan November, peretas menemukan cara untuk menginstal program sewenang-wenang di ponsel, yang memicu keluhan dari Google: “Kami berusaha sangat keras untuk mengamankan Android. Ini jelas merupakan bug besar. Alasan mengapa kami menganggapnya sebagai masalah keamanan yang besar adalah karena akses root pada perangkat merusak sandbox aplikasi kami.”[22]

---

[21] Security and Permissions in Android. http://code.google.com/android/devel/security.html.

[22] Shankland S. Google Details ‘Reboot’ Bug, Android Security Fixes. http://news.cnet.com/8301-1009_3- 10093573-83.html; November 11, 2008.

Faktanya, bug Android akan secara diam-diam dan tidak terlihat menafsirkan setiap kata yang diketik sebagai perintah dan menjalankan perintah dengan hak pengguna super.[23] Seorang penulis ZDnet menyebut kerentanan ini sebagai “Terburuk. Serangga. Pernah."

Pada akhir tahun 2010, peneliti keamanan mengunggah ke pasar Android aplikasi permainan Angry Birds palsu yang secara tidak terlihat mengunduh aplikasi lain tanpa persetujuan atau sepengetahuan pengguna. Unduhan tambahan tersebut berbahaya, mencuri informasi lokasi dan kontak ponsel, serta mengirimkan pesan teks terlarang. Penelitian ini menunjukkan contoh paling buruk dari kegagalan prinsip hak istimewa paling rendah: aplikasi yang sepenuhnya tidak tepercaya yang diunduh dari Internet memiliki hak istimewa untuk mematikan perangkat lunak sewenang-wenang dan mengakses data sensitif di perangkat. Sebagai bagian dari pekerjaan mereka, para peneliti melaporkan banyak kelemahan lain di Android, termasuk kesalahan penggunaan SSL, kurangnya otentikasi aplikasi, metode mudah untuk keluar dari sandbox mesin virtual Android Dalvik melalui kode asli, dan fokus serangan. , arsitektur izin yang lemah.[24]

Selanjutnya, kami mengunjungi kembali situs web favorit kami, Basis Data Kerentanan Nasional CERT AS. Pencarian di Android menemukan banyak kerentanan dengan tingkat keparahan yang berbeda-beda. Berikut adalah contoh pelanggar terburuk:

- **CVE-2011-0680**: Memungkinkan penyerang jarak jauh membaca pesan SMS yang ditujukan untuk penerima lain
- **CVE-2010-1807**: Memungkinkan penyerang jarak jauh mengeksekusi kode arbitrer
- **CVE-2009-2999, -2656**: Memungkinkan penyerang jarak jauh menyebabkan penolakan layanan (restart aplikasi dan pemutusan jaringan)
- **CVE-2009-1754**: Memungkinkan penyerang jarak jauh mengakses data aplikasi
- **CVE-2009-0985, -0986**: Buffer overflow memungkinkan penyerang jarak jauh mengeksekusi kode arbitrer

Kami menunjukkan kerentanan khusus ini karena kerentanan tersebut termasuk dalam kategori eksploitasi jarak jauh dengan tingkat keparahan paling serius.

Kerentanan ini khusus untuk tumpukan Android yang berjalan di atas Linux. Android tentu saja juga rentan terhadap kerentanan kernel Linux. Pada Bab 1, kami menyajikan studi kasus kompleksitas Linux, mengacu pada makalah tahun 2008 yang ditulis oleh beberapa pengembang kernel Linux terkemuka. Para penulis tersebut telah menerbitkan pembaruan tinjauan statistik pengembangan kernel Linux mereka,[25] dan jumlahnya sungguh mencengangkan. Dengan lebih dari 20.000 baris kode yang dimodifikasi per hari selama siklus pengembangan, 6.000 penulis unik, dan pertumbuhan pesat dalam keseluruhan basis kode, tidak mengherankan jika lusinan kerentanan kernel Linux dilaporkan setiap tahun dan bahwa aliran kerentanan yang belum ditemukan bersifat laten di setiap distribusi Linux yang diterapkan di lapangan. Meskipun sebagian besar pertumbuhan dan churn basis kode kernel Linux disebabkan oleh penambahan dukungan untuk mikroprosesor dan periferal baru, kernel

[23] Burnette Ed. Worst. Bug. Ever. http://blogs.zdnet.com/Burnette/?p¼680; November 7, 2008.

[24] Oberheide J, Lanier Z. TEAM JOCH vs. Android. Washington, DC: Presentation made at ShmooCon 2011

[25] Kroah-Hartman G, et al. Linux Kernel Development: How Fast It Is Going, Who Is Doing It, What They Are Doing, and Who Is Sponsoring It: An August 2009 Update. http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf; August 2009.

inti itu sendiri, seperti dukungan jaringan dan sistem file, juga mengalami perubahan yang cepat. CVE-2009-1185 mendokumentasikan kelemahan dalam implementasi soket netlink Linux dan merupakan salah satu contoh kerentanan Linux yang diduga digunakan untuk menyusupi perangkat Android.

6.3.2 Rooting Perangkat Android

Rooting Android (juga dikenal sebagai jailbreaking) adalah proses penggantian kernel yang diinstal pabrikan (Linux) dan/atau partisi sistem file penting. Setelah perangkat di-root, peretas dapat mengubah perilaku Android agar sesuai dengan keinginannya. Istilah rooting berasal dari konsep hak istimewa root UNIX, yang diperlukan untuk mengubah fungsi yang dilindungi. Sasaran peretas Android berkisar dari keinginan penghobi untuk melakukan overclock CPU untuk kinerja yang lebih baik (dengan mengorbankan masa pakai baterai) dan menginstal aplikasi khusus hingga tindakan yang lebih jahat seperti mendapatkan layanan jaringan operator secara ilegal dan memasang key logger dan pengintai SMS. Kumpulan file baru dan pengganti yang diinstal oleh peretas disebut sebagai ROM khusus, referensi lain yang tidak sempurna untuk konsep firmware yang sering digunakan dalam memori hanya-baca.Peretas sering menggunakan kerentanan Android untuk melakukan root pada ponsel Android.

Tingkat penemuan kerentanan Android sedemikian rupa sehingga hampir setiap perangkat konsumen berbasis Android telah di-root dalam waktu singkat, terkadang dalam satu atau dua hari setelah dirilis. Selain kerentanan perangkat lunak, masalah boot aman adalah sumber utama serangan rooting Android. Beberapa pembuat perangkat Android, seperti Barnes & Noble dengan Nook Color-nya, telah mengizinkan (jika tidak dianjurkan) rooting untuk memfasilitasi komunitas pengembang yang lebih luas dan penjualan perangkat. Dalam hal ini, rooting biasanya dilakukan dengan bentuk side loading/booting menggunakan kartu SD atau USB untuk menampung atau menginstal custom ROM. Boot loader yang dipasang oleh pabrikan tidak mengautentikasi firmware Android secara kriptografis, sehingga membuka jalan bagi eksekusi ROM.

Beberapa pembuat perangkat berusaha keras untuk mencegah rooting karena berbagai alasan. Tentu saja, banyak pengembang tertanam yang menggunakan Android ingin mengunci OS Android sepenuhnya untuk mencegah pengobatan tidak sah dan gangguan berbahaya. Salah satu kegagalan boot aman paling terkenal di dunia ini adalah Amazon Kindle. Tujuan yang diduga dari mengunci Kindle adalah untuk memaksa pengguna mengakses konten Amazon dan mengharuskan penggunaan perangkat lunak e-reader Kindle. Pendekatan boot aman Amazon mencoba mengautentikasi file sistem penting saat startup menggunakan pemeriksaan tanda tangan digital. Peretas menggunakan kerentanan di Linux untuk menghindari pemeriksaan ini dan menjalankan kode boot berbahaya, melakukan rooting pada perangkat. Informasi tentang intrusi yang berhasil dapat ditemukan di Internet publik. Menarik untuk dicatat bahwa Amazon meninggalkan postur anti-rootingnya dengan tablet Kindle Fire.

Ya, kami memberikan gambaran suram tentang keamanan Android. Namun, gambaran tersebut didasarkan pada fakta sederhana yang tidak mengherankan: Android tidak pernah dirancang untuk memberikan jaminan fungsi keamanan yang tinggi.

6.3.3 Perlindungan Data Ponsel: Studi Kasus Pertahanan Mendalam

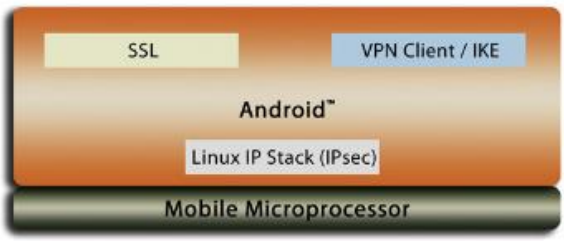
Popularitas Android yang luar biasa ditambah dengan kurangnya keamanan yang kuat telah memicu ketergesaan yang ketat oleh vendor perangkat lunak, OEM perangkat, integrator sistem, dan evaluator keamanan pemerintah untuk menemukan cara memperbaiki keamanan sistem yang lebih baik pada perangkat berbasis Android.

Salah satu pendekatan untuk meningkatkan tingkat jaminan perlindungan data dalam perangkat berbasis Android adalah dengan menggunakan beberapa lapisan enkripsi. Ponsel pintar Android, misalnya, dapat menggunakan klien SSL VPN lapisan empat (model OSI) untuk membuat sesi komunikasi data yang dilindungi. Aplikasi IPsec VPN, yang berjalan pada lapisan tiga, dapat digunakan untuk membuat koneksi independen kedua antara ponsel cerdas dan titik akhir jarak jauh (lihat Gambar 6.6). Koneksi sekunder ini menggunakan kunci publik independen untuk mewakili identitas statis titik akhir. Data dalam perjalanan dienkripsi ganda oleh dua koneksi bersamaan ini. Pendekatan keamanan berlapis ini merupakan contoh pendekatan pertahanan mendalam.

Konsep pertahanan mendalam berasal dari militer: pertahanan berlapis, seperti kombinasi ranjau dan kawat berduri, bukan hanya ranjau atau kawat berduri saja, untuk meningkatkan kemungkinan keberhasilan pertahanan serta potensi untuk berhasil. memperlambat kemajuan penyerang. Pertahanan mendalam telah berhasil diterapkan dalam perang sejak zaman kuno, dan konsep ini masih berlaku di dunia keamanan informasi.

Mari pertimbangkan beberapa ancaman terhadap aplikasi perlindungan data SSL. Penyerang dapat langsung menyerang aplikasi, mungkin mengeksploitasi kelemahan dalam tumpukan perangkat lunak SSL, untuk menonaktifkan enkripsi sepenuhnya atau mencuri kunci enkripsi yang berada di RAM selama pengoperasian. Penyerang dapat mencoba mencuri kunci SSL publik statis yang disimpan di disk. Jika kunci ini disusupi, penyerang dapat menyamar sebagai identitas terkait untuk mendapatkan akses ke klien jarak jauh melalui sesi SSL berbahaya. Malware di tempat lain di sistem Android dapat menggunakan serangan saluran samping untuk memecahkan enkripsi SSL dan memulihkan kuncinya.

Perlindungan data SSL/IPsec berlapis adalah aplikasi pertahanan mendalam yang masuk akal untuk melawan ancaman ini. Jika penyerang mampu memecahkan enkripsi SSL, lapisan IPsec akan terus melindungi data. Penyerang mungkin dapat mencuri kunci SSL tetapi tidak dapat mencuri kunci IPsec. Penyerang mungkin dapat menginstal malware ke dalam aplikasi SSL tetapi tidak pada aplikasi IPsec. Aplikasi SSL mungkin menunjukkan kelemahan saluran samping yang kebal terhadap aplikasi IPSec. Agar berhasil, penyerang harus memecahkan lapisan enkripsi SSL dan IPsec.



Gambar 6.6 SSL/IPsec dalam kotak pasir asli Android.

Jelasnya, pendekatan berlapis ini bergantung pada independensi lapisan tersebut. Lingkungan runtime harus menyediakan isolasi yang kuat pada lapisan aplikasi SSL dan IPsec, dan lingkungan runtime itu sendiri tidak boleh menyediakan permukaan serangan untuk memecahkan isolasi tersebut. Sebagian besar penelitian dan pengembangan produk yang ditujukan pada keamanan Android berfokus, dalam satu atau lain bentuk, pada penyediaan sandbox untuk isolasi data dan eksekusi fungsi-fungsi penting yang terlindungi. Kotak pasir tersebut akan digunakan untuk mewujudkan pendekatan enkripsi berlapis. Sekarang mari kita bandingkan dan kontraskan berbagai pendekatan pada sandboxing Android. Pengembang tertanam yang mempertimbangkan adopsi Android dalam desain generasi berikutnya dapat menggunakan perbandingan ini untuk membuat pilihan keamanan yang masuk akal.

6.3.4 Pendekatan Sandbox Android

Perangkat Keras Terpisah

Salah satu pendekatan sandboxing yang potensial adalah dengan memiliki beberapa mikroprosesor yang didedikasikan untuk tugas-tugas berbeda. Meskipun OEM ponsel pintar Android kemungkinan tidak akan menambah biaya perangkat keras tambahan pada desain mereka, pengembang sistem tertanam mungkin memiliki lebih banyak pilihan tergantung pada banyak faktor, termasuk fleksibilitas faktor bentuk. Misalnya, desain berkemampuan PCI mungkin dapat menghosting kartu VPN IPsec yang membungkus enkripsi lapisan kedua di sekitar SSL Android prosesor utama. Namun dalam beberapa kasus, ukuran, berat, daya, dan biaya perangkat keras tambahan akan menjadi penghalang untuk pendekatan ini.

Multi-Boot

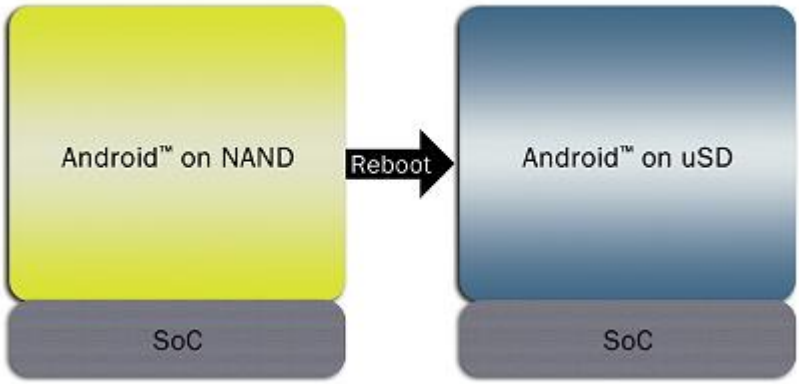
Konsep multi-boot telah dicoba pada beberapa laptop dan netbook selama bertahun-tahun. Dalam skenario laptop dual-boot, sistem operasi sekunder, biasanya Linux yang diperkecil, dapat diluncurkan sebagai pengganti sistem operasi platform utama. Sistem yang diperkecil biasanya hanya digunakan untuk penjelajahan web, dan tujuan utamanya adalah memungkinkan pengguna untuk mulai menjelajah dalam beberapa detik setelah cold boot. Sistem operasi sekunder berada di penyimpanan terpisah dan tidak pernah berjalan bersamaan dengan sistem operasi platform utama. Dalam beberapa kasus, lingkungan ringan dijalankan pada mikroprosesor sekunder (misalnya, SoC ARM yang tidak bergantung pada prosesor Intel utama netbook). Pada perangkat seluler Android, Android utama dapat dihosting di flash NAND internal, dan Android sekunder dapat dihosting di kartu microSD yang dimasukkan (lihat Gambar 6.7).

Sistem operasi sekunder memiliki isolasi yang baik dari sudut pandang keselamatan; namun, ketidaknyamanan saat melakukan boot ulang dan ketidakmampuan untuk beralih antar lingkungan dengan lancar telah sangat membatasi penerapannya. Opsi multi-boot juga tidak praktis untuk kasus penggunaan enkripsi berlapis yang memerlukan eksekusi kotak pasir secara bersamaan.

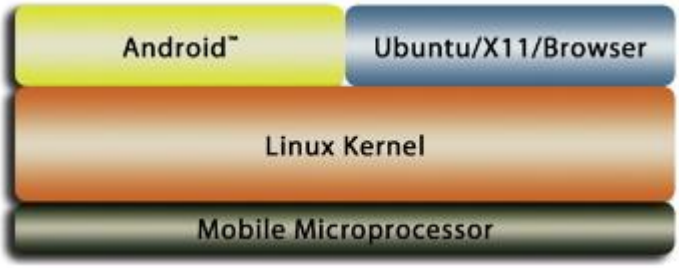
Webtop

Konsep webtop menyediakan lingkungan penjelajahan terbatas (webtop) yang tidak bergantung pada lingkungan sistem operasi utama. Namun, alih-alih melakukan dual boot, webtop berjalan sebagai sekumpulan aplikasi di atas sistem operasi utama. Dalam kasus smartphone Android Motorola Atrix, yang dirilis pada tahun 2011, webtop sandbox adalah partisi sistem file independen yang berisi kepribadian terbatas berbasis Ubuntu Linux (lihat

Gambar 6.8). Partisi Android utama terletak pada perangkat flash NAND internal yang sama di dalam telepon. Webtop Atrix dimaksudkan untuk menyediakan lingkungan seperti desktop bagi pengguna yang menyambungkan ponsel pada peralatan keyboard/video/mouse (KVM) yang dibeli secara terpisah.



Gambar 6.7 Android boot ganda.



Gambar 6.8 Lingkungan webtop Android.

Meskipun webtop kemungkinan besar tidak dimaksudkan sebagai kemampuan keamanan, salah satu pemetaan pendekatan ini terhadap kasus penggunaan enkripsi berlapis adalah dengan mengeksekusi IPsec dari lingkungan Android utama dan sesi web berbasis SSL dari sandbox webtop. Masalah dengan pendekatan ini adalah isolasi SSL webtop dari Android IPSec bergantung pada keseluruhan kernel Linux, termasuk tumpukan TCP/IP-nya.

Kontainer Terenkripsi Pengelolaan Perangkat Seluler

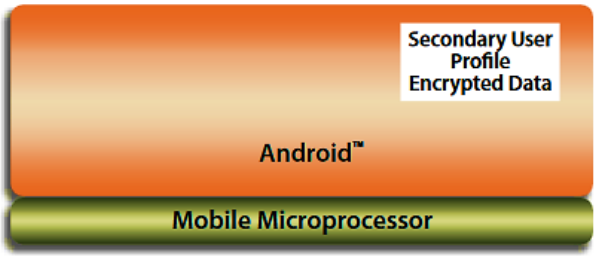
Semakin populernya perangkat seluler Android dan keinginan untuk menggunakannya di tempat kerja telah melahirkan lusinan produk dan perusahaan Mobile Device Management (MDM).

Dua tujuan utama Manajemen Perangkat Seluler adalah untuk menyediakan perlindungan data seluler dan layanan manajemen TI. Pengelolaan mencakup konfigurasi aplikasi (memastikan bahwa semua karyawan memiliki seperangkat perangkat lunak yang telah dimuat sebelumnya), audit, pengelolaan dokumen, dan penghapusan jarak jauh (menonaktifkan handset ketika seorang karyawan meninggalkan perusahaan). Perlindungan data mencakup data saat istirahat dan data dalam perjalanan (misalnya, VPN ke jaringan perusahaan).

Solusi MDM Android sering kali menggunakan enkripsi tingkat aplikasi. Misalnya, klien email perusahaan dapat menerapkan protokol enkripsinya sendiri untuk koneksi antara

perangkat seluler dan server email perusahaan dan enkripsinya sendiri terhadap folder email yang ada di telepon.

Beberapa solusi MDM menggunakan profil Android untuk membagi sistem Android menjadi dua rangkaian aplikasi: satu untuk lingkungan pribadi pengguna dan satu lagi untuk lingkungan yang dikelola perusahaan (lihat Gambar 6.9). Saat profil perusahaan dipanggil, produk MDM dapat secara otomatis mengaktifkan enkripsi untuk data yang terkait dengan profil tersebut.



Gambar 6.9 wadah MDM.

Jelasnya, pendekatan ini dapat digunakan untuk mengimplementasikan kasus penggunaan enkripsi berlapis: aplikasi MDM dapat membuat koneksi SSL di atas koneksi IPsec Android yang mendasarinya. Namun, sekali lagi, keamanan kedua lapisan bergantung pada sistem operasi Android yang mendasarinya.

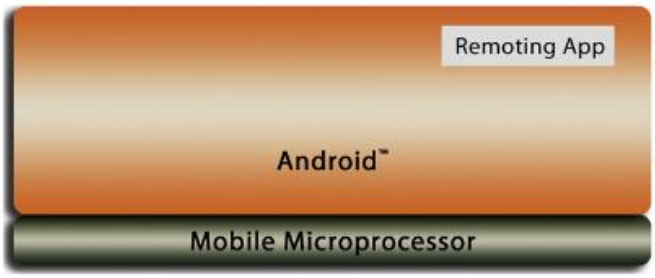
Jarak Jauh

Salah satu pendekatan terhadap perlindungan data perusahaan di Android adalah dengan tidak mengizinkan data perusahaan apa pun di perangkat seluler itu sendiri. Sebaliknya, satu-satunya cara untuk mengakses informasi perusahaan adalah menggunakan desktop jarak jauh dan/atau virtualisasi aplikasi. Jika perangkat tidak terhubung ke perusahaan (misalnya operasi offline), aplikasi dan layanan perusahaan tidak tersedia.

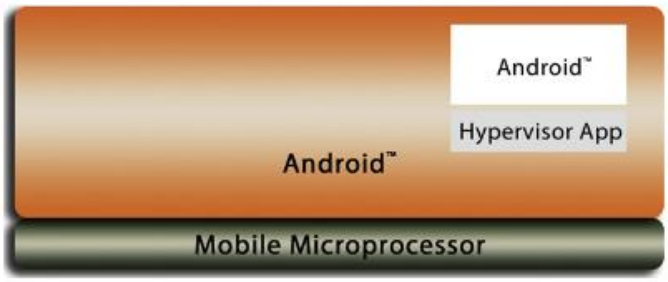
Jarak jauh menghalangi persyaratan perlindungan data lokal; namun, kasus penggunaan kami untuk perlindungan data bergerak yang berlapis tetap ada. Aplikasi jarak jauh (lihat Gambar 6.10) menyediakan enkripsi SSL sementara Android yang mendasarinya menjalankan IPsec. Namun, sekali lagi, keamanan kedua lapisan bergantung pada sistem operasi Android yang mendasarinya.

Hypervisor Tipe-2

Hypervisor tipe-2, yang diperkenalkan di Bab 2, mirip dengan webtop dan kontainer MDM di mana lingkungan sekunder dijalankan sebagai aplikasi di atas sistem operasi utama. Namun, alih-alih hanya menghosting browser, persona sekunder adalah sistem operasi tamu lengkap yang berjalan dalam mesin virtual yang dibuat oleh aplikasi hypervisor (lihat Gambar 6.11). Hypervisor menggunakan sistem operasi utama untuk menangani I/O dan fungsi manajemen sumber daya lainnya.



Gambar 6.10 Jarak jauh.



Gambar 6.11 Hypervisor tipe-2.

Produk hypervisor seluler Tipe-2 digunakan untuk memberikan persona manajemen perusahaan di atas lingkungan Android utama. Android yang tervirtualisasi dapat menggunakan koneksi SSL ke perusahaan sementara IPsec Android yang mendasarinya juga digunakan untuk menggabungkan komunikasi antar titik akhir.

Namun, sekali lagi, model Tipe-2 gagal memberikan isolasi yang kuat. Kesalahan atau kerentanan keamanan pada sistem operasi tujuan umum utama akan berdampak pada fungsi penting yang berjalan di mesin virtual. Selain itu, aplikasi hypervisor Tipe-2 yang diterapkan di ruang perusahaan ternyata mengandung kerentanan yang dapat merusak sandbox.

Kotak Pasir Dibangun di Atas Pasir

Pembaca tetap, jika Anda telah dengan tekun menyerap konsep keamanan perangkat lunak sistem (Bab 2), pengembangan perangkat lunak yang aman (Bab 3), dan sisa buku ini, kami harap Anda melihat dengan jelas kelemahan umum di antara semua sandboxing yang umum. pendekatan yang dijelaskan sebelumnya. Berbagai aplikasi Android, container MDM, aplikasi jarak jauh, webtop, dan hypervisor Tipe-2 semuanya berupaya untuk menyesuaikan keamanan pada lingkungan Android itu sendiri. Seperti yang telah kami tunjukkan secara rinci di awal bab ini, sistem Android/Linux, meskipun menyediakan fungsionalitas multimedia yang kaya sehingga desain tertanam dapat memanfaatkannya, namun penuh dengan kerentanan keamanan yang tidak dapat diperbaiki. Keamanan dengan jaminan tinggi harus dirancang sejak awal. Namun, meskipun jaminan tinggi tidak dapat diterapkan pada Android itu sendiri, jaminan tersebut dapat diterapkan pada tingkat sistem. Mari kita lihat caranya.

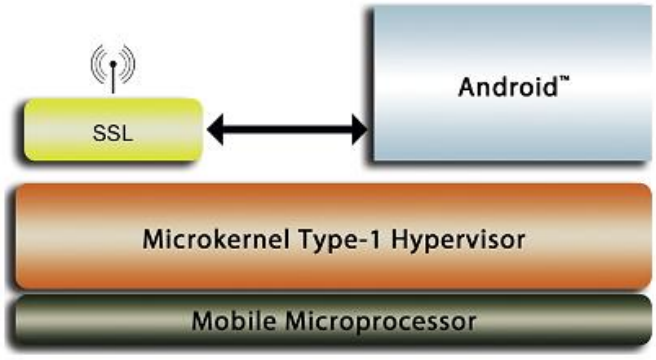
Hypervisor Tipe-1

Hypervisor tipe-1, yang diperkenalkan di Bab 2, juga memberikan kelengkapan fungsional dan eksekusi bersamaan dari persona perusahaan sekunder. Namun, karena hypervisor berjalan pada bare metal, isolasi persona tidak dapat dilanggar oleh kelemahan sistem operasi persona. Dengan demikian, hypervisor Tipe-1 mewakili pendekatan yang

menjanjikan baik dari segi fungsionalitas dan keamanan. Namun, ancaman kerentanan hypervisor masih ada, dan tidak semua hypervisor Tipe-1 dirancang untuk memenuhi tingkat keselamatan dan keamanan yang tinggi.

Salah satu varian tertentu, hypervisor Tipe-1 berbasis mikrokernel, dirancang khusus untuk memenuhi persyaratan keamanan penting. Seperti yang dibahas di Bab 2, mikrokernel menyediakan arsitektur yang unggul untuk keselamatan dan keamanan dibandingkan sistem operasi tujuan umum yang besar seperti Linux dan Android. Pada hypervisor mikrokernel Tipe-1, virtualisasi sistem dibangun sebagai layanan pada mikrokernel. Jadi, selain mesin virtual yang terisolasi, mikrokernel menyediakan antarmuka standar terbuka untuk aplikasi kritis ringan, yang tidak dapat dipercayakan kepada tamu tujuan umum. Misalnya, SSL dapat dihosting sebagai aplikasi mikrokernel, memberikan tingkat jaminan setinggi mungkin untuk lapisan enkripsi ini. Paket IPsec yang berasal dari Android dienkripsi ganda dengan layanan lapisan SSL jaminan tinggi sebelum ditransmisikan melalui antarmuka nirkabel (lihat Gambar 6.12).

Mikrokernel waktu nyata adalah pilihan yang sangat baik untuk sistem tertanam karena mikrokernel dapat menampung aplikasi apa pun yang tertanam dan waktu nyata yang tidak sesuai untuk lingkungan Android/Linux. Hypervisor mikrokernel Tipe-1 biasanya menggunakan MMU mikroprosesor untuk mengisolasi ruang memori lingkungan Android utama dan aplikasi enkripsi SSL asli. Namun, driver perangkat di Android mungkin menggunakan DMA yang dapat melanggar partisi memori dengan melewati MMU sepenuhnya. Pendekatan potensial untuk menjaga vektor serangan ini termasuk menjalankan hypervisor di TrustZone pada mikroprosesor berbasis ARM yang berlaku, menggunakan IOMMU, atau memediasi semua master bus DMA. TrustZone dan IOMMU dibahas lebih lanjut di Bab 2. Sifat isolasi dari beberapa mikrokernel yang aman bahkan dapat melindungi terhadap serangan rahasia dan serangan perangkat lunak saluran samping yang canggih.



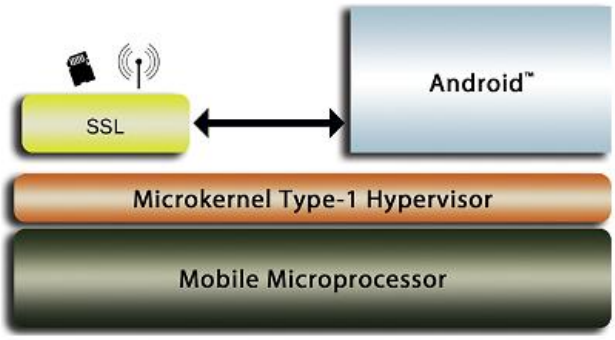
Gambar 6.12 Pendekatan hypervisor Microkernel Tipe-1 untuk enkripsi data bergerak berlapis.

Keamanan Fisik

Setelah kita memiliki pendekatan yang mencegah serangan perangkat lunak agar tidak merusak kotak pasir di antara lapisan perlindungan, perangkat yang tertanam dapat mengambil manfaat dari mengambil langkah pertahanan mendalam untuk melindungi sistem enkripsi berlapis dari serangan fisik menggunakan elemen aman yang terpasang.

Perangkat seluler yang hilang atau dicuri di tangan penyerang yang canggih, misalnya, rentan terhadap pengintaian memori, analisis kekuatan, dan serangan fisik invasif dan non-invasif lainnya. Meskipun perlindungan fisik seluruh sistem tertanam mungkin tidak praktis, perlindungan fisik yang ditargetkan dapat membuat perbedaan besar dalam keamanan sistem secara keseluruhan. Seperti yang dibahas di Bab 4, elemen aman dapat digunakan untuk memberikan perlindungan fisik terhadap parameter penting, termasuk kunci privat. Kunci publik statis layanan mikrokernel SSL asli dapat ditempatkan di elemen aman. Misalnya, kartu pintar dapat dimasukkan ke dalam perangkat microSD (yang sekaligus dapat digunakan untuk penyimpanan tambahan yang aman) dan dipasang ke ponsel cerdas atau sistem tertanam (lihat Gambar 6.13). Tentu saja, penerapannya akan bervariasi tergantung pada jenis dan kecanggihan perlindungan fisik yang tersedia. Perancang sistem tertanam dapat memeriksa tingkat sertifikasi FIPS 140 dan dokumentasi produk untuk membandingkan dan membedakan penawaran.

Enkripsi berlapis sebagai strategi pertahanan mendalam adalah pendekatan yang masuk akal untuk meningkatkan jaminan layanan perlindungan data berbasis Android; Namun, yang tidak masuk akal adalah menjalankan kedua lapisan tersebut di dalam lingkungan Android itu sendiri.



Gambar 6.13 Menambahkan perlindungan keamanan fisik melalui kartu pintar yang terpasang ke hypervisor mikrokernel Tipe-1.

Terdapat terlalu banyak kerentanan untuk mencegah kedua lapisan tersebut ditumbangkan secara bersamaan. Perancang sistem tertanam yang mempertimbangkan Android juga harus hati-hati melakukan sandbox pada fungsi keamanan penting di luar sistem Android. Mikroprosesor tertanam modern dan solusi perangkat lunak sistem menyediakan fitur-fitur yang diperlukan untuk mendapatkan yang terbaik dari kedua dunia: kekuatan infrastruktur penerapan multimedia dan aplikasi Android di samping, namun terpisah secara aman dari, fungsi keamanan sistem yang penting.

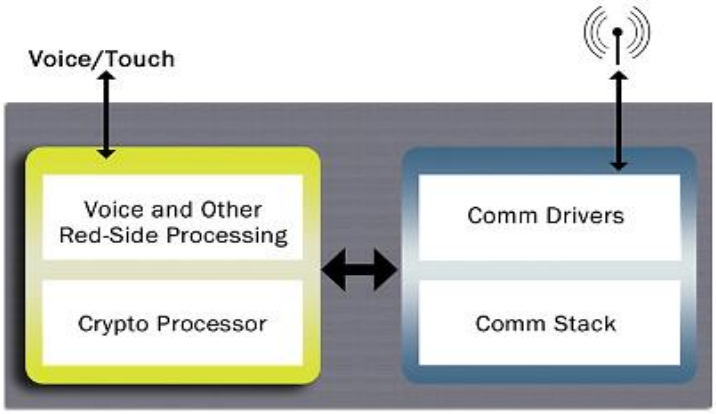
6.4 RADIO BUATAN PERANGKAT LUNAK GENERASI BERIKUTNYA

6.4.1 Pemisahan Merah-Hitam

Radio yang ditentukan perangkat lunak (SDR) yang memproses informasi rahasia biasanya dirancang dengan pemisahan merah-hitam standar di mana sisi merah bertanggung jawab atas pemrosesan informasi sensitif dan fungsi kriptografi, dan prosesor sisi hitam bertanggung jawab atas tumpukan komunikasi dan driver. Sisi merah dan hitam ditempatkan

pada komponen perangkat keras yang terpisah. Faktanya, sisi merah biasanya terdiri dari prosesor tujuan umum dan prosesor kriptografi terpisah.

Saat keluar, informasi rahasia yang berasal dari sisi merah dienkripsi dan dikirim melalui beberapa interkoneksi ke sisi hitam untuk transmisi. Saat masuk, informasi yang diterima oleh driver sisi hitam diteruskan ke interkoneksi untuk dekripsi dan pemrosesan sisi merah lainnya, seperti penjaga dan otentikasi. Arsitektur umum ini ditunjukkan pada Gambar 6.14. Perhatikan bahwa komponen kripto sisi merah logis dapat mencakup perangkat keras tujuan khusus yang terpasang, seperti FPGA, untuk eksekusi algoritma kriptografi, penyimpanan kunci, dan redundansi fisik. Perangkat lunak apa pun yang berjalan dalam batas kriptografi berada di bawah pengawasan ketat NSA selama sertifikasi Tipe-1. Hal ini tentu saja memberlakukan permintaan jaminan yang ketat dan batasan kompleksitas pada perangkat lunak sisi merah.



Gambar 6.14 Perangkat komunikasi tradisional berarsitektur merah-hitam.

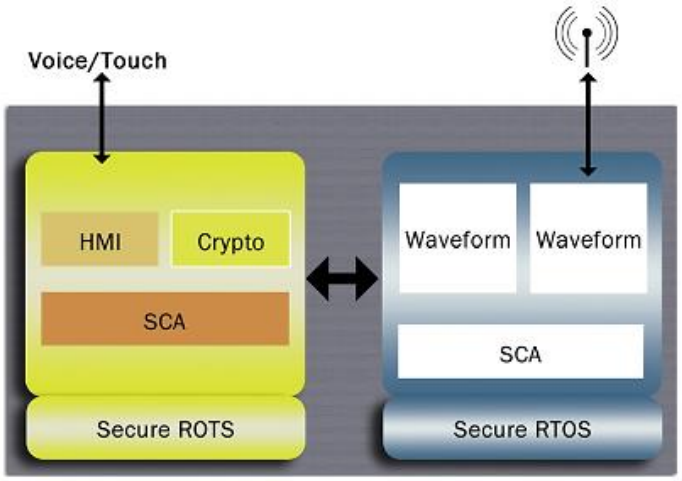
6.4.2 Arsitektur Radio Buatan Perangkat Lunak

Sekarang mari kita lihat arsitektur merah-hitam secara lebih rinci, dengan menggunakan contoh software-defined radio (SDR). Lebih khusus lagi, mari kita pertimbangkan perangkat yang menggunakan Arsitektur Komunikasi Perangkat Lunak (SCA), sebuah kerangka kerja terbuka yang digunakan untuk mengembangkan SDR. SCA menyediakan API standar untuk mengelola bentuk gelombang dan sumber daya lain yang relevan dengan radio. SCA berada di atas sistem operasi yang sesuai dengan SCA. Dalam SDR, SCA dapat digunakan pada sisi merah dan hitam. Sisi hitam biasanya mencakup komponen yang diperlukan untuk mengelola komunikasi radio, termasuk bentuk gelombang itu sendiri. Oleh karena itu, sistem operasi real-time (RTOS) seringkali penting.

Sisi merah mungkin menggunakan lebih sedikit fungsi SCA. Minimal, penyampaian pesan berstandar SCA dan fungsi manajemen komponen akan disertakan. Namun yang perlu diperhatikan adalah peran sisi merah dalam mengelola teks biasa. Komponen antarmuka manusia-mesin (HMI), seperti driver keyboard, codec suara, dan perangkat lunak manajemen layar sentuh, akan disertakan dalam pemrosesan sisi merah. Misalnya, operator radio genggam mungkin mengucapkan data suara yang harus dikumpulkan oleh sisi merah untuk pemrosesan kriptografi sebelum dapat dikirim ke sisi hitam untuk transmisi.

Seperti halnya perangkat komunikasi Tipe-1 lainnya, meminimalkan konten perangkat lunak yang relevan dengan sertifikasi di sisi merah adalah tujuan strategis pengembang SDR

serta lembaga sertifikasi. Karena persyaratan real-time dan keamanannya yang penting, RTOS yang tepercaya adalah pilihan yang tepat. Contoh arsitektur yang menunjukkan komponen SDR sisi merah dan hitam utama ditunjukkan pada Gambar 6.15.



Gambar 6.15 Arsitektur radio tradisional yang ditentukan perangkat lunak.

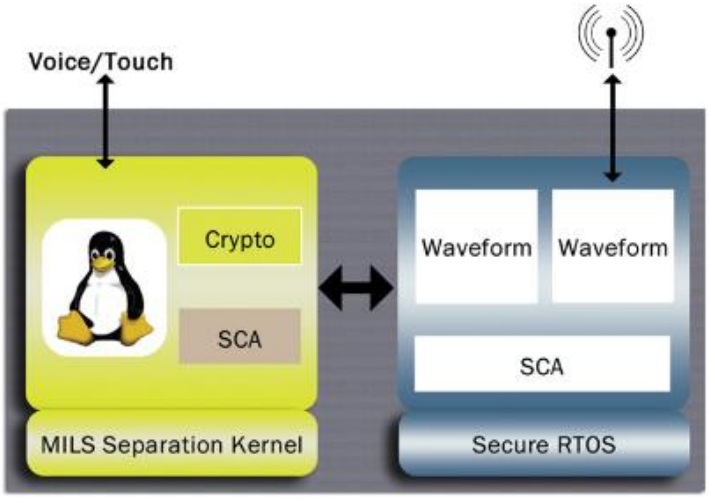
6.4.3 Masuk ke Linux

Meskipun radio suara analog memiliki HMI yang relatif sederhana, radio suara/data digital multiguna mungkin menggunakan HMI yang jauh lebih canggih. Linux dan variannya merupakan pilihan yang jelas untuk implementasi HMI karena ketersediaan kerangka HMI populer yang banyak digunakan di industri elektronik konsumen. Seperti yang telah kita bahas dalam studi kasus Android sebelumnya dan di Bab 1, Linux, sistem operasi monolitik, tidak memenuhi persyaratan sertifikasi dengan jaminan tinggi dan tidak sesuai untuk tugas-tugas real-time yang sulit dan berlatensi rendah. Untuk memasukkan Linux ke dalam HMI sisi merah perangkat komunikasi yang aman tanpa membawa seluruh basis kode sumber multi-juta baris ke dalam batas kriptografi, pilihan yang jelas adalah dengan memasukkan prosesor aplikasi kedua yang didedikasikan untuk Linux. Namun, menambahkan prosesor kedua akan meningkatkan jejak, daya, biaya produksi, dan kompleksitas sistem. Hal ini terutama menjadi masalah pada perangkat dengan sumber daya terbatas, seperti radio bertenaga baterai.

Ketika mereka digabungkan menggunakan prosesor sisi merah sekunder, sistem operasi tujuan umum biasanya hanya digunakan untuk komunikasi yang tidak rahasia. Komunikasi rahasia memerlukan antarmuka HMI khusus untuk mencapai jaminan integritas dan ketersediaan data dan perintah sensitif yang tinggi. Cawan Sucinya, tentu saja, adalah memasukkan Linux ke dalam prosesor merah asli tanpa mengorbankan kinerja waktu nyata; membahayakan keamanan; meningkatkan biaya sertifikasi; atau pertumbuhan jejak sistem, bobot, daya, dan biaya. Selain itu, kami ingin menggunakan Linux untuk antarmuka yang terklasifikasi maupun yang tidak terklasifikasi, memungkinkan pengguna untuk sepenuhnya menyadari manfaat produktivitas yang kaya dari Linux.

Tujuan ini dimungkinkan dengan menggunakan hypervisor Tipe-1 berbasis mikrokernel, di mana mikrokernel tersebut merupakan kernel pemisahan MILS. Bab 2 menjelaskan kernel pemisahan MILS dan MILS. Kernel pemisahan dapat menghosting Linux di mesin virtual yang terkotak dengan aman. Namun, tidak seperti hypervisor tradisional, kernel pemisahan MILS

dapat menampung aplikasi asli dan juga tamu. Penjadwalan sumber daya dan mekanisme perlindungan yang ketat dari kernel pemisahan memastikan bahwa mesin virtual dan aplikasi penyusunnya tidak dapat memengaruhi eksekusi aplikasi penting. Kernel pemisahan adalah satu-satunya perangkat lunak yang berjalan dalam mode prosesor yang paling istimewa. Mekanisme virtualisasi sistem bergantung pada kemampuan perangkat keras spesifik prosesor. Seperti yang dibahas di Bab 2, prosesor tertanam modern semakin banyak yang menggabungkan akselerasi virtualisasi perangkat keras, yang memungkinkan manajemen mesin virtual menjadi sesederhana dan seefisien mungkin.



Gambar 6.16 Virtualisasi MILS linux di SDR.

Kernel pemisahan dapat menyediakan jalur komunikasi antar-proses (IPC) yang dikontrol secara ketat antara HMI Linux dan aplikasi sisi merah lainnya sesuai kebutuhan. Misalnya, data Protokol Internet yang berasal dari subsistem jaringan Linux dapat ditransmisikan melalui pipa IPC ke subsistem kripto untuk enkripsi dan transmisi. Penerapan konsep virtualisasi MILS pada contoh SDR ditunjukkan pada Gambar 6.16.

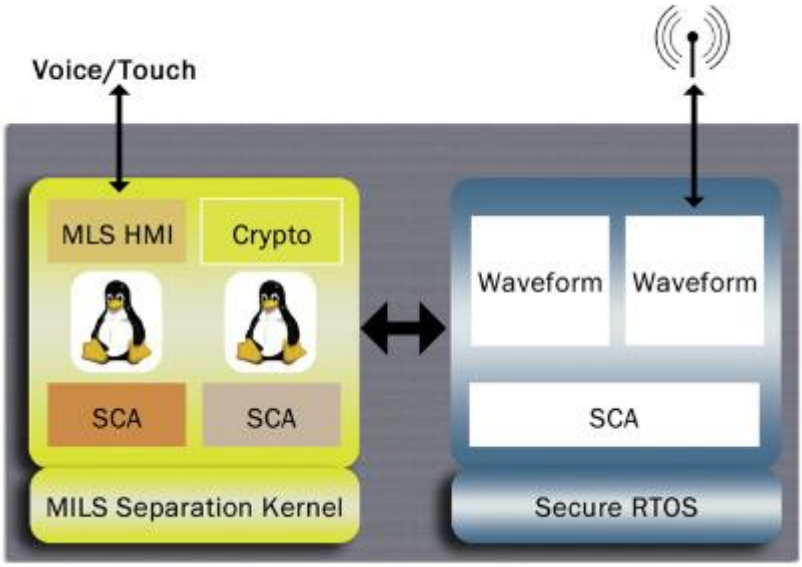
Namun, kesulitan dalam menggunakan Linux untuk mengelola komunikasi rahasia masih ada. Sebagai contoh, mari kita pertimbangkan penggunaan widget layar sentuh Linux untuk memasukkan perintah sensitif “zeroze all cryptographic key now.” Keberhasilan misi mungkin bergantung pada informasi perintah yang diteruskan dengan benar melalui Linux ke subsistem kriptografi. Namun rendahnya jaminan Linux membuat jaminan integritas dan ketersediaan yang diperlukan tidak mungkin dilakukan. Sekali lagi, virtualisasi MILS memberikan solusi. Dalam arsitektur virtualisasi MILS, perangkat grafis fisik dikontrol secara eksklusif oleh aplikasi tepercaya yang berjalan pada kernel pemisahan MILS; Linux hanya memiliki akses ke perangkat virtual. Jadi, ketika sebuah perintah dimasukkan melalui Linux dan melalui antarmuka tervirtualisasi, aplikasi tepercaya dapat memeriksa perintah tersebut. Perintah ini dilakukan hanya jika pengguna puas bahwa Linux telah menyampaikan maksudnya dengan setia. Tahap verifikasi ini menyediakan jalur tepercaya yang diterapkan MILS dari pengguna ke komponen dengan jaminan tinggi; Linux benar-benar keluar dari lingkaran.

6.4.4 Radio Multi-Domain

Beberapa radio yang ditentukan perangkat lunak harus mengelola informasi pada tingkat klasifikasi yang berbeda-beda. Kebijakan standar mengharuskan informasi pada tingkat yang berbeda dijaga agar tetap terisolasi secara fisik. Biasanya, hal ini dicapai dengan salah satu dari dua cara. Salah satu metodenya adalah dengan mewajibkan restart dingin (cold restart), yang mana sumber daya perangkat keras yang dapat ditulis dikosongkan agar dapat mengalihkan perangkat di antara tingkat keamanan dengan aman. Pendekatan ini tidak ramah bagi pengguna, dan reboot dapat menunda komunikasi dan berdampak pada efektivitas misi. Pendekatan kedua adalah dengan menggabungkan prosesor sisi merah yang terpisah untuk setiap tingkat keamanan dan hanya memerlukan peralihan aman pada perangkat antarmuka manusia (misalnya, layar, keyboard). Sekali lagi, elemen pemrosesan tambahan akan meningkatkan ukuran perangkat, biaya produksi, dan kompleksitas sistem. Virtualisasi MILS juga memecahkan masalah multi-domain. Contoh terpisah dari setiap subsistem yang diperlukan, termasuk mesin virtual dan kerangka SCA, dapat diisolasi dan dijadwalkan secara ketat oleh kernel pemisahan MILS. Selain itu, kemampuan kernel pemisahan untuk meng-host aplikasi asli dapat memecahkan masalah perangkat antarmuka manusia bersama: aplikasi HMI multi-level secure (MLS) yang tepercaya dapat berjalan langsung pada kernel pemisahan, multiplexing I/O multi-domain berdasarkan input fokus dari pengguna yang ditangkap langsung oleh komponen MLS. Arsitektur virtualisasi MILS multi-domain untuk SDR ditunjukkan pada Gambar 6.17.

Meskipun arsitektur yang dijelaskan pada Gambar 6.16 dan 6.17 dapat diimplementasikan menggunakan prosesor inti tunggal, munculnya prosesor tertanam multi-inti dapat membuatnya lebih praktis. Seperti dapat dilihat dari contoh-contoh ini, pemrosesan sisi merah yang canggih mencakup banyak komponen, banyak di antaranya dapat dijalankan secara bersamaan pada beberapa inti, di bawah pengawasan kernel pemisahan yang sadar multi-inti. Tenaga ekstra ini memungkinkan kinerja yang baik untuk mesin virtual sekaligus memastikan perilaku real-time untuk aplikasi penting.

Fungsionalitas yang kaya dari paket perangkat lunak multimedia terbaru, seperti Linux, dapat digabungkan bahkan ke dalam perangkat komunikasi militer yang aman dan real-time yang paling menuntut sekalipun tanpa menambah jejak perangkat keras, biaya, atau beban sertifikasi. Inovasi utamanya adalah virtualisasi MILS, yang memanfaatkan kemampuan manajemen sumber daya, lingkungan aplikasi asli, dan silsilah jaminan dari kernel pemisahan tepercaya dan teknik hypervisor modern untuk secara efektif mengkonsolidasikan subsistem tujuan umum dan kritis.



Gambar 6.17 SDR multi-domain dengan mesin virtual linux multi-instance dan komponen MLS dengan jaminan tinggi.

DAFTAR PUSTAKA

Ahmed, Ali. 2014. "Embedded Systems Security Issues and Challenges." Taylor & Francis, London.

Brown, Emma. 2022. "Securing Embedded Systems: Best Practices." Springer, Berlin.

Chan, Michael. 2009. "Embedded System Design for Secure Systems." McGraw-Hill, New York.

Chang, Li. 2002. "Secure Embedded Systems Design." Addison-Wesley, Boston.

Chen, Liang. 2022. "Secure Firmware Development for Embedded Systems." Springer, Berlin.

Chen, Yi-Han. 2015. "Cyber Security for Embedded Systems." Springer, Berlin.

Choi, Eun-Ji. 1999. "Design and Implementation of Secure Embedded Systems." Manning, Shelter Island.

Choi, Young-Jae. 2008. "Practical Security Measures for Embedded Systems." Addison-Wesley, Boston.

Gonzalez, Maria. 2018. "Secure Embedded Systems: Design Principles and Applications." Elsevier, Amsterdam.

Gupta, Rahul. 2016. "Advanced Security Techniques for Embedded Systems." Wiley, New York.

Gupta, Rajesh. 2001. "Fundamentals of Embedded Systems Security." Taylor & Francis, London.

Gupta, Sanjay. 2011. "Embedded Systems Security: Practical Methods for Safe and Secure Software Development." Addison-Wesley, Boston.

Hong, In-Ho. 1998. "Secure Firmware Design for Embedded Systems." Addison-Wesley, Boston.

Hong, Jae-Won. 2004. "Security Challenges in Embedded Systems." CRC Press, Boca Raton.

Jung, Hye-Jin. 2007. "Embedded Systems Security: A Comprehensive Approach." Wiley, New York.

Kim, Dong-Hyun. 2012. "Practical Security for Embedded Systems." Manning, Shelter Island.

Kim, Ji-Won. 1995. "Security Protocols for Embedded Systems." McGraw-Hill, New York.

Kim, Min-Ho. 2011. "Embedded Systems Security: Techniques and Applications." CRC Press, Boca Raton.

Kim, Seung-Hyun. 2020. "Cybersecurity for Embedded Devices." Addison-Wesley, Boston.

Kim, Young-Ho. 2004. "Embedded Security: Issues and Challenges." Springer, Berlin.

Kumar, Rajesh. 2019. "Security Analysis of Embedded Systems." McGraw-Hill, New York.

Kumar, Sanjay. 2015. "Embedded Systems Security: Threats, Vulnerabilities, and Countermeasures." Springer, Berlin.

Lee, Hyun-Joo. 2018. "Embedded Systems Security: Issues and Solutions." Elsevier, Amsterdam.

Lee, Jin-Soo. 1997. "Practical Security Measures for Embedded Systems." Wiley, New York.

Lee, Sang-Min. 2007. "Secure Embedded Systems: A Roadmap." Elsevier, Amsterdam.

Lee, Sun-Ho. 2010. "Secure Design Patterns for Embedded Systems." Taylor & Francis, London.

Li, Wei. 2021. "Cybersecurity for Embedded Systems." CRC Press, Boca Raton.

Lim, Soo-Young. 2005. "Advanced Techniques in Embedded Systems Security." Taylor & Francis, London.

Nguyen, Thanh. 2003. "Principles of Embedded System Security." CRC Press, Boca Raton.

Park, Ji-Hoon. 2005. "Security Engineering for Embedded Systems." Wiley, New York.

Park, Ji-Soo. 2017. "Secure Embedded Software Design." O'Reilly, Sebastopol.

Park, Min-Jae. 2016. "Embedded System Security: A Comprehensive Guide for Engineers and Programmers." Wiley, New York.

Park, Sang-Woo. 2009. "Embedded Systems Security and Privacy." Springer, Berlin.

Park, Soo-Jin. 1996. "Embedded Systems Security Handbook." Springer, Berlin.

Patel, Amit. 2021. "Practical Guide to Embedded System Security." CRC Press, Boca Raton.

Patel, Deepak. 2013. "Cybersecurity for Embedded Systems." Addison-Wesley, Boston.

Patel, Neha. 2008. "Security in Embedded Systems." Wiley, New York.

Patel, Ravi. 2020. "Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development." Addison-Wesley, Boston.

Ryu, Joon-Sik. 2010. "Security in Embedded Devices." Springer, Berlin.

Sharma, Ankit. 2023. "Embedded Systems Security: Threats and Countermeasures." Wiley, New York.

Smith, John. 2023. "Embedded System Security: Concepts and Challenges." Wiley, New York.

Tan, Mei-Ling. 2014. "Secure Embedded Systems Design Principles." Taylor & Francis, London.

Tan, Wei. 2017. "Practical Embedded Security: Building Secure Resource-Constrained Systems." O'Reilly, Sebastopol.

Wang, Peng. 2019. "Security Engineering for Embedded Systems." Taylor & Francis, London.

Wang, Tao. 2013. "Securing Embedded Systems and Software." Artech House, Norwood.

Wang, Xiao-Jun. 2012. "Practical Security for Embedded Systems." Springer, Berlin.

Wang, Xin. 2006. "Embedded Security Systems: A Practical Approach." O'Reilly, Sebastopol.

Yang, Min-Young. 1994. "Introduction to Embedded Systems Security." Elsevier, Amsterdam.

Yoon, Sun-Ho. 2000. "Embedded Systems Security: Techniques and Applications." Artech House, Norwood.

Yoon, Tae-Sung. 2006. "Security Protocols for Embedded Systems." Springer, Berlin.

KEAMANAN SISTEM TERTANAM

(Embedded System Security)

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

BIO DATA PENULIS

Penulis memiliki berbagai disiplin ilmu yang diperoleh dari Universitas Diponegoro (UNDIP) Semarang. dan dari Universitas Kristen Satya Wacana (UKSW) Salatiga. Disiplin ilmu itu antara lain teknik elektro, komputer, manajemen dan ilmu sosiologi. Penulis memiliki pengalaman kerja pada industri elektronik dan sertifikasi keahlian dalam bidang Jaringan Internet, Telekomunikasi, Artificial Intelligence, Internet Of Things (IoT), Augmented Reality (AR), Technopreneurship, Internet Marketing dan bidang pengolahan dan analisa data (komputer statistik).

Penulis adalah pendiri dari Universitas Sains dan Teknologi Komputer (Universitas STEKOM) dan juga seorang dosen yang memiliki Jabatan Fungsional Akademik Lektor Kepala (Associate Professor) yang telah menghasilkan puluhan Buku Ajar ber ISBN, HAKI dari beberapa karya cipta dan Hak Paten pada produk IPTEK. Sejak tahun 2023 penulis tercatat sebagai Dosen luar biasa di Fakultas Ekonomi & Bisnis (FEB) Universitas Diponegoro Semarang. Penulis juga terlibat dalam berbagai organisasi profesi dan industri yang terkait dengan dunia usaha dan industri, khususnya dalam pengembangan sumber daya manusia yang unggul untuk memenuhi kebutuhan dunia kerja secara nyata.

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

ISBN 978-623-8642-18-2 (PDF)
9 786238 642182