Kuliah 32: Kerentanan Keamanan Perangkat Seluler

Catatan Kuliah tentang "Keamanan Komputer dan Jaringan"

oleh Avi Kak (kak@purdue.edu)

7 Mei 2020
11:08

Tujuan:

- Apa yang membuat perangkat seluler kurang rentan terhadap malware - sejauh itu masalahnya
- Perlindungan diberikan dengan melakukan sandbox pada aplikasi
- Keamanan (atau ketiadaan) disediakan oleh enkripsi over-the-air untuk komunikasi seluler dengan implementasi Python dari cipher A5 / 1
- Serangan saluran samping pada perangkat seluler khusus
- Contoh serangan saluran samping: serangan injeksi kesalahan dan serangan waktu
- Skrip Python untuk mendemonstrasikan injeksi kesalahan dan serangan waktu
- Perangkat USB sebagai sumber malware yang mematikan
- IP Seluler

ISI

Kembali ke Daftar Isi

32.1 PERANGKAT JAHAT DAN PERANGKAT SELULER

- Perangkat seluler - ponsel, ponsel cerdas, kartu pintar, tablet, perangkat navigasi, memory stick, dll., - kini telah merasuki hampir semua aspek kehidupan kita sehari-hari. Dulu fungsi utama mereka hanyalah komunikasi, sekarang mereka digunakan untuk hampir semua hal: sebagai kamera, sebagai pemutar musik, sebagai pembaca berita, untuk memeriksa email, untuk web surfing, untuk navigasi, untuk perbankan, untuk berhubungan dengan teman melalui media sosial, dan, Ah !, jangan dilupakan, sebagai boarding pass saat bepergian lewat udara.

- Keputusan dengan suara bulat oleh Mahkamah Agung Amerika Serikat belum lama ini menunjukkan betapa perangkat tersebut telah menjadi integral dan sentral dalam kehidupan kita. Dalam keputusan 9-0 pada 25 Juni, 2014, hakim memutuskan bahwa polisi tidak boleh menggeledah ponsel tersangka tanpa surat perintah. Biasanya, polisi diizinkan untuk menggeledah barang-barang pribadi Anda - seperti dompet, tas kerja, kendaraan, dll. - tanpa surat perintah jika ada "kemungkinan penyebab" bahwa kejahatan telah dilakukan. Mengenai ponsel, Ketua Mahkamah Agung John Roberts berkata: “Mereka begitu luas dan bagian penting dari kehidupan sehari-hari bahwa pepatah pengunjung dari Mars mungkin menyimpulkan bahwa mereka adalah fitur penting manusia ilmu urai." Justice Roberts juga mengamati: "Ponsel modern,

sebagai sebuah kategori, implikasi masalah privasi jauh melampaui yang terkait dengan pencarian bungkus rokok, dompet, atau tas. Ponsel berbeda dalam arti kuantitatif dan kualitatif dari objek lain yang mungkin disimpan pada orang yang ditangkap. "

- Para hakim jelas mendasarkan keputusan mereka pada fakta bahwa orang sekarang secara rutin menyimpan informasi pribadi dan sensitif di perangkat seluler mereka - jenis informasi yang akan Anda simpan dengan aman di rumah di tahun-tahun yang lalu.

- Mengingat kenyataan modern ini, tidak mengherankan bahwa orang-orang yang terlibat dalam produksi dan penyebaran malware semakin melatih senjata mereka di perangkat seluler.

- Dalam laporan tentang keamanan perangkat seluler yang diserahkan ke Kongres, Kantor Akuntabilitas Pemerintah Amerika Serikat (GAO) menyatakan bahwa jumlah varian malware yang berbeda yang ditujukan untuk ponsel cerdas telah meningkat dari 14.000 menjadi 40.000 hanya dalam satu tahun (dari Juli 2011 hingga Mei 2012). Anda dapat mengakses ini lapor di http://www.gao.gov/assets/650/648519.pdf [Laporan yang sama juga menyebutkan bahwa penjualan perangkat seluler di seluruh dunia meningkat dari 300 juta menjadi 650 juta pada tahun 2012. Sekarang pertimbangkan fakta bahwa lebih dari 1,5 miliar smartphone (dengan total nilai penjualan mendekati $ 500 miliar) dijual pada 2017. Dan itu hanya smartphone di antara semua jenis perangkat seluler. Itu nomor untuk 2017 berasal dari https://www.statista.com/topics/840/smartphones/]

- Perangkat seluler telah menjadi magnet bagi produsen malware

karena mereka dapat menjadi sumber informasi sensitif yang mungkin dapat digunakan oleh penyerang untuk keuntungan moneter, untuk mencari keuntungan politik, digunakan sebagai sarana untuk masuk ke jaringan perusahaan, dan sebagainya.

- Seperti yang Anda perkirakan, banyak metode serangan di perangkat seluler sama dengan yang ada di perangkat komputasi tradisional seperti desktop, laptop, dll., - kecuali untuk satu perbedaan yang sangat penting: Kecuali dalam jaringan pribadi, a non-seluler host biasanya langsung dicolokkan ke internet di mana ia terus-menerus terkena upaya pembobolan melalui perangkat lunak yang memindai segmen besar blok alamat IP untuk menemukan host yang rentan. Artinya, selain menghadapi serangan yang ditargetkan melalui rekayasa sosial dan cara lain, host non-seluler yang terhubung ke internet juga menghadapi serangan yang tidak ditargetkan oleh penjahat dunia maya yang hanya ingin menemukan host (di mana pun mereka berada) tempat mereka berada. dapat memasang perangkat lunak perusak mereka.

- Di samping itu , Perangkat seluler ketika dicolokkan ke jaringan seluler hanya dapat diakses oleh pihak luar melalui gateway yang dikontrol ketat oleh perusahaan telepon seluler.

  [Pertimbangkan situasi kebalikan dari perangkat seluler yang dapat mengakses internet secara langsung melalui, katakanlah, jaringan WiFi. Saat menggunakan WiFi, perangkat seluler akan berada dalam jaringan pribadi (biasanya jaringan pribadi kelas A atau kelas C) di belakang router / titik akses nirkabel. Jadi perangkat seluler tidak akan langsung diekspos ke pemindaian blok alamat IP. Namun, sekarang, perangkat seluler bisa jadi rentan terhadapnya

penyadapan dan serangan man-in-the-middle jika, katakanlah, Anda bertukar informasi sensitif dengan host jarak jauh dalam teks biasa. Namun, dalam mode yang paling umum menggunakan ponsel cerdas, Anda tidak mungkin menjadi target serangan semacam itu karena keamanan keseluruhan yang disediakan oleh server. Misalnya, ponsel cerdas Anda akan membuat tautan aman dengan situs web seperti Amazon.com sebelum mengunggah informasi kartu kredit Anda ke situs web itu. Seperti yang Anda ketahui dari Kuliah 13, ponsel cerdas Anda akan mencapainya dengan mengunduh
Amazon.com sertifikat, memverifikasi sertifikat dengan kunci publik dari root CA yang berlaku yang telah disimpan di ponsel cerdas Anda, dan ponsel cerdas serta situs web jarak jauh Anda kemudian akan bersama-sama membuat kunci sesi untuk enkripsi konten.]

- Oleh karena itu, kecil kemungkinannya perangkat seluler yang Anda miliki akan terkena perangkat lunak penyerang acak.

- Karena perlindungan yang diberikan oleh (1) gateway perusahaan seluler; (2) perlindungan yang dimungkinkan oleh koneksi terenkripsi dengan server yang mencari informasi pribadi Anda; (3) perlindungan yang diberikan oleh toko aplikasi online (seperti Google Play dan Apple's App Store) melalui pemeriksaan aplikasi mereka untuk lubang keamanan sebelum membuatnya tersedia untuk Anda ; dan, akhirnya, (4) perlindungan yang diberikan oleh fakta bahwa OS seluler cenderung menjalankan aplikasi di kotak pasir ; tidak mengherankan jika tingkat infeksi malware di ponsel cerdas serendah yang disebutkan di bagian selanjutnya.

- Namun, perangkat seluler juga rentan terhadap sosial

rekayasa serangan sebagai perangkat komputasi yang lebih tradisional seperti desktop dan laptop. (Lihat Kuliah 30 untuk Sosial Serangan teknik.) Tentu saja, tidak perlu dikatakan lagi bahwa jika perangkat seluler berisi perangkat lunak yang belum ditambal dengan kerentanan yang diketahui, perangkat tersebut dapat dieksploitasi melalui serangan jaringan biasa yang tidak bergantung pada manipulasi psikologis.

- Selain itu, kelas tertentu dari perangkat seluler yang lebih terspesialisasi - kartu pintar khususnya - mungkin rentan terhadap serangan yang termasuk dalam kategori serangan saluran samping . [Kartu pintar telah ada di mana-mana. Mereka sekarang digunakan untuk membayar ongkos di sistem transportasi umum, pencurian mobil perlindungan (kunci mobil elektronik Anda), kontrol akses di gedung, dll.] Serangan ini paling efektif jika musuh dapat mengambil kendali fisik atas perangkat seluler dan menundukkannya pada pemeriksaan yang memperlakukannya sebagai kotak blok dan menerapkan berbagai jenis masukan padanya, atau, jika memungkinkan, memeriksanya secara langsung di tingkat perangkat keras / sirkuit. Karsten Nohl memberikan ceramah Black Hat pada tahun 2008 yang menunjukkan bagaimana dia dapat memecahkan enkripsi di kartu pintar Mifare langsung dari silikon. [Kalimat terkenal dari pembicaraan itu: “Tidak ada rahasia di dalamnya silikon] Lihat di:

https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Nohl/BlackHat-Japan-08-Nohl-Secret-Algorithms-in-Hardware.pdf

- Dalam sisa kuliah ini, saya pertama-tama akan meninjau konsep sandboxing aplikasi karena hal itu menambah secara signifikan perlindungan perangkat seluler terhadap aplikasi berbahaya.

- Selanjutnya, saya akan meninjau algoritma A5 / 1 yang telah banyak digunakan di seluruh dunia untuk mengenkripsi data suara dan SMS melalui udara di jaringan telepon seluler GSM (2G). Algoritma ini salah satu studi kasus terbaik tentang apa yang dapat terjadi ketika orang memutuskan untuk menciptakan keamanan dengan ketidakjelasan. Algoritma ini dirahasiakan selama beberapa tahun oleh operator telepon seluler. Seperti yang hampir selalu terjadi dengan hal-hal seperti itu, akhirnya algoritme itu bocor. Segera setelah algoritme masuk ke domain publik, algoritme tersebut terbukti tidak memiliki keamanan.

- Kemudian saya akan mempresentasikan apa yang dimaksud dengan serangan saluran samping . Seperti disebutkan sebelumnya di bagian ini, perangkat seluler khusus seperti kartu pintar sangat rentan terhadap serangan ini. Untuk memberikan kejelasan lebih lanjut tentang bagaimana seseorang dapat membangun serangan seperti itu, saya akan memberikan implementasi Python saya untuk beberapa bentuk serangan yang lebih umum.

- Terakhir, saya akan membahas topik yang banyak menjadi berita akhir-akhir ini: kemudahan penyebaran infeksi malware dengan perangkat USB seperti stik memori dan mengapa infeksi semacam itu tidak dapat dideteksi oleh alat anti-virus yang umum.

Kembali ke Daftar Isi

32.2 KABAR BAIK ADALAH ...

- Seperti yang disebutkan di akhir bagian sebelumnya, perangkat seluler - terutama jenis smartphone - manfaat dari perlindungan berlapis. Ini adalah:

  - Sebagian besar, smartphone individu hanya dapat diakses melalui gateway yang dikendalikan oleh perusahaan jaringan seluler;

  - Saat terlibat dalam interaksi e-niaga dan terlepas dari apakah ponsel cerdas berkomunikasi langsung melalui jaringan seluler atau melalui WiFi, fakta bahwa ponsel cerdas dan server membuat sesi terenkripsi sebelum informasi sensitif apa pun dipertukarkan di antara keduanya (sesuai dengan klien interaksi server yang dijelaskan di Kuliah 13);

  - Toko aplikasi (Google Play, Apple's App Store, Windows Phone Store) memindai dan menganalisis aplikasi untuk malware sebelum membuatnya tersedia untuk pelanggan;

  - Fakta bahwa aplikasi biasanya dijalankan oleh OS seluler di kotak pasir. Hal ini tentunya berlaku untuk OS Android untuk perangkat Android; iOS untuk semua perangkat seluler oleh Apple dan yang mencakup berbagai versi iPhone, iPod, dan iPad; dan OS Windows Phone untuk perangkat seluler berbasis Windows.

- Terlepas dari lapisan perlindungan ini, keamanan ponsel cerdas dapat dengan mudah dikompromikan oleh: (1) serangan man-in-the-middle saat perangkat dicolokkan ke jaringan WiFi yang tidak terkunci (terutama jika pengguna mengirim atau menerima informasi sensitif di teks biasa); dan (2) pengguna mengunjungi situs web yang menipu atau memikat pengguna untuk mengunduh dokumen yang merupakan perangkat lunak jahat atau berisi perangkat lunak jahat. Namun, bentuk kerentanan ini juga berlaku untuk perangkat komputasi non-seluler seperti desktop dan laptop.

- Meskipun demikian, tetap saja empat lapisan keamanan yang disebutkan di halaman sebelumnya memperkecil kemungkinan ponsel cerdas Anda berisi malware. Kesimpulan ini didukung oleh laporan "Android Security & Privacy, 2018 Year in Review" yang baru saja dirilis oleh Google yang akan Anda temukan di URL berikut:

  https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf

- Laporan Keamanan & Privasi Android terbaru ini mengatakan:

  > “Pada 2018, hanya 0,08% perangkat yang menggunakan Google Play secara eksklusif untuk mengunduh aplikasi terpengaruh oleh PHA. Sebaliknya, perangkat yang memasang aplikasi dari luar Google Play terpengaruh oleh PHA delapan kali lebih sering. Dibandingkan dengan tahun sebelumnya, bahkan perangkat tersebut mengalami penurunan malware sebesar 15% karena kewaspadaan Google Play Protect. ”

  di mana "PHA" adalah singkatan dari "Aplikasi yang Berpotensi Berbahaya".

  [Selain menyenangkan, versi tahun lalu dari laporan yang sama menyebutkan

berikut: “Pada 2017, mengunduh PHA dari Google Play lebih kecil kemungkinannya daripada kemungkinan asteroid menghantam bumi.” Apakah kita berasumsi bahwa perangkat Android memiliki peluang untuk mengunduh malware? Atau, mungkinkah Google memiliki perkiraan yang lebih baik sekarang untuk kemungkinan asteroid menghantam bumi?]

- Meskipun nyaman mengetahui bahwa secara keseluruhan kemungkinan perangkat Android Anda terinfeksi malware rendah, masih akan sangat mendidik bagi pembaca untuk membaca empat halaman terakhir dari laporan keamanan Android dan membaca tentang berbagai kelompok malware yang saat ini sedang berputar-putar di ekosistem Android.

- Salah satu contoh jahat dari malware ini yang banyak terdapat di dalamnya berita di tahun 2017 memata-matai pengguna Android melalui aplikasi jinak yang digunakan mungkin sah SDK jaringan iklan untuk mengeluarkan iklan. Seperti yang sudah Anda ketahui, SDK adalah singkatan dari "Software Development Kit" dan SDK jaringan iklan adalah paket perangkat lunak, yang disediakan oleh organisasi jaringan iklan, yang dimasukkan oleh pengembang aplikasi ke dalam aplikasi mereka. Selanjutnya, jaringan iklan yang dimaksud dapat memutuskan iklan apa yang akan ditampilkan kepada pengguna perangkat Android dan melacak keefektifan iklan tersebut dalam menghasilkan pendapatan yang dibagi oleh organisasi jaringan iklan dan pengiklan.

- Lookout, sebuah perusahaan keamanan (https://www.lookout.com) mengkhususkan diri dalam

masalah terkait perangkat seluler, menemukan bahwa SDK jaringan iklan yang disediakan oleh perusahaan jaringan iklan bernama Igexin mengunduh plugin berbahaya yang dienkripsi ke perangkat Android dari server Igexin dan mengunggah ke server data pribadi pengguna. Yang terpenting di sini adalah saat aplikasi ini dikirimkan ke Google Play, tidak ada petunjuk malware di dalamnya. Agaknya, pengembang aplikasi sendiri sama sekali tidak mengetahui tentang bagaimana aplikasi mereka akan disalahgunakan oleh organisasi jaringan iklan setelah aplikasi tersebut diterima oleh Google Play.

- Dalam laporannya, Lookout menyebutkan bahwa aplikasi yang berisi SDK yang terpengaruh yang dijelaskan di atas diunduh lebih dari 100 juta kali di seluruh ekosistem Android.

- Seperti yang disebutkan sebelumnya, empat halaman terakhir dari laporan Keamanan & Privasi Google Android 2018 menyediakan daftar berbagai jenis malware terkait Android yang menarik saat ini. Dari jenis yang berbeda ini, inilah yang dikatakan dalam laporan tentang jenis yang disebut "Chamois":

  > “Chamois adalah salah satu keluarga PHA paling berpengaruh di Android pada tahun 2018 dengan lebih dari 199 juta pemasangan. ... Chamois adalah malware yang direkayasa dengan baik dan canggih. Sampai November 2018, ada lima varian Chamois yang diketahui botnet keluarga, tiga di antaranya muncul setelah November 2017. Varian ini terdiri dari empat atau lima tahap dengan fitur anti-analisis dan infrastruktur perintah dan kontrol untuk menerapkan muatannya. Klasifikasi Google Play Protect

> Chamois sebagai pintu belakang karena kemampuan perintah dan kendali jarak jauh yang dimilikinya. Muatan untuk Chamois berkisar dari berbagai muatan penipuan iklan hingga penipuan SMS hingga pemuatan kode dinamis. ”

dan tentang jenis yang disebut "Snowfox":

> “Snowfox adalah SDK periklanan dengan dua varian; satu varian mencuri token OAuth dari perangkat dan varian lainnya memasukkan JavaScript untuk penipuan klik ke WebView dengan iklan yang dimuat. Kampanye Snowfox dimulai pada akhir 2017 dan mencapai puncaknya pada Maret 2018. Selama tahun 2018, aplikasi dengan Snowfox SDK yang berbahaya telah diinstal lebih dari 16 juta kali .
> Snowfox sebagian besar didistribusikan di luar Google Play oleh aplikasi termasuk SDK. Namun ada beberapa mekanisme distribusi tempat aplikasi secara dinamis mendownload Snowfox SDK sebagai plugin, mungkin untuk melewati analisis statis. ”

Kembali ke Daftar Isi

32.3 MEMASANG APLIKASI

- Sebagian besar keamanan yang Anda dapatkan dengan perangkat seluler seperti ponsel cerdas disebabkan oleh fakta bahwa perangkat lunak pihak ketiga (aplikasi) dijalankan di kotak pasir. Ini berlaku untuk semua sistem operasi seluler utama saat ini, seperti OS Android, iOS, OS Windows Phone, dll.

- Secara umum, setiap aplikasi dijalankan sebagai proses terpisah di kotak pasirnya sendiri.

- Sandboxing berarti mengisolasi proses aplikasi dari satu sama lain, di satu sisi, dan dari sumber daya sistem, di sisi lain. Sandboxing juga memerlukan penempatan kerangka izin yang secara ketat mengatur aplikasi lain mana yang mendapatkan akses ke data yang dihasilkan oleh aplikasi tertentu. [Sandboxing sekarang juga banyak digunakan untuk aplikasi desktop / laptop. Browser web di desktop / laptop Anda kemungkinan besar adalah dijalankan di kotak pasir. Dengan begitu, data apa pun yang diunduh / dibuat dengan mengatakan plugin seperti Adobe Flash atau Microsoft Silverlight tidak mungkin merusak file Anda yang lain meskipun data yang diunduh berisi perangkat lunak jahat. Sandbox sekarang juga digunakan oleh pembaca dokumen untuk PDF dan format lain sehingga malware masuk ke makro dokumen tersebut tidak merusak file lainnya.]

- Sisa dari bagian ini berfokus terutama pada bagaimana Android

mengisolasi proses dengan menjalankannya di kotak pasir. Namun, sebelum berbicara tentang sandboxing di Android, mari kita segera meninjau beberapa hal penting dari OS Android karena OSlah yang menuntut agar setiap aplikasi berjalan di kotak pasirnya sendiri.

- Saya kira Anda sudah tahu bahwa Android lahir dari Linux. Rilis pertama Android didasarkan pada kernel Linux Versi 2.6.25. Versi Android yang lebih baru didasarkan pada kernel Linux Versi 3.4, 3.8, dan 3.10, bergantung pada platform perangkat keras yang tepat. (Pada pertengahan April 2018, versi stabil terbaru dari kernel Linux adalah 4.16.2 menurut informasi yang diposting di http://www.kernel.org situs web.) [Jika Anda menggunakan pengetahuan Anda tentang Linux sebagai batu loncatan untuk mempelajari Android, tempat yang baik untuk dikunjungi untuk mempelajarinya fitur-fitur yang unik untuk Android adalah http://elinux.org/Android_Kernel_Features. Untuk meringkas beberapa perbedaan utama: (1) Satu perbedaan yang signifikan berkaitan dengan bagaimana komunikasi antarproses itu dilakukan di Android. (2) Primitif manajemen daya Android yang dikenal sebagai "wakelock" yang bisa digunakan aplikasi digunakan untuk menjaga agar CPU tetap bersenandung pada frekuensi normalnya dan layar tetap menyala bahkan saat pengguna tidak ada interaksi. Mode normal operasi ponsel cerdas mengharuskan ponsel masuk ke mode tidur nyenyak (dengan memutar o layar dan mengurangi frekuensi CPU untuk menghemat daya) ketika tidak ada pengguna interaksi. Namun, itu tidak berfungsi untuk, katakanlah, aplikasi Facebook yang mungkin perlu memeriksa setiap acara beberapa menit. Aplikasi semacam itu dapat memperoleh wakelock untuk menjaga CPU tetap berjalan pada frekuensi normalnya dan, jika perlu, untuk menghidupkan layar saat peristiwa baru yang menarik bagi pemilik ponsel cerdas terdeteksi. (Karena kebutuhan aplikasi Facebook untuk memperoleh wakelock setiap beberapa menit dapat menguras baterai Anda, beberapa pengguna Android memasang aplikasi root gratis bernama Greenify untuk pertama-tama mengetahui berapa banyak baterai yang digunakan dikonsumsi oleh aplikasi semacam itu dan untuk lebih mengontrol kebutuhannya untuk sering bangun.) (3) Memori Android fungsi alokasi. (4) Dan seterusnya. Selain perbedaan antara Linux dan Android ini, perhatikan juga bahwa OS Android harus bekerja dengan beberapa jenis sensor dan komponen perangkat keras yang berbeda a

OS desktop / laptop tidak perlu repot dengan. Kita berbicara tentang sensor dan komponen perangkat keras seperti layar sentuh, kamera, komponen audio, sensor orientasi, akselerometer, giroskop, dll. Terakhir, perhatikan bahwa Android pada awalnya dikembangkan untuk arsitektur ARM. Namun, sekarang juga didukung untuk x86 dan arsitektur prosesor MIPS.] Meskipun ada perbedaan antara Linux dan Android, Linux Foundation dan banyak lainnya menganggap Android sebagai distribusi Linux (meskipun tidak disertai dengan pustaka Gnu C, dll. Android dilengkapi dengan pustaka C sendiri yang memiliki footprint memori yang lebih kecil; disebut Bionic) .

- Seperti yang telah disebutkan, setiap aplikasi Android - yang ditulis di Java - dijalankan di kotak pasir sebagai proses terpisah. [Lebih tepatnya berbicara, terpisah proses dibuat untuk ID pengguna Linux yang ditandatangani secara digital. Jika ada beberapa aplikasi yang terkait ID pengguna Linux yang sama, semuanya dapat dijalankan dalam proses Linux yang sama. Berikut adalah tutorial yang bagus tentang bagaimana Anda pergi tentang membuat kunci publik dan pribadi untuk menandatangani aplikasi Android yang telah Anda buat secara digital: http://www.ibm.com/developerworks/library/x-androidsecurity/] Saat Anda mengunduh aplikasi baru atau perbarui salah satu aplikasi yang sudah ada di perangkat Anda, fitur sandboxing inilah yang menyebabkan ponsel cerdas Anda menanyakan apakah aplikasi diizinkan mengakses data yang dihasilkan oleh program lain dan berbagai komponen ponsel cerdas Anda - ini adalah informasi lokasi, kamera, log, bookmark, dll.

- Secara default, aplikasi berjalan tanpa izin yang ditetapkan untuknya. Saat sebuah aplikasi meminta akses ke data yang dihasilkan oleh aplikasi lain, itu tunduk pada aturan yang dideklarasikan dalam file manifes yang terakhir.

- Sandboxing memastikan bahwa, secara umum, file apa pun yang dibuat oleh aplikasi hanya dapat dibaca oleh aplikasi itu. Android memang memberi fasilitas pengembang aplikasi untuk membuat lebih banyak file yang dapat diakses secara global melalui mode bernama MODE DUNIA TERTULIS dan MODE DUNIA DAPAT DIBACA. Aplikasi yang menggunakan mode baca / tulis ini tunduk pada pengawasan yang lebih cermat dari sudut pandang keamanan.

- Untuk kontrol yang lebih besar atas proses aplikasi lain yang dapat mengakses data yang dibuat oleh aplikasi Anda sendiri, daripada menggunakan dua mode baca / tulis yang disebutkan di poin sebelumnya, aplikasi Anda dapat menempatkan datanya dalam objek yang merupakan subclass dari kelas Java Android Penyedia konten dan tentukan android: diekspor, android: protectionLevel, dan lainnya atribut. [Dalam banyak kasus, a Penyedia konten menyimpan informasinya dalam database SQlite, yaitu namanya adalah database SQL untuk menyimpan informasi terstruktur. Sebuah aplikasi meminta informasi dari database seperti itu pertama-tama harus membuat klien dengan membuat subclass dari kelas Java ContentResolver.]

- Dalam sistem Linux, dua teknik yang paling banyak digunakan untuk proses sandboxing adalah SELinux dan AppArmor. SELinux - nama kependekan dari "Security Enhanced Linux" - adalah modul kernel Linux yang memungkinkan sistem operasi untuk menjalankan kontrol akses mendetail terkait dengan permintaan sumber daya dengan menjalankan program.

- Baik SELinux dan AppArmor didasarkan pada LSM (Linux Security Modules) API untuk menjalankan apa yang dikenal sebagai

Kontrol Akses Wajib (MAC) . MAC dimaksudkan secara khusus untuk sistem operasi untuk menempatkan batasan pada sumber daya yang dapat diakses dengan menjalankan program. Yang kami maksud dengan sumber daya adalah file, direktori, port, antarmuka komunikasi, dll.

- Mungkin perbedaan yang paling signifikan antara SELinux dan AppArmor adalah bahwa yang pertama didasarkan pada label konteks yang terkait dengan semua file, antarmuka, sumber daya sistem, dll., dan yang terakhir didasarkan pada nama jalur yang sama. [Oleh default, Ubuntu menginstal Linux dengan AppArmor. Namun, Anda sendiri dapat menginstal patch kernel SELinux melalui Synaptic Package Manager. Perlu diingat, saat Anda menginstal SELinux, AppArmor paket akan dihapus secara otomatis. Tentang membandingkan SELinux dengan AppArmor, ada banyak hal pengembang di luar sana yang lebih memilih yang terakhir karena mereka mempertimbangkan aturan kebijakan SELinux untuk mengisolasi file proses menjadi terlalu kompleks untuk pembuatan manual. Meskipun ada alat yang dapat menghasilkan file aturan untuk Anda, kompleksitas aturan membuatnya sulit untuk diverifikasi, menurut para pengembang ini. Di di sisi lain, aturan AppArmor relatif sederhana, dapat diekspresikan secara manual, dan oleh karena itu lebih dari itu setuju dengan validasi manusia. Namun, kontrol akses yang dapat Anda capai dengan AppArmor tidak sebaik itu berbutir seperti apa yang bisa Anda dapatkan dengan SELinux.] Tiga sumber berikut memberikan a penilaian komparatif SELinux dan AppArmor untuk mengisolasi proses yang berjalan di komputer Anda:

  http://elinux.org/images/3/39/SecureOS_nakamura.pdf

  http://researchrepository.murdoch.edu.au/6177/1/empowering_end_users.pdf

  https://www.suse.com/support/security/apparmor/features/selinux_comparison.html

- Kontrol Akses Wajib (MAC) yang digunakan oleh Android untuk mengisolasi proses dengan menjalankannya di sandbox didasarkan pada

SELinux. [Ini berlaku untuk Android versi 4.3 dan yang lebih tinggi. Saya percaya yang terbaru versi Android adalah 8.1 Oreo.] Oleh karena itu, sisa dari bagian ini berfokus pada SELinux.

- Titik awal yang baik untuk memahami kontrol akses yang dimungkinkan oleh SELinux adalah apa yang Anda dapatkan dari distribusi Linux standar. Distribusi standar mengatur akses berdasarkan hak istimewa yang terkait dengan program yang sedang berjalan. Secara umum, jika program berjalan dengan hak superuser (yaitu, hak istimewa yang terkait dengan ID pengguna 0), program dapat melewati semua batasan keamanan. Artinya, program seperti itu tidak memiliki batasan mengenai file, antarmuka, komunikasi antarproses, dan sebagainya, yang dapat diaksesnya. (Bayangkan saja program jahat di mesin Anda yang telah berhasil menebak kata sandi root Anda.) [Dalam hal Anda kebetulan memikirkan hak akses di platform Windows, SISTEM akun, dan ADMINISTRATOR memiliki hak yang mirip dengan hak akses root pada sistem Unix / Linux.] Akses kontrol dalam distribusi standar Linux disebut sebagai Linux Discretionary Access Control (DAC).

- SELinux, di sisi lain, mengaitkan a label konteks dengan setiap file, direktori, akun pengguna, proses, dll., di komputer Anda. Label konteks terdiri dari empat bagian yang dipisahkan titik dua (dengan bagian terakhir opsional):

  pengguna: wewenang : Tipe : tingkat

  Anda dapat melihat label konteks yang terkait dengan file atau direktori dengan menjalankan perintah ' ls -Z nama file '. Untuk

Misalnya, ketika saya menjalankan perintah ' ls -Z / rumah / kak / ', berikut beberapa baris yang saya dapatkan:

```
system_u: object_r: file_t: s0      AdaBoost /
system_u: object_r: file_t: s0      admin /
system_u: object_r: file_t: s0      analitik /
system_u: object_r: file_t: s0      ArgoUML /
system_u: object_r: file_t: s0      av /
system_u: object_r: file_t: s0      cadangan /
system_u: object_r: file_t: s0      beamer /
. . .
. . .
```

Apa yang Anda lihat di kolom pertama adalah label konteks yang dibuat oleh SELinux untuk subdirektori di direktori home saya. Oleh karena itu, untuk subdirektori AdaBoost, 'pengguna' adalah sistem kamu, _ peran' objek t, tipe' file t, dan 'level' s0. SELinux menggunakan potongan informasi individual ini untuk membuat keputusan kontrol akses. Jika kebijakan keamanan mengizinkannya, Anda dapat mengubah komponen label konteks secara selektif dengan menggunakan chcon perintah. Perintah itu adalah singkatan dari "ubah konteks".

- Dan jika Anda ingin melihat label konteks yang terkait dengan proses yang sedang berjalan di komputer Anda, jalankan perintah ' ps -eZ '. Ketika saya menjalankan perintah ini di laptop Ubuntu saya, saya mendapatkan daftar yang sangat panjang, di antaranya hanya beberapa dari entri awalnya adalah:

```
system_u: system_r: kernel_t: s0      1?      00:00:02 init
system_u: system_r: kernel_t: s0      2?      00:00:00 kthreadd
system_u: system_r: kernel_t: s0      3?      00:00:01 ksoftirqd / 0
```

```
system_u: system_r: kernel_t: s0        5?        00:00:00 kworker / 0: 0H
system_u: system_r: kernel_t: s0        7?        00:10:26 rcu_sched
system_u: system_r: kernel_t: s0        8?        00:05:49 rcuos / 0
system_u: system_r: kernel_t: s0        9?        00:03:22 rcuos / 1
. . .
. . .
. . .
```

- Seperti yang dapat Anda bayangkan, saat Anda mengaitkan dengan setiap entitas di komputer Anda label konteks dari jenis yang ditunjukkan di atas, Anda dapat menyiapkan kebijakan kontrol akses yang terperinci dengan menempatkan batasan pada sumber daya mana yang digunakan pengguna (dalam arti yang digunakan dalam konteks label) dalam peran tertentu dan jenis serta tingkat tertentu diizinkan untuk diakses dengan mempertimbangkan label konteks sumber daya itu sendiri. Anda sekarang dapat membuat kebijakan Kontrol Akses Berbasis Peran (RBAC), atau kebijakan Penegakan Jenis (TE), dan, jika SELinux menentukan kolom 'level' opsional dalam label konteks, kebijakan Keamanan Multi Level (MLS). Selain itu, Anda dapat menyiapkan kebijakan Keamanan Multi-Kategori (MCS) - kita akan membicarakannya nanti.

- Untuk menunjukkan contoh sederhana penerapan tipe dari SELinux FAQ, asumsikan bahwa semua file di akun pengguna diberi label tipe pengguna rumah t. Dan asumsikan bahwa browser Firefox yang berjalan di komputer Anda diberi label tipe firefox t. Deklarasi kontrol akses berikut

```
izinkan firefox_t user_home_t: file {read write};
```

kemudian akan memastikan bahwa browser hanya membaca dan menulis

izin sehubungan dengan file pengguna - meskipun browser dijalankan oleh seseorang dengan hak akses root . [Anda dapat melihat mengapa beberapa orang pikirkan SELinux sebagai pertarungan antar program. Biasanya, seperti yang Anda lihat di Kuliah 18, sebuah firewall mengatur lalu lintas antara komputer dan seluruh jaringan.]

- Untuk mempermudah pembuatan kebijakan kontrol akses untuk aplikasi baru, SELinux memberi Anda Kebijakan Referensi yang dapat dimodifikasi sesuai kebutuhan. Perusahaan bernama Tresys Technologies memperbarui Kebijakan Referensi berdasarkan umpan balik pengguna yang dikirim ke milis Project Kebijakan di GitHub. Kebijakan referensi ini biasanya disesuaikan oleh penyedia platform Linux Anda.

- Untuk menjadi lebih akrab dengan SELinux, Anda dapat mengunduh dan menginstal SELinux di mesin Ubuntu melalui Synaptic Package Manager Anda. [Atau Anda bisa melakukan ' sudo apt-get remove apparmor ' diikuti oleh ' sudo apt-get install selinux '] Pastikan Anda memilih meta-paket selinux dan bukan paketnya selinux-basics. SELinux menjadi operasional (meskipun tidak diaktifkan) hanya dengan menginstalnya. Perhatikan bahwa dengan Ubuntu, referensi kebijakan Anda get disimpan dalam file / etc / selinux / ubuntu / policy / policy29.

- Setelah Anda menginstal SELinux seperti yang dijelaskan di atas, Anda perlu mem-boot ulang mesin untuk mengaktifkan SELinux. [Selama reboot ini, semua file pada disk akan mendapatkan label konteks sesuai dengan penjelasannya disajikan sebelumnya di bagian ini.] Selanjutnya, jika Anda menjalankan perintah

Suka ' sestatus '(Anda tidak harus menjadi root untuk menjalankan perintah ini), Anda akan melihat output berikut yang dikembalikan oleh SELinux:

Status SELinux:	diaktifkan
Dudukan SELinuxfs:	/ sys / fs / selinux
Direktori root SELinux:	/ etc / selinux
Nama kebijakan yang dimuat:	ubuntu
Mode saat ini:	permisif
Mode dari file konfigurasi:	permisif
Status MLS kebijakan:	diaktifkan
Status deny_unknown kebijakan:	diizinkan
Versi kebijakan kernel maks:	28

Jika sekarang Anda menjalankan perintah ' sudo setenforce 1 ', Anda akan melihat hasil yang sama seperti yang ditunjukkan di atas, tetapi dengan garis' Arus mode: permisif 'berubah menjadi' Mode saat ini: menegakkan '.

- Jika Anda ingin melihat daftar semua pengguna SELinux, Anda dapat memasukkan perintah ' seinfo -u '. Ketika saya menjalankan perintah ini di laptop Ubuntu saya, saya mengerti

  Pengguna: 6
  - sysadm_u
  - system_u
  - akar
  - staff_u
  - user_u
  - unconfined_u

  Dan jika saya menjalankan perintah ' seinfo -r 'untuk melihat semua peran yang digunakan dalam label konteks, saya kembali

Peran: 6
staff_r
user_r
object_r
sysadm_r
system_r
unconfined_r

Sepanjang baris yang sama, menjalankan perintah ' seinfo -t 'mengembalikan daftar panjang semua tipe yang digunakan dalam label konteks. Daftar di laptop saya ini memiliki 1041 entri. Daftarnya dimulai dengan:

Jenis: 1041
bluetooth_conf_t
etc_runtime_t
audisp_var_run_t
auditd_var_run_t
ipsecnat_port_t
. . .
. . .
. . .

- Untuk mendapatkan gambaran tentang bagaimana kontrol akses berbutir halus dengan SELinux dapat dibuat, jalankan perintah ' seinfo -a 'untuk melihat daftar sejumlah besar atribut yang disertakan dengan label jenis. Ketika saya menjalankan perintah ini, saya mendapatkan daftar dengan 174 entri. Berikut adalah bagaimana daftar itu dimulai:

Atribut: 174
direct_init
privfd
file_type

mlsnetinbound
can_setenforce
exec_type
xproperty_type
dbusd_unconfined
kern_unconfined
mlsxwinwritecolormap.dll
node_type
packet_type
proc_type
port_type
. . .
. . .
. . .

- Jika Anda penasaran, Anda dapat melihat label konteks yang ditetapkan ke nama akun Anda dengan memasukkan 'biasa' Indo 'perintah. Sebelum menginstal SELinux, perintah ini hanya mengembalikan ID pengguna dan ID grup yang terkait dengan akun Anda. Namun, setelah menginstal SELinux, saya mendapatkan kembali yang berikut (semua dalam satu baris):

uid = 1001 (kak) gid = 1001 (kak) grup = 1001 (kak), 4 (adm), 7 (lp), 27 (sudo), 109 (lpadmin), 124 (sambashare) konteks = system_u: system_r: kernel_t : s0

Apa yang Anda lihat di bagian akhir adalah label konteks yang terkait dengan akun saya. Jika saya hanya ingin melihat label konteks, saya dapat menjalankan perintah ' id -Z '. Menjalankan perintah ini menghasilkan

system_u: system_r: kernel_t: s0

yang mengatakan bahwa saya adalah a sistem u pengguna (mungkin karena hak sudo saya), peran saya adalah sistem r, bahwa jenis label yang terkait dengan saya adalah kernel t, dan itulah level saya s0.

- Ingat enam pengguna SELinux berikut yang dikembalikan oleh perintah ' seinfo -u ': sysadm u, sistem u, root, staf u, pengguna u, dan unconfined u. Misalkan kita ingin mengetahui peran apa saja yang dapat dimainkan oleh masing-masing pengguna ini, kita dapat menjalankan perintah ' sudo semanage pengguna -l '. Perintah ini mengembalikan:

Pengguna SELinux	Pelabelan Awalan	MLS / Rentang MCS	MLS / Level MCS	Peran SELinux
akar	pengguna	s0	s0-s0: c0.c255	staff_r sysadm_r system_r
staff_u	pengguna	s0	s0-s0: c0.c255	staff_r sysadm_r
sysadm_u	pengguna	s0	s0-s0: c0.c255	sysadm_r
system_u	pengguna	s0	s0-s0: c0.c255	system_r
unconfined_u	pengguna	s0	s0-s0: c0.c255	system_r unconfined_r
user_u	pengguna	s0	s0	user_r

Tampilan ini menunjukkan bahwa ' akar 'pengguna di laptop saya diizinkan untuk memperoleh salah satu dari tiga peran: staf r, sysadm r, dan sistem r. Namun, karena sebagai 'kak' saya adalah seorang sistem u pengguna, satu-satunya peran yang diizinkan adalah sistem r.

- Sekarang mari kita bicara tentang kolom keempat dalam presentasi tabel yang dikembalikan oleh perintah ' sudo semanage pengguna -l '. Ini adalah kolom dengan tajuk “Rentang MLS / MCS”. Setiap entri di kolom ini terdiri dari dua bagian yang dipisahkan oleh titik dua. Apa yang ada di sebelah kiri usus besar adalah kisaran level

yang diizinkan untuk setiap pengguna SELinux (di mana, sekali lagi, yang kami maksud dengan 'pengguna' adalah salah satu dari enam label pengguna yang dipahami SELinux). Seperti yang Anda ingat, di awal bagian ini kita berbicara tentang MLS singkatan dari Multi-Level Security yang dimungkinkan oleh bidang level dalam label konteks. Apa yang ditampilkan di sebelah kanan titik dua - penting untuk implementasi kebijakan kontrol akses MCS (Multi-Category Security) yang disebutkan sebelumnya - adalah kisaran kategori yang diizinkan untuk setiap pengguna SELinux. Anda dapat, misalnya, mengasosiasikan sekumpulan kategori yang berbeda dengan setiap file dalam direktori. Seorang pengguna akan dapat mengakses file dalam direktori itu hanya jika pengguna tersebut termasuk dalam semua kategori yang ditentukan untuk file itu. Sintaks deklarasi ' c0.c255 'adalah singkatan untuk kategori c0 melalui c255. Ketika kontrol akses berbasis MCS digunakan, itu datang setelah kontrol akses yang ditetapkan oleh LDAC, dan batasan akses yang dibuat melalui penegakan kontrol akses berbasis peran, berbasis tipe, dan berbasis level. Jadi MCS hanya dapat membatasi sumber daya apa yang dapat diakses di komputer. [Seperti yang disebutkan sebelumnya di bagian ini, kontrol akses yang disediakan oleh distribusi standar Linux disebut sebagai Linux Discretionary Kontrol Akses (DAC).]

- Jika perlu, Anda dapat menonaktifkan SELinux dengan perintah seperti

```
sudo setenforce 0
```

dan mengaktifkannya kembali dengan

```
sudo setenforce 1
```

Seperti yang disebutkan sebelumnya, Anda dapat memeriksa status SELinux yang berjalan di komputer kami dengan

```
sestatus
```

Jika tertulis "menegakkan", itu berarti SELinux memberikan perlindungan. Untuk sepenuhnya menonaktifkan penginstalan SELinux di mesin Anda, ubah variabel SELINUX di
/ etc / selinux / config file untuk dibaca

```
SELINUX = dinonaktifkan
```

- Jika Anda mengalami semacam kemacetan setelah menginstal SELinux di sebuah host, mungkin Anda dapat mencoba menjalankan perintah ' sudo setenforce 0 'di host itu untuk menempatkan SELinux dalam mode permisif. Untuk menguraikan dengan sebuah contoh, katakanlah Anda mencoba scp file ke host berkemampuan SELinux sebagai pengguna bernama ' xxxx ' (berasumsi bahwa xxxx memiliki hak masuk di host) dan tidak berfungsi. Anda memeriksa
'/ var / log / auth.log file di host dan Anda melihat kesalahan di sana pesan " gagal mendapatkan konteks keamanan SELinux default untuk xxxx (dalam mode pemberlakuan) ". Dalam urutan untuk mengatasi masalah ini, Anda perlu memperbaiki label konteks yang terkait dengan pengguna 'xxxx'. Selain itu, Anda juga dapat sementara waktu menempatkan host dalam mode permisif melalui perintah ' sudo setenforce 0 'dan menyelesaikan pekerjaan.

- Apa yang dicapai di Linux dengan MAC dicapai dalam sistem Windows dengan Mandatory Integrity Control (MIC) itu

mengaitkan salah satu dari lima Tingkat Integritas (IL) berikut dengan proses: Penginstal Rendah, Sedang, Tinggi, Sistem, dan Tepercaya.

- Meskipun kami mencapai keamanan yang signifikan dengan melakukan sandbox pada aplikasi, orang tidak dapat terbuai dengan pemikiran bahwa itulah jawaban untuk semua kerentanan terkait sistem di perangkat komputasi. Ketika sampai pada masalah terkait sistem dalam keamanan komputer, berikut beberapa pemikiran untuk dipikirkan: Apakah mungkin bootstrap loader OS Anda (seperti bootloader GRUB) dapat digunakan oleh program jahat untuk mengunduh kernel OS yang rusak? Mungkinkah / sbin / init file (yang digunakan untuk meluncurkan proses init dari mana semua proses lain muncul di platform Unix / Linux) dapat dengan sendirinya diganti dengan versi yang rusak? Dan bagaimana dengan musuh yang mengeksploitasi ptrace.dll alat yang biasanya digunakan di Linux oleh satu proses untuk mengamati dan mengontrol pelaksanaan proses lain?

Kembali ke Daftar Isi

32.4 TENTANG KEAMANAN KOMUNIKASI OVER-THE-AIR DENGAN PERANGKAT SELULER?

- Bahkan jika kita berpikir bahwa smartphone kita bebas dari malware, apa yang akan menyiratkan sehubungan dengan kemampuan perangkat untuk terlibat dalam komunikasi suara dan data yang aman dengan stasiun pangkalan operator seluler? Ini pertanyaannya yang akan saya bahas secara singkat di bagian ini.

- Jawaban atas pertanyaan yang diajukan di atas tergantung pada generasi teknologi ponsel yang Anda bicarakan. Seperti yang Anda ketahui, kami sekarang memiliki 2G (GSM), 3G (UMTS), 4G (LTE, ITU), dan sekarang standar nirkabel 5G untuk komunikasi telepon seluler. Algoritme yang digunakan untuk mengenkripsi komunikasi suara dan data melalui udara dengan standar yang berbeda ini disebut sebagai algoritme seri A5. Algoritme yang digunakan untuk mengenkripsi suara dan SMS dalam standar 2G (yang masih mendominasi sebagian besar wilayah geografis di dunia) adalah stream cipher A5 / 1. A5 / 2, versi lebih lemah dari A5 / 1 yang dibuat untuk memenuhi batasan ekspor tertentu sekitar satu dekade lalu, ternyata merupakan sandi yang sangat lemah dan telah dihentikan. A5 / 3 dan A5 / 4 dimaksudkan untuk teknologi nirkabel 3G dan 4G. [GSM

standar mendefinisikan seperangkat algoritma untuk layanan enkripsi dan otentikasi. Algoritme ini diberi nama 'Ax' di mana 'x' dalam bilangan bulat yang menunjukkan fungsi algoritme. Misalnya, pemancar dapat menelepon pada algoritme A3 untuk mengautentikasi perangkat seluler. Algoritma A5 menyediakan enkripsi / dekripsi jasa. Algoritma A8 digunakan untuk menghasilkan kunci sesi 64 bit. Algoritme dengan nama COMP128 menggabungkan fungsionalitas A3 dan A8.]

- Baik A5 / 3 dan A5 / 4 didasarkan pada block cipher KASUMI, yang pada gilirannya didasarkan pada block cipher yang disebut MISTY1 yang dikembangkan oleh Mitsubishi. Sandi KASUMI digunakan dalam Mode Umpan Balik Keluaran yang kita bicarakan di Kuliah 9, yang menghasilkan aliran bit dalam kelipatan 64 bit. Mengenai KASUMI, ini adalah cipher blok 64-bit dengan struktur Feistel (yang Anda pelajari di Kuliah 3) dengan delapan putaran. KASUMI membutuhkan kunci enkripsi 128-bit.

- Sisa dari bagian ini, dan sub-bagian berikutnya, berfokus pada cipher A5 / 1 yang digunakan secara luas di jaringan seluler 2G. Sekarang sudah diketahui dengan baik bahwa sandi ini pada dasarnya tidak memberikan keamanan karena kecepatan yang dapat digunakan untuk membobolnya menggunakan perangkat keras komputasi biasa.

- Yang membuat A5 / 1 menarik adalah bahwa ini adalah studi kasus yang bagus tentang bagaimana hal-hal bisa salah jika Anda percaya keamanan melalui ketidakjelasan. Seperti yang saya sebutkan di Bagian 32.1, algoritma ini dirahasiakan selama beberapa tahun oleh operator ponsel. Tapi, akhirnya, itu bocor dan ditemukan hampir tidak memberikan

keamanan terkait privasi data suara dan pesan SMS.

- A5 / 1 adalah stream cipher tingkat bit dengan kunci enkripsi 64-bit. Kunci enkripsi dibuat untuk setiap sesi dari kunci master yang digunakan bersama oleh operator ponsel (yang digunakan untuk mendaftarkan ponsel) dan kartu SIM di ponsel. Ketika sebuah stasiun pangkalan (yang mungkin dimiliki oleh beberapa operator ponsel lain) membutuhkan kunci sesi, itu mengambilnya dari operator ponsel yang memegang kunci master.

- Transmisi GSM meledak. Multiplexing pembagian waktu digunakan untuk dengan cepat mengirimkan aliran bit yang dikumpulkan yang perlu dikirim melalui tautan komunikasi yang diberikan antara stasiun pangkalan dan telepon. Satu semburan di setiap arah terdiri dari 114 bit dengan durasi 4,615 milidetik.

- Tujuan dari A5 / 1 adalah untuk menghasilkan dua aliran pseudorandom 114-bit - disebut keystream - satu untuk uplink dan yang lainnya untuk downlink. Data 114-bit di setiap arah di-XOR dengan keystream. Tujuan dapat memulihkan data asli dengan melakukan XOR pada aliran bit yang diterima dengan aliran kunci yang sama.

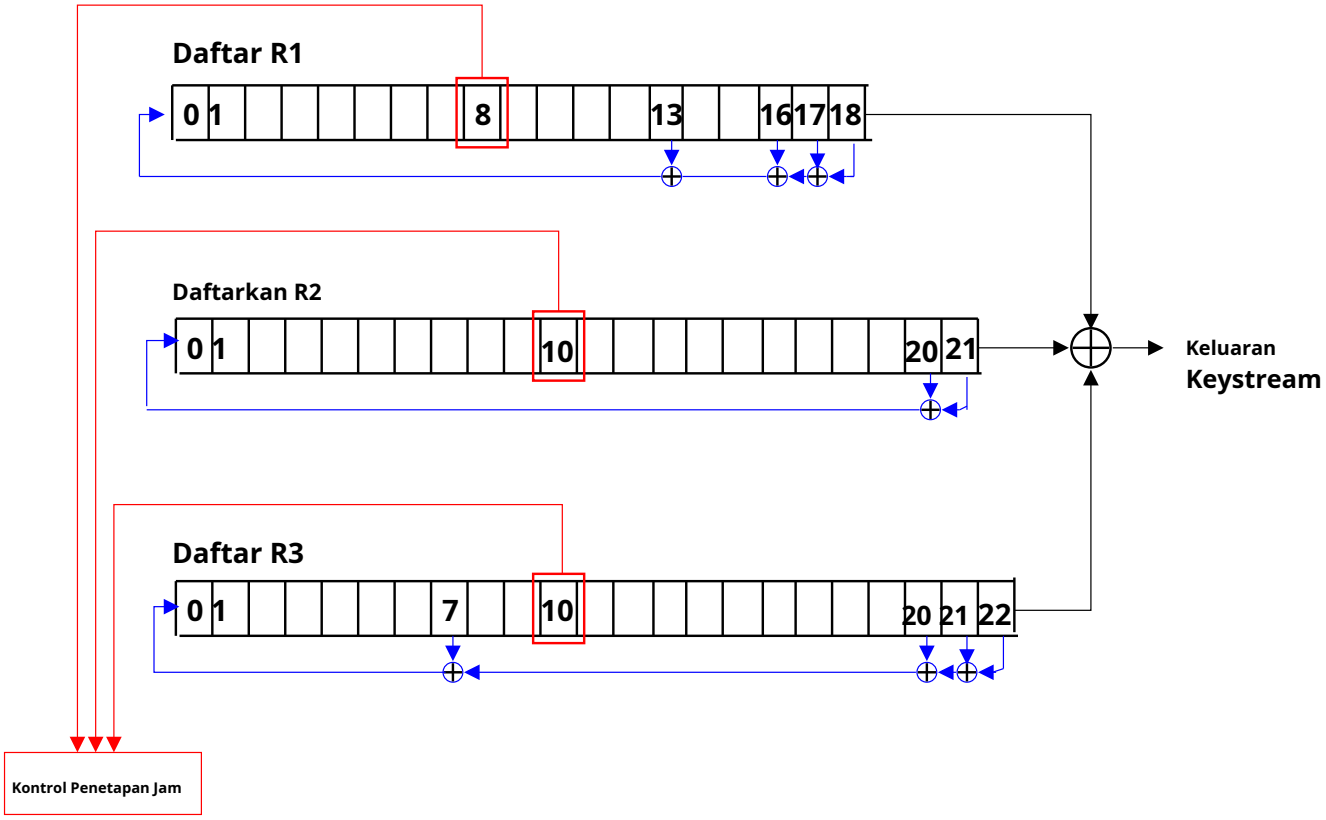
- Selain kunci 64-bit, enkripsi dari setiap aliran 114-bit juga dikontrol oleh nomor bingkai 22-bit yang

selalu dikenal publik.

- A5 / 1 bekerja pada tiga LFSR (Linear Feedback Shift Register), yang ditunjuk R1, R2, dan R3, dengan ukuran 19, 22, dan 23 bit, seperti yang ditunjukkan pada Gambar 1. Setiap register geser diinisialisasi dengan kunci enkripsi 64-bit dan nomor bingkai 22-bit dengan cara yang diilustrasikan oleh kode Python di sub-bagian berikutnya.

- Setiap register geser memiliki apa yang dikenal sebagai a sedikit clocking - untuk setiap register ditandai dengan kotak merah pada Gambar 1. Seperti yang Anda ketahui dari gambar, untuk R1, bit clocking berada pada indeks 8, dan untuk R2 dan R3 pada indeks 10. Selama produksi keystream, bit clocking digunakan untuk memutuskan apakah atau tidak
untuk mencatat waktu register geser. [Perhatikan perbedaan antara sedikit clocking, dari yang HANYA ada SATU di setiap register geser, dan bit umpan balik, yang masing-masing berisi BEBERAPA register geser, seperti yang ditunjukkan pada Gambar 1.]

- Clocking register geser melibatkan operasi berikut: (1) Anda merekam bit pada tap umpan balik di register; (2) Anda menggeser register dengan posisi satu bit ke arah MSB (yang ada di ujung kanan register geser, ujung dengan posisi bit terindeks tertinggi) ; dan (3) Anda mengatur nilai LSB ke XOR dari bit umpan balik. Ketika Anda pertama kali menginisialisasi register dengan kunci enkripsi, Anda menambahkan langkah keempat, yaitu ke XOR LSB dengan bit kunci yang sesuai dengan detak jam itu, dll.



Gambar 1: Gambar ini menunjukkan bagaimana tiga Register Geser Umpan Balik Linear digunakan dalam algoritma A5 / 1 untuk mengenkripsi suara dan SMS di jaringan selular 2G. (Angka ini dari Kuliah 32 "Catatan Kuliah Keamanan Komputer dan Jaringan" oleh Avi Kak)

- Setelah register geser diinisialisasi, Anda menghasilkan keystream dengan melakukan hal berikut di setiap detak jam:

  - Anda mengambil suara mayoritas bit clocking di tiga register R1, R2, dan R3. Pemilihan mayoritas berarti Anda mengetahui apakah setidaknya dua dari tiga adalah 0 atau 1.

  - Anda hanya mencatat register yang bit clockingnya sesuai dengan bit mayoritas.

  - Anda mengambil XOR dari MSB dari tiga register dan itu menjadi bit keluaran.

- Subbagian berikutnya menyajikan implementasi Python dari logika ini untuk menghilangkan ambiguitas tentang berbagai langkah yang diuraikan di atas.

- A5 / 1 telah menjadi subyek kriptanalisis oleh beberapa peneliti. Serangan terbaru terhadap A5 / 1, oleh Karsten Nohl, dipresentasikan pada konferensi Black Hat 2010. PDF makalah tersedia di:

  https://srlabs.de/blog/wp-content/uploads/2010/07/Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf

  Berikut kutipan dari makalah Karsten Nohl:

> "..... A5 / 1 bisa rusak dalam hitungan detik dengan penyimpanan cepat 2TB dan dua kartu grafis. Serangan tersebut menggabungkan beberapa teknik pertukaran memori waktu dan mengeksploitasi ukuran kunci efektif yang relatif kecil yaitu 61 bit "

Nohl telah mendemonstrasikan bahwa serangan meja pelangi dapat dipasang dengan sukses pada A5 / 1. Anda belajar tentang tabel pelangi di Kuliah 24.

- Makalah lain yang menarik (tetapi lebih teoritis) tentang pemasangan serangan pada A5 / 1 adalah "Cryptanalysis of the A5 / 1 GSM Stream Cipher" oleh Eli Biham dan Orr Dunkelman yang muncul dalam Progress in Cryptology - INDOCRYPT, 2000. Publikasi penting lainnya yang berbicara tentang cryptanalysis dari cipher A5 adalah "Cryptanalysis Instan-Hanya Cryptanalysis Komunikasi GSM Terenkripsi" oleh Elad Barkan, Eli Biham, dan Nathan Keller.

Kembali ke Daftar Isi

32.4.1 Implementasi Python dari A5 / 1 Sandi

- Agar Anda dapat lebih memahami langkah-langkah algoritmik untuk stream cipher A5 / 1 yang dijelaskan di bagian sebelumnya, sekarang saya akan menyajikan di sini implementasi Python-nya. Seperti yang dikatakan oleh blok komentar di bagian atas file kode, implementasi Python saya didasarkan pada kode C yang disediakan untuk algoritme oleh Marc Briceno, Ian Goldberg, dan David Wagner.

- Baris (1) kode mendefinisikan tiga register R1, R2, dan R3 sebagai tiga BitVektor dengan ukuran masing-masing 19, 22, dan 23 bit. Cara terbaik untuk memvisualisasikan register ini seperti yang ditunjukkan pada Gambar 1. BitVectors yang dibangun sebenarnya akan berisi LSB di ujung kiri dan MSB di ujung kanan.

- Baris (2) mendefinisikan BitVektor yang diperlukan untuk tap umpan balik pada R1, R2, dan R3. Kami mengatur bit tap di Baris (3), (4) dan (5). Kita bisa mendapatkan bit umpan balik di setiap register hanya dengan mengambil logika AND dari register BitVectors, seperti yang dijelaskan di Baris (1), dan tap BitVektor, seperti yang dijelaskan di Baris (2).

- Baris (9) sampai (11) mengatur kunci enkripsi. Kunci ini bisa

jelas dapat diatur ke apa pun asalkan panjangnya 64 bit. Nilai spesifik yang ditampilkan untuk kunci tersebut sama dengan yang digunakan oleh Briceno, Goldberg, dan Wagner dalam kode C mereka.

- Dengan cara yang sama, Garis (12) dan (13) mengatur nomor bingkai yang harus berupa nomor 22-bit. Saya telah menggunakan nomor yang sama dengan Briceno et al.

- Baris (14) dan (15) mendefinisikan dua BitVektor sepanjang 114-bit yang digunakan nanti untuk menyimpan dua arus kunci keluaran.

- Baris (16) sampai (32) mendefinisikan rutinitas dukungan keseimbangan(), mayoritas (), clockone (), dan clockall (). Definisi mereka harus menjelaskan logika yang digunakan dalam fungsi ini.

- Itu setupkey () di Baris (33) hingga (44) menginisialisasi tiga register geser dengan, pertama, clocking di 64 bit kunci enkripsi, kemudian, dengan clocking di 22 bit nomor frame, dan, akhirnya, dengan hanya mencatat register 100 kali untuk efek "longsoran". Perhatikan perbedaan penting antara bagaimana register dicatat di Baris (34) sampai (39) dan di Baris (40) sampai (44). Di Baris (34) hingga (39), kita mencatat ketiga register di setiap detak jam. Namun demikian, pada baris (40) sampai (44), sebuah register diberi clock tergantung pada bagaimana bit clockingnya dibandingkan dengan bit clocking pada dua register lainnya.

- Fungsi yang menghasilkan keystream, Lari(), didefinisikan dalam Baris (45) sampai (55). Saya telah menggabungkan produksi dua keystream menjadi satu loop 228-iterasi tunggal di Garis (48) hingga (53). 114 bit pertama yang dihasilkan dengan cara ini adalah untuk aliran kunci uplink dan 114 bit berikutnya untuk aliran kunci downlink. Ini direfleksikan oleh pembagian yang dibuat dalam baris (54) dan (55).

- Sisa kode ini untuk memeriksa keakuratan implementasi terhadap vektor uji yang disediakan oleh Briceno et al. dalam implementasi berbasis C mereka. Variabel goodAtoB dan goodBtoA menyimpan nilai yang benar untuk dua arus kunci untuk kunci enkripsi Baris (9) dan nomor bingkai Baris (12).

```
#! / usr / bin / env python

# # A5_1.py
# # Avi Kak ( kak@purdue.edu )
# # 21 April 2015

# #    Ini adalah implementasi Python dari kode C yang disediakan oleh Marc Briceno, Ian Goldberg, dan
# #    David Wagner di situs web berikut:
# #
# #        http://www.scard.org/gsm/a51.html
# #
# #    Untuk akurasi, saya telah membandingkan keluaran kode Python ini dengan vektor uji yang disediakan
# #    oleh mereka.

# #    Algoritme A5 / 1 digunakan dalam 2G GSM untuk enkripsi data suara dan SMS melalui udara. Atas dasar
# #    pembacaan sandi ini dan serangan tabel pelangi yang lebih baru, algoritme A5 / 1 sekarang dianggap tidak
# #    memberikan keamanan sama sekali. Meskipun demikian, ini membentuk studi kasus yang menarik yang
# #    menunjukkan bahwa ketika algoritme keamanan tidak dibuka untuk pengawasan publik (karena beberapa
# #    orang di luar sana percaya pada "keamanan melalui ketidakjelasan"), mungkin saja algoritme semacam itu
# #    diterapkan pada dunia yang benar-benar global. dasar sebelum kekurangannya menjadi jelas.
# #
# #
```

```
# #    Algoritma A5 / 1 adalah stream cipher tingkat bit berdasarkan tiga LFSR (Linear Feedback Shift Register).
# #    Operasi dasar yang Anda lakukan dalam LFSR pada setiap detak jam terdiri dari tiga langkah berikut: (1)
# #    Anda merekam bit pada tap umpan balik di register; (2) Anda menggeser register dengan posisi satu bit ke
# #    arah MSB; dan (3) Anda menetapkan nilai LSB ke XOR dari bit umpan balik. Ketika Anda pertama kali
# #    menginisialisasi register dengan kunci enkripsi, Anda menambahkan langkah keempat, yaitu ke XOR LSB
# #    dengan bit kunci yang sesuai dengan detak jam itu, dll.
# #
# #

dari impor BitVector *

# Tiga register geser
R1, R2, R3 = BitVector (ukuran = 19), BitVector (ukuran = 22), BitVector (ukuran = 23)                    # (1)

# Ketukan umpan balik
R1TAPS, R2TAPS, R3TAPS = BitVector (ukuran = 19), BitVector (ukuran = 22), BitVector (ukuran = 23)        # (2)
R1TAPS [13] = R1TAPS [16] = R1TAPS [17] = R1TAPS [18] = 1                                                 # (3)
R2TAPS [20] = R2TAPS [21] = 1                                                                             # (4)
R3TAPS [7] = R3TAPS [20] = R3TAPS [21] = R3TAPS [22] = 1                                                  # (5)

cetak "R1TAPS:", cetak R1TAPS                                                                             # (6)
"R2TAPS:", cetak R2TAPS                                                                                   # (7)
"R3TAPS:", R3TAPS                                                                                         # (8)

keybytes = [BitVector (hexstring = x) .reverse () untuk x dalam ['12', '23', '45', '67', \
                                                                 '89', 'ab', 'cd', 'ef']]                 # (9)
key = mengurangi (lambda x, y: x + y, keybytes) print                                                     # (10)
"kunci enkripsi:", kunci                                                                                  # (11)

frame = BitVector (intVal = 0x134, size = 22) .reverse () print "nomor                                    # (12)
bingkai:", bingkai                                                                                        # (13)

# # Kami akan menyimpan dua arus kunci keluaran di dua BitVektor ini, masing-masing berukuran 114
# # bit. Satu untuk uplink dan yang lainnya untuk downlink: AtoBkeystream =
    BitVector (size = 114)                                                                                # (14)
BtoAkeystream = BitVector (ukuran = 114)                                                                  # (15)

# # Fungsi ini digunakan oleh fungsi clockone (). Seperti setiap register geser
# # clocked, umpan balik terdiri dari paritas semua bit tap: def paritas (x):
                                                                                                          # (16)
    countbits = x.count_bits ()                                                                           # (17)
    mengembalikan hitungan% 2                                                                             # (18)

# # Untuk memutuskan apakah register geser harus dicatat atau tidak
# # clock tick, kita perlu memeriksa bit clocking di setiap register dan melihat apa
# # mayoritas mengatakan:
def mayoritas ():                                                                                         # (19)
    jumlah = R1 [8] + R2 [10] + R3 [10] jika                                                              # (20)
    jumlah> = 2:                                                                                          # (21)
        return 1                                                                                          # (22)
    lain:                                                                                                 # (23)
        return 0                                                                                          # (24)

# # Fungsi ini hanya menjalankan satu register yang disediakan sebagai arg pertama ke
# # fungsi. Argumen kedua harus menunjukkan posisi bit dari umpan balik
```

```
# # ketuk untuk mendaftar.
def clockone (register, tap):                                                    # (25)
    tapsbits = daftar & ketuk                                                    # (26)
    register.shift_right (1)                                                     # (27)
    register [0] = paritas (tapsbits)                                            # (28)

# # Fungsi ini diperlukan untuk menginisialisasi tiga register geser. def clockall ():
                                                                                 # (29)
    clockone (R1, R1TAPS)                                                        # (30)
    clockone (R2, R2TAPS)                                                        # (31)
    clockone (R3, R3TAPS)                                                        # (32)

# #   Fungsi ini menginisialisasi tiga register geser dengan, pertama, kunci enkripsi 64-bit, kemudian dengan
# #   nomor bingkai 22 bit, dan, terakhir, dengan hanya mencatat register 100 kali untuk menciptakan efek
# #   'longsoran salju'. Perhatikan bahwa selama pembuatan longsoran salju, pencatatan jam kerja setiap
# #   register sekarang bergantung pada bit pencatatan jam kerja di ketiga register.
# #
def setupkey ():                                                                 # (33)
    # Jam ke register 64 bit kunci enkripsi: untuk i dalam jangkauan (64):
                                                                                 # (34)
        jam ()                                                                   # (35)
        R1 [0] ^ =^kunci [i]; R2 [0] ^ =^kunci [i]; R3 [0] ^ =^kunci [i]         # (36)
    # Jam ke register 22 bit nomor bingkai: untuk i dalam kisaran (22):
                                                                                 # (37)
        jam ()                                                                   # (38)
        R1 [0] ^ =^bingkai [i]; R2 [0] ^ =^bingkai [i]; R3 [0] ^ =^bingkai [i]   # (39)
    # Sekarang clock ketiga register 100 kali, tapi kali ini biarkan clocking
    # dari setiap register bergantung pada pemungutan suara mayoritas bit clocking: untuk i
      dalam kisaran (100):
                                                                                 # (40)
        maj = mayoritas ()                                                       # (41)
        if (R1 [8]! = 0) == maj: clockone (R1, R1TAPS) if (R2 [10]! = 0)         # (42)
        == maj: clockone (R2, R2TAPS) if (R3 [10]! = 0) == maj:                  # (43)
        clockone (R3, R3TAPS)                                                    # (44)

# # Setelah tiga register geser diinisialisasi dengan kunci enkripsi dan
# # nomor bingkai, Anda siap menjalankan register geser untuk menghasilkan dua bit 114
# # bit keystream panjang, satu untuk uplink dan lainnya untuk downlink. def run ():
                                                                                 # (45)
    global AtoBkeystream, BtoAkeystream                                          # (46)
    keystream = BitVector (ukuran = 228)                                         # (47)
    untuk saya dalam jangkauan (228):                                            # (48)
        maj = mayoritas ()                                                       # (49)
        if (R1 [8]! = 0) == maj: clockone (R1, R1TAPS) if (R2 [10]! = 0)         # (50)
       == maj: clockone (R2, R2TAPS) if (R3 [10]! = 0) == maj: clockone          # (51)
       (R3, R3TAPS) keystream [i] = R1 [-1] ^ R2 [-1] ^ R3 [-1]                  # (62)
       AtoBkeystream = keystream [: 114]                                         # (53)
                                                                                 # (54)
    BtoAkeystream = keystream [114:]                                             # (55)

# # Inisialisasi tiga register geser: setupkey ()
                                                                                 # (56)
# # Sekarang buat keystream: run ()
                                                                                 # (57)

# # Tampilkan dua arus utama:
```

```
cetak "\ nAtoBkeystream:          ", AtoBkeystream                                    # (58)
cetak "\ nBtoAkeystream:          ", BtoAkeystream                                    # (59)

# # Berikut adalah nilai yang benar untuk dua arus utama:
goodAtoB = [BitVector (hexstring = x) untuk x dalam ['53', '4e', 'aa', '58', '2f', 'e8', '15', '1a', \
                                                     'b6', 'e1', '85', '5a', '72', '8c', '00']]   # (60)
goodBtoA = [BitVector (hexstring = x) untuk x dalam ['24', 'fd', '35', 'a3', '5d', '5f', 'b6', '52', \
                                                     '6d', '32', 'f9', '06', 'df', '1a', 'c0']]   # (61)
goodAtoB = kurangi (lambda x, y: x + y, goodAtoB)                                    # (62)
goodBtoA = kurangi (lambda x, y: x + y, goodBtoA)                                    # (63)

print "\ nBaik: AtoBkeystream:", goodAtoB [: 114] print "\ nBaik:                    # (64)
BtoAkeystream:", goodBtoA [: 114]                                                    # (65)

if (AtoBkeystream == goodAtoB [: 114]) dan (AtoBkeystream == goodAtoB [: 114]):      # (66)
        print "\ nPemeriksaan mandiri berhasil: Semuanya terlihat bagus"             # (67)
```

- Ketika Anda menjalankan kode ini, Anda akan melihat output berikut

  R1TAPS: 0000000000000100111
  R2TAPS: 00000000000000000011
  R3TAPS: 000000010000000000000111
  kunci enkripsi: 010010001100010010100010111001101001000111010101101100111111 0111 nomor bingkai: 00101100100000000000

  AtoBkeystream: 01010011010011101010101001011000001011111110100000010101000110101011011100011000010101011010011100101000110000

  BtoAkeystream: 00100100111111010011010110100011010111010101111110110110010100100110110100110010111110010000011011011111000110101 1

  AtoBkeystream yang bagus: 01010011010011101010101001011000001011111110100000010101000110101011011100011000010101011010011100101000110000

  BtoAkeystream yang bagus: 00100100111111010011010110100011010111010101111110110110010100100110110100110010111110010000011011011111000110101 1

  Pemeriksaan mandiri berhasil: Semuanya terlihat bagus

- Anda mungkin bertanya-tanya mengapa saya tidak menunjukkan keystream dalam hex. Secara umum, Anda dapat menampilkan objek BitVector dalam hex dengan memanggil metode instance-nya

dapatkan hex dari bitvector () - asalkan jumlah bitnya adalah kelipatan 4. Keystream kami memiliki panjang 114 bit, yang bukan kelipatan 4. Saya bisa menambah keystream dengan menambahkan beberapa angka nol di akhir, tetapi kemudian Anda mengambil kebebasan dengan kebenaran dari keluarannya.

Kembali ke Daftar Isi

32.5 SERANGAN SISI SALURAN PADA PERANGKAT SELULER KHUSUS

- Sekarang saya akan menjelaskan serangan yang paling baik dilakukan jika musuh memiliki kepemilikan fisik atas perangkat komputasi. Oleh karena itu, pada dasarnya, perangkat seluler rentan terhadap serangan bentuk ini - terutama perangkat seluler yang lebih terspesialisasi seperti kartu pintar yang berisi perangkat keras dan perangkat lunak yang belum sempurna dibandingkan dengan yang Anda temukan di ponsel cerdas saat ini.
  Dengan secara fisik menundukkan koneksi perangkat keras pada perangkat tersebut ke kesalahan sesaat yang disuntikkan secara eksternal (katakanlah dengan lonjakan tegangan transien dari sumber eksternal), atau dengan mengukur waktu yang dibutuhkan oleh rutin kriptografi untuk sejumlah besar input, dimungkinkan untuk tebak dengan baik parameter keamanan perangkat tersebut.

- Sebelum membaca bagian ini lebih lanjut (dan juga sebelum membaca Bagian 32.7 dan 32.8), Anda harus membaca pembicaraan Black Hat 2008 Karsten Nohl di tautan yang ditunjukkan di bawah ini. Pembicaraan ini akan memberi Anda gambaran yang baik tentang sifat intrusif dari serangan yang dapat Anda pasang di perangkat seperti kartu pintar untuk memecahkan enkripsi:

  https://www.blackhat.com/presentations/bh-usa-08/Nohl/BH_US_08_Nohl_Mifare.pdf

- Secara umum, serangan saluran samping berarti bahwa musuh mencoba memecahkan sandi menggunakan informasi yang BUKAN bersifat intrinsik pada detail matematika dari algoritme enkripsi / dekripsi, tetapi dapat disimpulkan dari berbagai pengukuran "eksternal" seperti daya dikonsumsi oleh perangkat keras yang menjalankan algoritme untuk berbagai masukan yang mungkin, waktu yang dibutuhkan oleh perangkat keras untuk hal yang sama, bagaimana perangkat keras merespons kesalahan yang diinjeksi secara eksternal, dll.

- Berbagai bentuk serangan saluran samping adalah:

Serangan Injeksi Kesalahan: Ini didasarkan pada sengaja mendapatkan perangkat keras yang menjalankan bagian tertentu dari algoritma enkripsi / dekripsi untuk memberikan jawaban yang salah. Seperti yang ditunjukkan di bagian selanjutnya, jawaban yang salah mungkin memberikan petunjuk yang memadai untuk mengetahui parameter algoritma kriptografi yang digunakan.

Waktu Serangan: Serangan ini mencoba menyimpulkan kriptografi kunci dari waktu yang diperlukan prosesor untuk menjalankan algoritme dan ketergantungan waktu ini pada input yang berbeda.

Serangan Analisis Kekuatan: Di sini tujuannya adalah untuk menganalisis jejak kekuatan dari algoritma kriptografi yang sedang dieksekusi untuk mengetahui apakah instruksi tertentu telah dieksekusi

pada waktu tertentu. Telah terbukti bahwa jejak seperti itu dapat mengungkapkan kunci kriptografi yang digunakan.

Serangan Analisis EM: Dengan asumsi bahwa perangkat keras menerapkan rutinitas kriptografi tidak cukup terlindung dari kebocoran radiasi elektromagnetik (pada frekuensi clock prosesor), jika Anda dapat membuat jejak radiasi ini, Anda mungkin dapat menyimpulkan apakah instruksi tertentu dijalankan pada waktu tertentu atau tidak. - seperti dalam serangan analisis kekuatan. Dari informasi tersebut, Anda mungkin dapat menarik kesimpulan tentang bit dalam kunci enkripsi.

- Pada bagian selanjutnya, saya akan membahas dua dari serangan ini secara lebih rinci: serangan injeksi kesalahan dan serangan waktu. Untuk menjelaskan prinsip-prinsip yang terlibat, untuk kedua serangan ini, saya akan berasumsi bahwa perangkat seluler diisi dengan penandatanganan pesan keluar secara digital dengan algoritma RSA. Tujuan dari serangan tersebut adalah menebak eksponen pribadi yang digunakan untuk membuat tanda tangan digital. Catat itu hari-hari ini jika sebuah serangan dapat dengan andal menebak bahkan satu rahasia pun, itu dianggap sebagai serangan yang berhasil.

Kembali ke Daftar Isi

32.6 SERANGAN INJEKSI KESALAHAN

- Tujuan dari bagian ini adalah untuk menunjukkan bahwa jika Anda dapat membuat prosesor perangkat seluler menghasilkan nilai yang salah untuk sebagian kalkulasi, Anda mungkin dapat membuat perangkat berpisah dengan rahasianya, yang dapat berupa enkripsi. kunci yang Anda cari.

- Saya akan berasumsi bahwa prosesor perangkat seluler memiliki kunci pribadi tertanam untuk menandatangani pesan secara digital dengan algoritma RSA.

- Pembaca akan mengingat kembali dari Kuliah 12 yang diberi modulus n dan pasangan kunci publik dan pribadi (e, d), kita bisa menandatangani pesan M dengan menghitung tanda tangan digitalnya $S = M^d \bmod n$. [Dalam praktek, Anda cenderung menghitung tanda tangan hanya dari hash pesan M. Namun, detail itu tidak ubah keseluruhan penjelasan yang disajikan di bagian ini.]

- Seperti yang dijelaskan di Bagian 12.5 dari Kuliah 12, perhitungan tanda tangan $S = M^{d(} \bmod n$) dapat dipercepat dengan menggunakan Chinese Remainder Theorem (CRT). Sejak pemilik kunci pribadi d juga akan mengetahui faktor prima p dan q dari modulus n, dengan CRT Anda pertama kali menghitung [Dalam penjelasan di Bagian

12.5 dari Kuliah 12, fokus kami adalah pada enkripsi / dekripsi dengan RSA. Oleh karena itu, eksponen privat d dulu diterapkan ke integer ciphertext C. Di sini kita berbicara tentang tanda tangan digital, yang membutuhkan penerapan eksponen pribadi ke pesan itu sendiri (atau ke hash pesan).]

$$V_p = M^d \bmod p$$
$$V_q = M^d \bmod q$$

Untuk membangun tanda tangan S dari V_p dan V_q, kita harus menghitung koefisien:

$$X_p = q \times (q^{-1} \bmod p)$$
$$X_q = p \times (p^{-1} \bmod q)$$

Teorema CRT Bagian 11.7 dari Kuliah 11 kemudian memberi tahu kita itu tandatangannya S terkait dengan hasil antara V_p dan V_q oleh

$$S = \left(V_p \times X_p + V_q \times X_q \right) \bmod n$$
$$= \left(q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q \right) \bmod n \quad (1)$$

- Sekarang mari kita asumsikan bahwa kita telah memasukkan kesalahan perhitungan V_p dengan, katakanlah, membuat perangkat keras mengalami lonjakan tegangan sesaat. Karena lonjakan tegangan dibatasi durasi, kami berasumsi bahwa sementara V_p sekarang dihitung secara keliru sebagai $\hat{V}_p$, nilai dari V_q tetap tidak berubah. Mari kita gunakan $\hat{S}$ untuk mewakili tanda tangan yang dihitung menggunakan kesalahan $\hat{V}_p$. Kita bisa menulis:

$$\hat{S} = \left(q \times (q^{-1} \bmod p) \times \hat{V}_p + p \times (p^{-1} \bmod q) \times \hat{V}_q \right) \bmod n$$

- Mengurangi tanda tangan yang salah $\hat{S}$ dari nilai aslinya S, kita punya

$$S - \hat{S} = \left(q \times (q^{-1} \bmod p) [V_p - \hat{V}_p] \right) \bmod n \qquad (2)$$

- Hasil di atas menyiratkan itu

$$q = \gcd (S - \hat{S}, n) \qquad (3)$$

Seperti yang Anda lihat, penyerang dapat segera mengetahui faktor prima q dari modulus dengan menghitung GCD dari S - $\hat{S}$ dan n. [Lihat Kuliah 5 untuk cara terbaik menghitung PBM dua angka.] Selanjutnya, a pembagian sederhana akan menghasilkan faktor prima lainnya bagi penyerang p. Dengan cara ini, penyerang akan dapat mengetahui faktor prima dari modulus RSA tanpa harus memfaktorkannya. Setelah memperoleh faktor prima p dan q, menjadi masalah sepele bagi penyerang untuk menemukan apa kunci privatnya d karena penyerang mengetahui kunci publik e.

- Taktik yang dijelaskan di atas mengharuskan penyerang menghitung kedua tanda tangan yang sebenarnya S dan tanda tangan yang salah $\hat{S}$ untuk pesan M. Ternyata, penyerang dapat melakukan eksploitasi yang sama hanya dengan tanda tangan yang salah $\hat{S}$ bersama dengan pesannya M.

- Untuk melihat mengapa eksploitasi yang sama bekerja dengan M dan $\hat{S}$ ^, perhatikan terlebih dahulu bahwa jika kita diberi tanda tangan yang benar S, kita bisa pulih M oleh $M = S^e \bmod n$. Perhatikan juga bahwa sejak itu $S^e \bmod n = M$, kita bisa menulis:

$$\begin{aligned} S^e &= k_1 \times n + M \\ &= k_1 \times p \times q + M \end{aligned}$$

untuk beberapa nilai konstanta integer k_1. Hubungan kedua yang ditunjukkan di atas mengarah ke:

$$S^e \bmod p = M \qquad (4)$$

$$S^e \bmod q = M \qquad (5)$$

- Perhatikan juga bahwa, dengan menggunakan Persamaan (1), kita dapat menulis tanda tangan yang benar:

$$\begin{aligned} S &= \Big(q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q \Big) \bmod n \\ &= q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q + k_2 \times p \times q \end{aligned}$$

untuk beberapa nilai konstanta k_2. Karena itu kami dapat menulis:

$$S^e = \Big(q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q + k_2 \times p \times q \Big)^e$$

dan itu menyiratkan

$$S^e \bmod p = (q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q + k_2 \times p \times q)^e \bmod p$$

$$= (q \times (q^{-1} \bmod p) \times V_p + p \times (p^{-1} \bmod q) \times V_q + k_2 \times p \times q \bmod p)^e \bmod p$$

$$= (q \times (q^{-1} \bmod p) \times V_p)^e \bmod p \qquad (6)$$

Kami dapat memperoleh hasil serupa untuk $S^e \bmod q$. Menuliskan dua hasil bersama, kami punya

$$S^e \bmod p = (q \times (q^{-1} \bmod p) \times V_p)^e \bmod p = M \qquad (7)$$

$$S^e \bmod q = (p \times (p^{-1} \bmod q) \times V_q)^e \bmod q = M. \qquad (8)$$

di mana kami juga telah menempatkan hasil yang diturunkan sebelumnya di Persamaan (4) dan (5).

- Sekarang mari kita coba melihat apa yang terjadi jika melakukan operasi serupa pada tanda tangan yang salah $\hat{S}\hat{}$. Namun, sebelumnya kita angkat $\hat{S}\hat{}$ ke kekuasaan e, ayo tulis ulang $\hat{S}$ sebagai

$$\hat{S} = (q \times (q^{-1} \bmod p) \times \hat{V}_p + p \times (p^{-1} \bmod q) \times V_q) \bmod n$$

$$= q \times (q^{-1} \bmod p) \times \hat{V}_p + p \times (p^{-1} \bmod q) \times V_q + k_3 \times p \times q$$

untuk beberapa nilai bilangan bulat k_3. Sekarang kita dapat menulis untuk $\hat{S}\hat{}^e$:

$$\hat{S}^e = (q \times (q^{-1} \bmod p) \times \hat{V}_p + p \times (p^{-1} \bmod q) \times V_q + k_3 \times p \times q)^e$$

Ini memungkinkan kami untuk menulis:

$$\hat{\hat{S}}_e \bmod p = \left(q \times (q^{-1} \bmod p) \times \hat{V}_p + p \times (p^{-1} \bmod q) \times V_q + k_3 \times p \times q \right) \hat{V}_p + p \times (p^{-1} \bmod q) \times V_q \Big)^e \bmod p$$

$$= \left(q \times (q^{-1} \bmod p) \times \hat{V}_q + k_3 \times p \times q \bmod p \right)^e \bmod p$$

$$= q \times (q^{-1} \bmod p) \times \hat{V}_p \bmod p \qquad (9)$$

- Dengan cara yang sama, seseorang dapat menunjukkannya

$$\hat{\hat{S}}_e \bmod q = \left(p \times (p^{-1} \bmod q) \times V_q \right)^e \bmod q \qquad (10)$$

- Membandingkan hasil dalam Persamaan (6) dan (7) dengan persamaan (4) dan (5), kami mengklaim

$$\hat{\hat{S}}_e \bmod p \quad 6 = \quad M \qquad (11)$$

$$\hat{\hat{S}}_e \bmod q \quad = \quad M \qquad (12)$$

- Persamaan (9) menyiratkan bahwa kita bisa menulis

$$\hat{\hat{S}}_e \quad = M + k_4 \times q \qquad (13)$$

untuk beberapa nilai konstanta k_4. Hubungan ini dapat dinyatakan sebagai

$$\hat{\hat{S}}_e - M = k_4 \times q \qquad (14)$$

- Sejak n = p × q, apa yang kita miliki adalah itu $\hat{S}^{\wedge}_e - M$ dan modulus n berbagi faktor umum, q. Sejak n hanya memiliki dua faktor, p dan q, karena itu kita bisa menulis

$$\text{gcd}\,(\hat{S}^{\wedge}_e - M, n) = q \qquad (15)$$

Kembali ke Daftar Isi

32.6.1 Demonstrasi Injeksi Kesalahan dengan sebuah Python Script

- Tujuan dari peragaan ini adalah untuk mengilustrasikan hal itu ketika Anda salah hitung (sengaja) juga V_p atau V_q pada langkah CRT dari eksponen modular yang diperlukan oleh algoritme RSA, Anda dapat dengan mudah menemukan kunci privat d.

- Dalam skrip Python berikut, baris (1) sampai (18) menunjukkan dua fungsi, `gcd()` dan `MI()` yang Anda lihat sebelumnya di Kuliah 5. Itu `gcd()` adalah algoritma Euclid untuk menghitung pembagi persekutuan terbesar dari dua bilangan bulat. Dan fungsinya `MI()` mengembalikan pembalikan perkalian dari bilangan bulat argumen pertama di ring yang sesuai dengan bilangan bulat argumen kedua.

- Selanjutnya, baris (19) sampai (29) pertama menyatakan dua faktor prima untuk modulus RSA dan kemudian menghitung nilai yang akan digunakan untuk eksponen publik e dan eksponen pribadi d. Seperti yang pembaca ingat dari Bagian 12.2.2 dari Kuliah 12, e harus relatif prima untuk keduanya p - 1 dan q - 1, yang merupakan dua faktor dari penjumlahan n. Evaluasi bersyarat pada baris (25) menjamin itu. Setelah pengaturan e, pernyataan pada baris (28) menetapkan eksponen pribadi d.

- Kode di baris (30) sampai (37) pertama kali menetapkan bilangan bulat pesan M dan kemudian menghitung hasil perantara V. p dan V. q, seperti yang dijelaskan di bagian sebelumnya. Perhatikan bahwa kami menggunakan Fermat Teorema Kecil (lihat Bagian 11.2 dari Kuliah 11) untuk mempercepat perhitungan V. p dan V. q. [Mengingat kecilnya ukuran angka yang terlibat, ada jelas tidak ada alasan khusus untuk menggunakan FLT di sini. Meskipun demikian, pembaca harus memutuskan untuk bermain-main dengan ini demonstrasi menggunakan jumlah besar, menggunakan FLT tentu akan membuat waktu respon lebih cepat dari kode demonstrasi.] Sejalan (36), kami menggunakan teorema CRT untuk menggabungkan nilai untuk V. p dan V. q ke dalam tanda tangan digital berbasis RSA dari bilangan bulat pesan M.

- Terakhir, kode dalam baris (39) hingga (47) adalah demonstrasi injeksi kesalahan dan bagaimana kode tersebut dapat digunakan untuk menemukan faktor prima q dari modulus RSA n. Kami mensimulasikan injeksi kesalahan dengan menambahkan angka acak kecil dengan nilai V. p sejalan (42). Selanjutnya, kami menggunakan Persamaan (10) dari bagian sebelumnya untuk perkirakan nilainya q sejalan (44).

```
#! / usr / bin / env python

# # FaultInjectionDemo.py
# # Avi Kak (30 Maret 2015)

# # Skrip ini mendemonstrasikan eksploitasi injeksi kesalahan pada langkah CRT dari
# # dari algoritme RSA.

# # Kalkulator GCD (Dari Kuliah 5) def gcd (a, b):                                    # (1)
    sedangkan b:                                                                      # (2)
        a, b = b, a% b                                                                # (3)
    kembali a                                                                         # (4)

# # Kode yang ditunjukkan di bawah ini menggunakan implementasi aritmatika integer biasa
# # Algoritma Euclid Diperluas untuk menemukan MI dari integer argumen pertama
# # vis-a-vis bilangan bulat arg kedua. (Segmen kode ini dari Kuliah 5) def MI (num, mod):
                                                                                      # (5)
```

```
    '' '
    Fungsi mengembalikan multiplicative inverse (MI) dari num modulo mod '' '

    NUM = num; MOD = mod                                                    # (6)
    x, x_old = 0L, 1L                                                       # (7)
    y, y_old = 1L, 0L                                                       # (8)
    sedangkan mod:                                                          # (9)
        q = num // mod                                                      # (10)
        num, mod = mod, num% mod                                            # (11)
        x, x_old = x_old - q * x, x                                         # (12)
        y, y_old = y_old - q * y, y                                         # (13)
    jika num! = 1:                                                          # (14)
        naikkan ValueError ("NO MI. Namun, GCD dari% d dan% d adalah% u" \
                                                                % (NUM, MOD, num))  # (15)
    lain:                                                                   # (16)
        MI = (x_old + MOD)% MOD                                             # (17)
        mengembalikan MI                                                    # (18)


# Setel parameter RSA:
p = 211                                                                     # (19)
q = 223                                                                     # (20)
n = p * q                                                                   # (21)
cetak "parameter RSA:"
cetak "p =% d          q =% d          modulus =% d "% (p, q, n)            # (22)
totient_n = (p-1) * (q-1)                                                   # (23)
# Temukan kandidat eksponen publik: untuk e
  dalam rentang (3, n):
                                                                            # (24)
    if (gcd (e, p-1) == 1) dan (gcd (e, q-1) == 1):                         # (25)
        istirahat                                                           # (26)
cetak "eksponen publik e =", e                                              # (27)
# Sekarang atur eksponen privat: d = MI
  (e, totient_n)
                                                                            # (28)
cetak "eksponen pribadi d =", d                                             # (29)


pesan = 6789                                                                # (30)
print "\ nmessage =", pesan                                                 # (31)

# Menerapkan Teorema Sisa Cina untuk menghitung
# pesan ke kekuatan d mod n: dp = d% (p - 1)
                                                                            # (32)
dq = d% (q - 1)                                                             # (33)
V_p = ((pesan% p) ** dp)% p V_q =                                           # (34)
((pesan% q) ** dq)% q                                                       # (35)

tanda tangan = (q * MI (q, p) * V_p          + p * MI (p, q) * V_q)% n      # (36)

print "\ nsignature =", tanda tangan                                        # (37)

impor acak                                                                  # (38)

cetak "\ nESTIMASI q MELALUI KESALAHAN YANG TERSEDIA:" untuk
i dalam kisaran (10):                                                       # (39)
    error = random.randrange (1,10)                                         # (40)
```

```
V_hat_p = V_p + kesalahan                                                    # (42)
cetak "\ nV_p =% d          V_hat_p =% d          error =% d "% (V_p, V_hat_p, error)   # (41)
signature_hat = (q * MI (q, p) * V_hat_p                  + p * MI (p, q) * V_q)% n   # (43)
q_estimate = gcd ((signature_hat ** e - message)% n, n) print                # (44)
"kemungkinan nilai untuk q =", q_estimate                                    # (45)
jika q_estimate == q:                                                        # (46)
        print "Serangan berhasil !!!"                                       # (47)
```

- Di bawah ini adalah output dari skrip. Seperti yang bisa dilihat pembaca, untuk semua nilai kesalahan acak yang ditambahkan ke nilai V_p, kami dapat memperkirakan faktor prima dengan benar q dari RSA modulus.

```
Parameter RSA:
p = 211          q = 223          modulus = 47053
eksponen publik e = 11 eksponen
pribadi d = 21191

pesan = 6789

tanda tangan = 42038

ESTIMASI q MELALUI KESALAHAN YANG TERSEDIA:

V_p = 49         V_hat_p = 56          error = 7
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49         V_hat_p = 55          error = 6
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49         V_hat_p = 53          error = 4
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49         V_hat_p = 52          error = 3
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49         V_hat_p = 54          error = 5
```

```
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49        V_hat_p = 52        error = 3
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49        V_hat_p = 53        error = 4
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49        V_hat_p = 56        error = 7
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49        V_hat_p = 58        error = 9
nilai yang mungkin untuk q = 223
Serangan berhasil !!!

V_p = 49        V_hat_p = 58        error = 9
nilai yang mungkin untuk q = 223
Serangan berhasil !!!
```

- Serangan injeksi kesalahan pertama kali ditemukan oleh Dan Boneh, Richard DeMillo, dan Richard Lipton pada tahun 1997 dan dijelaskan dalam publikasi Journal of Cryptology 2001 mereka "Tentang Pentingnya Menghilangkan Kesalahan dalam Perhitungan Kriptografi." Logika yang digunakan dalam skrip Python yang ditunjukkan di bagian ini didasarkan pada perbaikan serangan asli oleh A.
  K. Lenstra. Perbaikan ini juga disebutkan dalam publikasi Boneh et al.

Kembali ke Daftar Isi

32.7 SERANGAN WAKTU

- Serangan pengaturan waktu didasarkan pada premis bahwa jika Anda dapat memantau berapa lama waktu yang dibutuhkan untuk mengeksekusi segmen tertentu dari rutinitas kriptografi, Anda mungkin dapat membuat tebakan yang baik untuk parameter rahasia algoritme.

- Untuk menguraikannya, mari pertimbangkan algoritma berikut untuk eksponensiasi modular yang Anda lihat sebelumnya di Bagian 12.5.1 Kuliah 12: [Diskusi Fault Injection di Bagian 32.6 dari kuliah saat ini difokuskan pada langkah CRT dari implementasi keseluruhan dari algoritma eksponensial modular. Seperti yang Anda ingat dari Bagian 12.5.1 dari Kuliah 12, setelah Anda melakukan penyederhanaan eksponen modular dengan CRT, Anda masih perlu menghitung kuantitas seperti SEBUAH ^{B} mod n.]

```
hasil = 1
sedangkan B> 0:
    jika B & 1:                                  # (1)
        hasil = (hasil * A)% n B = B >> 1        # (2)

    A = (A * A)% n hasil
pengembalian
```

Seperti yang dijelaskan di Bagian 12.5.1 dari Kuliah 12, algoritma ini melakukan pemindaian eksponen bitwise B dari bit yang paling tidak signifikan hingga bit yang paling signifikan. Ini menghitung kuadrat dari alasnya SEBUAH di setiap langkah pemindaian. Nilai kuadrat ini dikalikan dengan nilai antara untuk hasil hanya jika bit eksponen ditetapkan pada langkah saat ini.

- Sekarang bayangkan bahwa Anda telah memperoleh cara untuk memantau berapa lama waktu yang dibutuhkan untuk mengeksekusi kode pada baris (1) dan (2) yang ditunjukkan di atas. Dengan asumsi bahwa pengukuran waktu Anda cukup akurat, pengukuran waktu ini akan langsung menghasilkan eksponen B. Dan bahkan jika pengukuran waktu Anda tidak begitu andal, mungkin Anda dapat melakukan operasi eksponensial berulang kali dan kemudian menghitung rata-rata kebisingannya. Ini adalah persis dasar untuk demonstrasi dalam skrip Python yang ditampilkan selanjutnya.

- Jelas, reaksi pertama Anda terhadap klaim yang dibuat di atas adalah: Bagaimana Anda masuk ke dalam perangkat keras perangkat seluler untuk memantau waktu eksekusi segmen kode untuk menyimpulkan rahasianya melalui waktu yang dibutuhkan oleh bagian-bagian kode tersebut? Dalam hampir semua situasi, hal yang paling bisa dilakukan penyerang adalah memasukkan pesan yang berbeda ke dalam perangkat seluler dan mengukur total waktu yang dibutuhkan oleh algoritme untuk setiap pesan tersebut. Selanjutnya, jika memungkinkan, penyerang perlu menyimpulkan rahasia dari saat-saat itu.

- Tujuan dari sub-bagian berikut ini adalah untuk menunjukkannya aku s mungkin untuk menentukan kunci enkripsi dari waktu keseluruhan diambil oleh algoritme untuk setiap kumpulan besar pesan yang dibuat secara acak.

- Namun, tujuan dari bagian saat ini hanyalah untuk memusatkan perhatian

menunjukkan bagaimana seseorang dapat mengukur waktu eksekusi yang terkait dengan fragmen kode dan rata-rata yang diperlukan untuk mengurangi efek noise yang terkait dengan pengukuran tersebut.

- Sekarang mari kita bahas pertanyaan tentang bagaimana seseorang dapat mengukur waktu yang terkait dengan eksekusi seluruh algoritme, atau hanya dengan sebagian kecil kode, dan mengapa pengukuran seperti itu secara inheren berisik. Anda dapat mencoba mengukur waktu eksekusi dengan mengambil selisih waktu jam dinding sebelum masuk ke dalam segmen kode dan setelah keluar dari segmen kode tersebut. Perkiraan semacam itu pasti hanya merupakan perkiraan waktu sebenarnya yang dihabiskan di prosesor oleh segmen kode itu. Anda lihat, pada saat tertentu, mungkin ada puluhan, jika tidak ratusan, proses dan utas yang berjalan "secara bersamaan" di komputer Anda. Dengan asumsi demi argumen bahwa Anda memiliki prosesor inti tunggal, artinya semua proses dan utas dipotong waktu berkaitan dengan aksesnya ke CPU. Yaitu, proses atau utas yang saat ini sedang dieksekusi di CPU diluncurkan dan statusnya disimpan dalam memori ketika kuantum waktu yang diizinkan untuk dieksekusi kedaluwarsa. Selanjutnya, salah satu proses atau utas menunggu digulung ke dalam CPU, dan seterusnya. [Semuanya modern

  sistem operasi memelihara beberapa antrian untuk eksekusi bersamaan dari beberapa proses dan utas.

  Misalnya, ada antrian proses yang menunggu giliran di CPU. Haruskah proses itu

  sedang dijalankan oleh CPU membutuhkan akses ke perangkat I / O tertentu, itu diambil dari CPU dan

  ditempatkan dalam antrian untuk perangkat I / O tersebut. Setelah selesai dengan I / O, ini kembali ke antrian proses

  menunggu giliran mereka di CPU. Kecuali jika proses diambil dari CPU karena alasan I / O, atau karena alasan tersebut

terputus, dll., yang lebih umum, proses diambil dari CPU karena pembagian waktu yang dialokasikan di CPU telah kedaluwarsa. Dalam sistem Unix / Linux, ada proses khusus PID 0 yang bertindak sebagai prosesor penjadwal. Tugas penjadwal adalah untuk mencari tahu proses menunggu mana yang mendapat giliran di CPU.]

- Untuk mendemonstrasikan seberapa bising pengukuran running time, berikut adalah 10 percobaan dari algoritma yang sama yang terdiri dari 16 langkah. Waktu eksekusi setiap langkah diukur sebagai selisih antara waktu jam dinding sebelum dan sesudah eksekusi segmen kode yang sesuai dengan langkah tersebut. Bahwa beberapa entri adalah '0,0' tidak mengherankan karena 12 dari 16 langkah pada dasarnya adalah langkah tidak melakukan apa-apa. Namun, empat sisanya memang membutuhkan perkalian yang besar. Empat langkah yang melibatkan perkalian besar berada di langkah pertama, kedelapan, kesepuluh, dan keenam belas.

1: [5.96e-06, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 9.53e-07, 0.0, 0.0]

2: [5.96e-06, 9.53e-07, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0,0, 0,0]

3: [5.96e-06, 9.53e-07, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 1.19 e-06]

4: [5.96e-06, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 1.19e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.19e-06, 0.0]

5: [5.96e-06, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 9.53e-07, 1.19e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

6: [5.96e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 1.19e-06, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 9.53e-07, 0.0, 0.0, 0.0]

7: [5.96e-06, 9.53e-07, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 9.53e-07, 0.0, 9.53e-07, 0,0, 0,0]

8: [5.96e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

9: [8.10e-06, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 1.19e-06, 0.0, 0.0, 9.53e-07]

10: [6.19e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 9.53e-07, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

- Tujuan kami di bagian ini adalah untuk mengilustrasikan cara memulihkan nilai

dari eksponen B dalam perhitungan SEBUAH B mod n dari waktu eksekusi yang berisik dari satu langkah kunci dalam algoritma eksponensial modular yang ditunjukkan di bagian sebelumnya.

- Dalam skrip berikutnya, baris (1) hingga (12) mendefinisikan algoritme eksponensial modular yang sama dengan yang Anda lihat sebelumnya - kecuali untuk pernyataan pengukuran waktu yang diselingi. Pernyataan pengukuran waktu ada di baris (2), (5), (8), dan (9). Kami ingin mengukur waktu yang diperlukan untuk menghitung langkah perkalian pada baris (7) dengan menyadari bahwa ini
  perkalian berlangsung sesuai dengan kondisi pada baris (6). Jumlah iterasi dalam sementara loop yang dimulai pada baris (4) sama dengan jumlah bit dalam representasi biner dari eksponen B.

- Selanjutnya, untuk mengatasi gangguan dalam pengukuran waktu eksekusi seperti yang ditunjukkan di bagian sebelumnya, kami definisikan fungsinya pengukuran waktu berulang () dalam garis (13) sampai (20). Semua yang dilakukan fungsi ini adalah memanggil fungsi eksponensial modular berulang kali. Ini yang ketiga
  argumen untuk pengukuran waktu berulang () bahwa menentukan berapa kali fungsi eksponensial modular akan dipanggil. Pengukuran waktu dalam setiap panggilan ke eksponen modular disimpan dalam daftar daftar yang terikat ke
  variabel daftar jejak waktu.

- Di baris (21) hingga (24), kami kemudian menetapkan nilai untuk basis SEBUAH,

eksponen B, modulus n, dan jumlah pengulangan untuk memanggil fungsi eksponen modular. Sejalan (25), kami lalu panggil pengukuran waktu berulang () dengan nilai-nilai ini.

- Sisa kode, dalam baris (26) sampai (38), pertama-tama digunakan untuk rata-rata pengukuran waktu bising untuk masing-masing langkah, menemukan ambang sebagai titik tengah antara pengukuran waktu minimum dan maksimum, thresholding pengukuran waktu, dan membangun string bit dari 0 dan 1 sehingga diperoleh.

```
#! / usr / bin / env python

# # EstimatingExponentFromExecutionTime.py

# # Avi Kak ( kak@purdue.edu )
# # 31 Maret 2015

# # Skrip ini mendemonstrasikan ide dasar tentang bagaimana mungkin untuk menyimpulkan
# # bidang bit eksponen dengan mengukur waktu yang dibutuhkan untuk membawa
# # keluar salah satu langkah kunci dalam algoritme eksponensial modular.

waktu impor

# # Ini adalah skrip dasar kami untuk eksponensial modular.          Lihat Bagian 12.5.1 dari
# # Kuliah 12:
def modular_exponentiate (A, B, modulus):                                          # (1)
    time_trace = []                                                                # (2)
    hasil = 1                                                                      # (3)
    sedangkan B> 0:                                                                # (4)
        mulai = waktu.waktu ()                                                     # (5)
        jika B & 1:                                                                # (6)
            result = (result * A)% modulus elapsed =                               # (7)
        time.time () - start time_trace.append (berlalu)                           # (8)
                                                                                   # (9)
        B = B >> 1                                                                 # (10)
        A = (A * A)% hasil modulus                                                 # (11)
    return, time_trace                                                             # (12)

# # Karena satu percobaan tidak menghasilkan pengukuran waktu yang dapat diandalkan
# # diambil oleh langkah komputasi, fungsi ini membantu kita melakukan pengulangan
# # percobaan:
```

```
def repeat_time_measurements (A, B, modulus, how_many_times):                    # (13)
        list_of_time_traces = []                                                  # (14)
        hasil = []                                                                # (15)
        untuk saya dalam jangkauan (how_many_times):                              # (16)
                hasil, schedulerace = modular_exponentiate (A, B, modulus)        # (17)
                list_of_time_traces.append (schedulerace)                         # (18)
                results.append (hasil)                                            # (19)
        # Juga kembalikan 'hasil' untuk debugging, dll. Return
          list_of_time_traces, results                                            # (20)

A = 12345678901234567890123456789012345678901234567890123456789012345678901234567890 B =  # (21)
0b1111110101001001
modulus = 987654321                                                               # (23)
jumlah_iterasi = 1000                                                             # (24)

list_of_time_traces, results = repeat_time_measurements (A, B, modulus, num_iterations) # (25)

jumlah = [jumlah (e) untuk e dalam zip (* list_of_time_traces)]                   # (26)
rata-rata = [x / num_iterasi untuk x dalam jumlah] rata-rata =                    # (27)
daftar (dibalik (rata-rata))                                                      # (28)
cetak "\ nwaktu:", rata-rata                                                      # (29)
minval, maxval = min (rata-rata), max (rata-rata) ambang =                        # (30)
(maxval - minval) / 2                                                             # (31)
bitstring = ''                                                                    # (32)
untuk item rata-rata:                                                             # (33)
        jika item> ambang batas:                                                  # (34)
                bitstring + = '1'                                                 # (35)
        lain:                                                                     # (36)
                bitstring + = '0'                                                 # (37)
print "\ nbitstring untuk B dibuat dari pengaturan waktu:", bitstring             # (38)
```

- Jika Anda menjalankan skrip Python seperti yang ditunjukkan di atas, ia mengeluarkan string bit:

  1111110101001001

  yang sama dengan pola bit untuk eksponen dalam baris (22). Anda dapat menjalankan eksperimen yang sama dengan pilihan eksponen lainnya B sejalan (22). Sebagai contoh, jika saya mengubah baris itu menjadi B = 0b1100110101110101, jawaban yang dikembalikan oleh skrip adalah 1100110101110101, dan seterusnya.

- Ini membuktikan bahwa, meskipun pada dasarnya berisik

pengukuran waktu, Anda dapat menghitung nilai eksponen dalam eksponen modular yang diperlukan untuk kalkulasi kriptografi hanya dengan mengukur berapa lama waktu yang dibutuhkan untuk mengeksekusi salah satu langkah kunci dari algoritme.

- Bahwa dimungkinkan untuk memasang serangan waktu pada rutinitas kriptografi pertama kali diduga oleh Paul Kocher pada tahun 1996 dalam sebuah makalah berjudul "Serangan Waktu pada Implementasi Di ffi e-Hellman, RSA, DSS, dan Sistem Lain," yang muncul di CRYPTO ' 96, Catatan Kuliah dalam Ilmu Komputer, Vol. 1355.

- Dengan mengingat pertimbangan ini, subbagian berikutnya mendemonstrasikan elemen dasar dari Timing Attack dengan bantuan skrip Python.

Kembali ke Daftar Isi

32.7.1 Skrip Python Yang Mendemonstrasikan Cara Menggunakan Waktu Eksekusi Kode untuk Memasang Timing Attack

- Sekarang mari kita bicara tentang bagaimana sebenarnya memasang serangan waktu menggunakan waktu yang diperlukan untuk sepenuhnya menghitung tanda tangan RSA untuk kumpulan pesan yang dibuat secara acak. Dengan kata lain, kami tidak akan lagi berasumsi bahwa kami dapat mengukur waktu yang dibutuhkan oleh langkah-langkah individual dari algoritme eksponensial modular.

- Saat ini, untuk upaya serius dalam memasang serangan waktu, kita perlu menerapkannya dengan cara yang dijelaskan dalam makalah "Penerapan Praktis dari Serangan Waktu" oleh Jean-Francois Dhem, Francois Koeune, Philippe- Alexandre Leroux, Patrick Mestre, Jean-Jacques Quisquater, dan Jean-Louis Willems yang muncul dalam Proceedings of the International Conference on Smart Cards

  dan Applications, 1998, hlm. 167-182. [Ini adalah pendekatan probabilistik yang mencakup: (1) memindai posisi bit dalam kunci enkripsi dari kanan ke kiri; (2) membentuk dua hipotesis di setiap bit posisi, satu untuk bit menjadi 0 dan yang lainnya untuk bit menjadi 1; (3) Menemukan dukungan probabilistik untuk masing-masing hipotesis dengan mempertimbangkan bit yang ditemukan sejauh ini, dan perbedaan antara ukuran pesan populasi di bawah dua hipotesis (keanggotaan dalam populasi memperhitungkan fakta bahwa hipotesis yang meminta bit menjadi 1 akan memerlukan komputasi yang sedikit lebih lama).] Menggunakan ini

pendekatan, penulis mampu memecahkan kunci 512-bit dalam beberapa menit menggunakan 300.000 pengukuran waktu.

- Tujuan saya di bagian ini bukan untuk mereplikasi pekerjaan yang dijelaskan dalam publikasi yang dikutip di atas. Di sisi lain, yang ingin saya tunjukkan adalah bahwa ada korelasi antara pengukuran waktu untuk eksponen modular untuk kumpulan pesan yang dibuat secara acak dan waktu yang diukur untuk eksponen yang sama di bawah hipotesis bahwa bit tertentu dalam eksponen adalah 1 atau 0. Selanjutnya, korelasi ini dapat dimanfaatkan untuk membuat tebakan untuk bit individu eksponen.

- Untuk membingkai masalah yang coba dipecahkan oleh skrip Python di bagian ini dengan implementasi mainan, mari kita kembali ke kasus perangkat yang menggunakan algoritme RSA untuk menandatangani pesan keluar secara digital. Seperti yang dinyatakan sebelumnya, diberi pesan M dan eksponen pribadi d, perangkat ini harus menghitung $M^d \bmod n$ dimana n adalah modulus RSA. Sebelumnya, di Bagian 32.6.1, kita melihat bagaimana langkah CRT yang digunakan untuk menyederhanakan eksponen modular dapat terkena injeksi kesalahan untuk menemukan nilai kunci privat.

- Kami sekarang akan berasumsi bahwa kami secara langsung menghitung eksponensial modular $M^d \bmod n$ diperlukan untuk menandatangani pesan secara digital M dengan eksponen pribadi d. Tujuan kami adalah menemukan d hanya dari waktu yang dibutuhkan untuk menghitung tanda tangan untuk sebuah

kumpulan pesan yang sewenang-wenang. Seperti yang akan selalu terjadi, kami akan berasumsi demikian d ganjil dan, oleh karena itu, bitnya yang paling tidak signifikan selalu 1. Tujuan kita adalah menemukan bit lainnya.

- Logika keseluruhan dari skrip ini adalah memperkirakan bit eksponen privat d, satu bit pada satu waktu, mulai dari bit yang paling tidak signifikan (yang, seperti telah disebutkan, adalah 1). Estimasi didasarkan pada menemukan korelasi antara waktu yang dibutuhkan untuk menghitung tanda tangan dalam dua kondisi: ketika bit yang akan diestimasi dapat diasumsikan menjadi 0 dan ketika dapat diasumsikan sebagai 1. Di bawah setiap hipotesis, korelasinya adalah dengan pengukuran waktu untuk perhitungan tanda tangan yang sebenarnya. Kami menyatakan nilai untuk bit berikutnya atas dasar korelasinya lebih besar.

- Pekerja keras dalam skrip berikut ini adalah metode temukan bit kunci pribadi berikutnya (). Dua blok utamanya adalah di garis (F9) sampai (F25) dan di baris (F33) sampai (F46). Di blok pertama, dalam baris (F9) hingga (F25), fungsi ini menghitung korelasi untuk kasus ketika kita mengasumsikan 0 untuk posisi bit berikutnya dalam eksponen privat d. Pada blok kedua, di baris (F33) hingga (F46), kami menghitung korelasi yang serupa untuk kasus ketika kami mengasumsikan bit berikutnya adalah 1. Kedua nilai korelasi dibandingkan dalam baris (F47).

- Anda harus menggunakan sejumlah besar bilangan bulat pesan agar serangan dapat bekerja sampai batas tertentu. Seperti yang akan Anda perhatikan dari

konstruktor memanggil baris (A1) hingga (A6), percobaan saya sendiri dengan skrip ini biasanya melibatkan 100.000 bilangan bulat pesan.

- Anda mungkin berpikir bahwa dalam beberapa rangkaian serangan keseluruhan di baris (A9) hingga (A25), kami dapat mempercepat keseluruhan waktu yang diambil oleh skrip dengan melakukan panggilan yang menghasilkan sejumlah besar pesan dalam baris (A11) di luar lingkaran. Perhatikan bahwa waktu yang dibutuhkan untuk menghasilkan 100.000 pesan adalah sebagian kecil dari waktu yang dibutuhkan oleh eksponen modular dari pesan-pesan tersebut melalui kode dalam baris (X1) hingga (X11).

- Kamus terikat ke variabel contoh korelasi cache in line (J15) digunakan di
  temukan bit kunci pribadi berikutnya () dalam garis (F8) dan (F30). Kamus ini membantu menghindari duplikasi kalkulasi korelasi untuk nilai eksponen privat yang sama.

```
#! / usr / bin / env python

# # TimingAttack.py

# # Avi Kak ( kak@purdue.edu )
# # 13 April 2015

# #    Skrip ini menunjukkan ide dasar tentang bagaimana Timing Attack dapat digunakan untuk
# #    menyimpulkan bit eksponen privat yang digunakan dalam menghitung tanda tangan digital
# #    berbasis RSA.
# #
# #    PERHATIAN: Implementasi sederhana ini didasarkan pada satu kemungkinan
# #                    interpretasi makalah asli tentang serangan waktu oleh Paul Kocher. Perhatikan
# #                    bahwa implementasi ini hanya dicoba pada modulus 8-bit.
# #
# #
# #                    Saya cukup yakin bahwa implementasi yang sangat sederhana ini
```

```
# #               TIDAK akan bekerja pada modulus RSA dari ukuran yang sebenarnya digunakan dalam
# #               algoritma kerja.
# #
# #               Untuk serangan waktu yang lebih kredibel, Anda perlu memasukkan dalam
# #               implementasi ini logika probabilistik yang dijelaskan dalam makalah "Implementasi
# #               Praktis dari Serangan Waktu '' oleh Dhem, Koeune, Leroux, Mestre, Quisquater,
# #               dan Willems.

waktu impor
impor acak
impor matematika

kelas TimingAttack (object):                                                                    # (I1)

    def __init __ (diri, ** kwargs):                                                            # (J2)
        if kwargs.has_key ('num_messages'): num_messages = kwargs.pop ('num_messages') if       # (J3)
        kwargs.has_key ('num_trials'): num_trials = kwargs.pop ('num_trials')                   # (J3)
        jika kwargs.has_key ('private_exponent'): private_exponent = kwargs.pop ('private_exponent')
                                                                                                # (J4)
        jika kwargs.has_key ('modulus_width'): modulus_width = kwargs.pop ('modulus_width') # (J5) self.num_messages =
        num_messages                                                                            # (J6)
        self.num_trials = num_trials                                                            # (J7)
        self.modulus_width = modulus_width                                                      # (J8)
        self.d = private_exponent                                                               # (J9)
        self.d_reversed = '{: b}'. format (private_exponent) [:: - 1]                           # (J10)
        self.modulus = Tidak ada                                                                # (J11)
        self.list_of_messages = []                                                              # (J12)
        self.times_taken_for_messages = []                                                      # (J13)
        self.bits_discovered_for_d = []                                                         # (J14)
        self.correlations_cache = {}                                                            # (J15)

    def gen_modulus (diri):                                                                     # (G1)
        modulus = self.gen_random_num_of_specified_width (self.modulus_width / 2) * \
                                self.gen_random_num_of_specified_width (self.modulus_width / 2) # (G2)
        print "modulus is:", modulus                                                            # (G3)
        self.modulus = modulus                                                                  # (G4)
        modulus kembali                                                                         # (G5)


    def gen_random_num_of_specified_width (diri, lebar):                                        # (R1)
        ''  '
        Fungsi ini menghasilkan nomor acak dari lebar bidang bit yang ditentukan: '' '

        kandidat = random.getrandbits (lebar) jika                                              # (R2)
        kandidat & 1 == 0: kandidat + = 1 kandidat | = (1 <<                                    # (R3)
        lebar - 1)                                                                              # (R4)
        kandidat | = (2 << width - 3) kandidat                                                  # (R5)
        kembali                                                                                 # (R6)

    def modular_exponentiate (self, A, B):                                                      # (X1)
        ''  '
        Ini adalah fungsi dasar kami untuk eksponen modular seperti yang dijelaskan di Bagian
        12.5.1 dari Kuliah 12:
        ''  '
        jika self.modulus adalah None:                                                          # (X2)
```

```
            naikkan SyntaxError ("Anda harus terlebih dahulu menyetel                    # (X3)
        modulus") time_trace = []                                                        # (X4)
        hasil = 1                                                                        # (X5)
        sedangkan B> 0:                                                                  # (X6)
            jika B & 1:                                                                  # (X7)
                hasil = (hasil * A)% self.modulus B = B >> 1                             # (X8)
                                                                                         # (X9)
            A = (A * A)% self.modulus hasil                                             # (X10)
        pengembalian                                                                    # (X11)

def berkorelasi (self, series1, series2):                                               # (C1)
        jika len (seri1)! = len (seri2):                                                # (C2)
            meningkatkan ValueError ("kedua rangkaian harus memiliki panjang yang sama") # (C3)
        mean1, mean2 = sum (series1) / float (len (series1)), sum (series2) / float (len (series2)) # (C4) mseries1, mseries2 =
        [x - mean1 untuk x dalam seri1], [x - mean2 untuk x dalam seri2] # (C5) produk = [mseries1 [i] * mseries2 [i] untuk
        i dalam rentang (len (mseries1))]                                               # (C6)
        mseries1_squared, mseries2_squared = [x ** 2 untuk x di mseries1], [x ** 2 untuk x di mseries2]
                                                                                        # (C7)
        korelasi = jumlah (produk) / math.sqrt (sum (mseries1_squared) * sum (mseries2_squared))
                                                                                        # (C8)
        korelasi pengembalian                                                           # (C9)

def gen_messages (self):                                                                # (M1)
        '' '
        Buat daftar pesan yang dibuat secara acak. kendala pada      Pesan-pesan itu harus mematuhi yang biasa
        dua bit paling signifikan: '' '

        self.correlations_cache = {}                                                    # (M2)
        self.times_taken_for_messages = []                                              # (M3)
        self.list_of_messages = []                                                      # (M4)
        untuk saya dalam jangkauan (self.num_messages):                                 # (M5)
            message = self.gen_random_num_of_specified_width (self.modulus_width)       # (M6)
            self.list_of_messages.append (pesan)                                        # (M7)
        print "Selesai menghasilkan% d pesan"% (self.num_messages)                      # (M8)

def get_exponentiation_times_for_messages (sendiri):                                    # (T1)
        '' '
        Untuk setiap pesan di list_of_messages, temukan waktu yang dibutuhkan untuk menghitung tanda
        tangannya. Rata-rata setiap kali pengukuran selama num_trials:
        '' '
        jika self.modulus adalah None:                                                  # (T2)
            meningkatkan SyntaxError ("Anda harus terlebih dahulu mengatur              # (T3)
        modulus") untuk pesan di self.list_of_messages:                                 # (T4)
            times = []                                                                  # (T5)
            untuk j dalam rentang (self.num_trials):                                    # (T6)
                mulai = waktu.waktu ()                                                  # (T7)
                self.modular_exponentiate (pesan, self.d)                               # (T8)
                elapsed = time.time () - waktu                                          # (T9)
                mulai.append (berlalu)                                                 # (T10)
            avg = sum (times) / float (len (times))                                    # (T11)
            self.times_taken_for_messages.append (avg)                                 # (T12)
        print "Selesai menghitung tanda tangan untuk semua pesan"                      # (T13)

def find_next_bit_of_private_key (self, list_of_previous_bits):                         # (F1)
        '' '
```

```
Dimulai dengan LSB, diberi urutan bit eksponen pribadi d yang dihitung sebelumnya, sekarang
hitung bit berikutnya:
''
num_set_bits = kurangi (lambda x, y: x + y, \
                                    filter (lambda x: x == 1, list_of_previous_bits))          # (F2)
korelasi0, korelasi1 = Tidak ada, Tidak ada                                                    # (F3)
arg_list1, arg_list2 = list_of_previous_bits [:], list_of_previous_bits [:] B = int (''. join (map (str, list  # (F4)
(reversed (arg_list1)))), 2)                                                                   # (F5)
cetak "\ nB =", B                                                                              # (F6)
jika B di self.correlations_cache:                                                             # (F7)
    korelasi0 = self.correlations_cache [B]                                                    # (F8)
lain:                                                                                          # (F9)
    times_for_p Partial_exponentiation = []                                                    # (F10)
    untuk pesan di self.list_of_messages:                                                      # (F11)
        tanda tangan = Tidak ada                                                               # (F12)
        times = []                                                                             # (F13)
        untuk j dalam rentang (self.num_trials):                                               # (F14)
            mulai = waktu.waktu ()                                                             # (F15)
            self.modular_exponentiate (pesan, B)                                               # (F16)
            elapsed = time.time () - waktu                                                     # (F17)
            mulai.append (berlalu)                                                             # (F18)
        avg = sum (times) / float (len (times)) times_for_p                                    # (F19)
        Partial_exponentiation.append (avg)                                                    # (F20)
    korelasi0 = self.correlate (self.times_taken_for_messages, \
                                            times_for_pihak_exponentiation)                    # (F22)
    korelasi0 / = num_set_bits                                                                 # (F23)
    self.correlations_cache [B] = korelasi0                                                    # (F24)
print "korelasi0:", korelasi0                                                                  # (F25)
# Sekarang mari kita lihat korelasinya ketika kita mencoba 1 untuk bit berikutnya
  arg_list2.append (1)
                                                                                               # (F26)
B = int (''. Join (map (str, list (reversed (arg_list2))))), 2) print" B = ", B                # (F27)
                                                                                               # (F28)
jika B di self.correlations_cache:                                                             # (F29)
    korelasi1 = self.correlations_cache [B]                                                    # (F30)
lain:                                                                                          # (F31)
    times_for_p Partial_exponentiation = []                                                    # (F32)
    untuk pesan di self.list_of_messages:                                                      # (F33)
        tanda tangan = Tidak ada                                                               # (F34)
        times = []                                                                             # (F35)
        untuk j dalam rentang (self.num_trials):                                               # (F36)
            mulai = waktu.waktu ()                                                             # (F37)
            self.modular_exponentiate (pesan, B)                                               # (F38)
            elapsed = time.time () - waktu                                                     # (F39)
            mulai.append (berlalu)                                                             # (F40)
        avg = sum (times) / float (len (times)) times_for_p                                    # (F41)
        Partial_exponentiation.append (avg)                                                    # (F42)
    korelasi1 = self.correlate (self.times_taken_for_messages, \
                                              times_for_pihak_exponentiation)                  # (F43)
    korelasi1 / = (num_set_bits + 1) self.correlations_cache                                   # (F44)
    [B] = korelasi1                                                                            # (F45)
cetak "korelasi1:", korelasi1 jika korelasi1>                                                  # (F46)
korelasi0:                                                                                     # (F47)
    return 1                                                                                   # (F48)
lain:                                                                                          # (F49)
    return 0                                                                                   # (F50)
```

```
    def discover_private_exponent_bits (sendiri):                                    # (D1)
        ''
        Asumsikan bahwa eksponen privat akan selalu ganjil dan, oleh karena itu, LSB-nya akan selalu 1.
        Sekarang coba temukan bit lainnya.
        ''
        found_bits = [1]                                                             # (D2)
        untuk bitpos dalam kisaran (1,    self.modulus_width):                       # (D3)
            nextbit = self.find_next_bit_of_private_key (found_bits) print "nilai bit # (D4)
            berikutnya:", nextbit                                                    # (D5)
            print "nilainya harus:", self.d_reversed [bitpos] if nextbit! = int      # (D6)
            (self.d_reversed [bitpos]):                                              # (D7)
                meningkatkan ValueError ("Hasil salah untuk bit pada indeks% d"% bitpos) # (D8)
            found_bits.append (nextbit)                                              # (D9)
            print "bit yang ditemukan:", found_bits                                 # (D10)
        self.bits_discovered_for_d = found_bits                                      # (D11)
        kembali found_bits                                                           # (D12)

jika __name__ == '__main__':

    private_exponent = 0b11001011                                                    # (A1)
    timing_attack = TimingAttack (                                                   # (A2)
                                num_messages = 100000,                               # (A3)
                                jumlah_trials = 1000,                                # (A4)
                                modulus_width = 8,                                   # (A5)
                                private_exponent = private_exponent,                 # (A6)
                            )
    modulus_to_discovered_bits = {}                                                  # (A7)
    untuk saya dalam jangkauan (10):                                                 # (A8)
        cetak "\ n \ n ============ Memulai proses% d dari keseluruhan percobaan ============== \ n"% i
                                                                                     # (A9)
        found_bits = []                                                             # (A10)
        timing_attack.gen_messages ()                                               # (A11)
        modulus = timing_attack.gen_modulus ()                                      # (A12)
        timing_attack.get_exponentiation_times_for_messages ()                      # (A13)
        mencoba:                                                                    # (A14)
            found_bits = timing_attack.discover_private_exponent_bits () kecuali ValueError, e: # (A15)
                                                                                    # (A16)
            print "pengecualian tertangkap di main:", ee =                          # (A17)
            str (e) .strip ()                                                       # (A18)
            jika e [-1] .isdigit ():                                                # (A19)
                pos = int (e.split () [- 1])                                        # (A20)
                cetak "\ n                                    Mendapat% d bit !!! "% pos # (A21)
            terus                                                                   # (A22)
        jika ditemukan_bits:                                                        # (A23)
            modulus_to_discovered_bits [i] = \
                        (modulus, ".join (map (str, list (reversed (found_bits))))) # (A24)
        cetak "\ n                                                  KEBERHASILAN!!!!!!!" # (A25)
```

- Di bawah ini adalah output dari satu sesi dengan kode yang ditunjukkan di atas. Perhatikan bahwa, bahkan untuk modulus yang sama, file

hasil akan bervariasi dari satu proses ke proses lainnya karena pesan dibuat secara acak untuk setiap proses.

- Dalam 10 proses kode yang outputnya ditampilkan di bawah ini, tiga proses berhasil menemukan dengan benar enam dari delapan bit eksponen d. Sesekali, Anda akan melihat bahwa seluruh eksponen diestimasi dengan benar oleh kode.

```
============ Memulai proses 0 dari keseluruhan eksperimen =============

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0:              0,00535503170757
B = 3
korelasi1:              0.11955357822
nilai selanjutnya    bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0:              0.11955357822
B = 7
korelasi1:              0,146688433404
nilai selanjutnya    bit: 1
nilainya harus: 0
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 2

                                                Punya 2 bit !!!


============ Memulai proses 1 dari keseluruhan eksperimen =============

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0:              0,00658805175542
B = 3
korelasi1:              0.144786607015
nilai selanjutnya    bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]
```

B = 3
korelasi0: 0.144786607015
B = 7
korelasi1: 0.191148434475
nilai selanjutnya bit: 1
nilainya harus: 0
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 2

Punya 2 bit !!!

========== Memulai proses 2 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,0111837174243
B = 3
korelasi1: 0.146686335386
nilai selanjutnya bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0.146686335386
B = 7
korelasi1: 0,0666330591075
nilai selanjutnya bit: 0
nilainya harus: 0 bit yang
ditemukan: [1, 1, 0]

B = 3
korelasi0: 0.146686335386
B = 11
korelasi1: 0.166780797308
nilai selanjutnya bit: 1
nilainya harus: 1
bit yang ditemukan: [1, 1, 0, 1]

B = 11
korelasi0: 0.166780797308
B = 27
korelasi1: 0.143863234986
nilai selanjutnya bit: 0
nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0]

B = 11
korelasi0: 0,166780797308
B = 43
korelasi1: 0,161661497094
nilai bit berikutnya: 0

nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0, 0]

B = 11
korelasi0: 0.166780797308
B = 75
korelasi1: 0.140458705926
nilai selanjutnya bit: 0
nilainya harus: 1
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 6

Punya 6 bit !!!

========== Memulai proses 3 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 message
modulus adalah: 225
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,0069115683713
B = 3
korelasi1: 0,351567105915
nilai selanjutnya bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0,351567105915
B = 7
korelasi1: 0.268789028694
nilai selanjutnya bit: 0
nilainya harus: 0 bit yang
ditemukan: [1, 1, 0]

B = 3
korelasi0: 0,351567105915
B = 11
korelasi1: 0.285057307844
nilai selanjutnya bit: 0
nilainya harus: 1
pengecualian tertangkap di utama: Hasil yang salah untuk bit pada indeks 3

Punya 3 bit !!!

========== Memulai proses 4 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,00241843558209

B = 3
korelasi1: 0,186079682903
nilai bit berikutnya: 1 nilainya
harus: 1 bit yang ditemukan: [1,
1]

B = 3
korelasi0: 0.186079682903
B = 7
korelasi1: 0,204226222605
nilai selanjutnya bit: 1
nilainya harus: 0
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 2

Punya 2 bit !!!

========== Memulai proses 5 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 modulus pesan
adalah: 169
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,0184536640473
B = 3
korelasi1: 0,217174073139
nilai selanjutnya bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0,217174073139
B = 7
korelasi1: 0.202723379241
nilai selanjutnya bit: 0
nilainya harus: 0 bit yang
ditemukan: [1, 1, 0]

B = 3
korelasi0: 0,217174073139
B = 11
korelasi1: 0.241820663832
nilai selanjutnya bit: 1
nilainya harus: 1
bit yang ditemukan: [1, 1, 0, 1]

B = 11
korelasi0: 0.241820663832
B = 27
korelasi1: 0.192410585206
nilai selanjutnya bit: 0
nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0]

B = 11
korelasi0: 0.241820663832
B = 43
korelasi1: 0.189418029495
nilai selanjutnya bit: 0
nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0, 0]

B = 11
korelasi0: 0.241820663832
B = 75
korelasi1: 0.175041915625
nilai selanjutnya bit: 0
nilainya harus: 1
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 6

Punya 6 bit !!!

========== Memulai proses 6 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,00865525117668
B = 3
korelasi1: 0.177818285803
nilai selanjutnya bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0.177818285803
B = 7
korelasi1: 0.194471520198
nilai selanjutnya bit: 1
nilainya harus: 0
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 2

Punya 2 bit !!!

========== Memulai proses 7 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 message
modulus adalah: 225
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,000834328683801
B = 3
korelasi1: 0,296449299753
nilai bit berikutnya: 1

nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0,296449299753
B = 7
korelasi1: 0,268359146286
nilai selanjutnya bit: 0
nilainya harus: 0 bit yang
ditemukan: [1, 1, 0]

B = 3
korelasi0: 0,296449299753
B = 11
korelasi1: 0,200498385434
nilai selanjutnya bit: 0
nilainya harus: 1
pengecualian tertangkap di utama: Hasil yang salah untuk bit pada indeks 3

Punya 3 bit !!!

========== Memulai run 8 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 modulus pesan
adalah: 195
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: 0,0099350807053
B = 3
korelasi1: 0.100855277594
nilai selanjutnya bit: 1
nilainya harus: 1 bit yang
ditemukan: [1, 1]

B = 3
korelasi0: 0.100855277594
B = 7
korelasi1: 0.123326809251
nilai selanjutnya bit: 1
nilainya harus: 0
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 2

Punya 2 bit !!!

========== Memulai proses 9 dari keseluruhan eksperimen ==========

Selesai menghasilkan 100000 message
modulus adalah: 225
Selesai menghitung tanda tangan untuk semua pesan

B = 1
korelasi0: -0,00389727670499

B = 3
korelasi1: 0,251815183197
nilai bit berikutnya: 1 nilainya
harus: 1 bit yang ditemukan: [1,
1]

B = 3
korelasi0: 0.251815183197
B = 7
korelasi1: 0,224629240235
nilai selanjutnya bit: 0
nilainya harus: 0 bit yang
ditemukan: [1, 1, 0]

B = 3
korelasi0: 0.251815183197
B = 11
korelasi1: 0,253504735599
nilai selanjutnya bit: 1
nilainya harus: 1
bit yang ditemukan: [1, 1, 0, 1]

B = 11
korelasi0: 0,253504735599
B = 27
korelasi1: 0.205049470386
nilai selanjutnya bit: 0
nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0]

B = 11
korelasi0: 0,253504735599
B = 43
korelasi1: 0.186280401626
nilai selanjutnya bit: 0
nilainya harus: 0
bit yang ditemukan: [1, 1, 0, 1, 0, 0]

B = 11
korelasi0: 0,253504735599
B = 75
korelasi1: 0.195741658334
nilai selanjutnya bit: 0
nilainya harus: 1
pengecualian tertangkap di main: Hasil yang salah untuk bit pada indeks 6

Punya 6 bit !!!

Kembali ke Daftar Isi

32.8 TONGKAT MEMORI USB SEBAGAI SUMBER PERANGKAT LUNAK MEMATIKAN

- Siapa yang bisa membayangkan bahwa memory stick USB yang tampak tidak berbahaya akan menjadi sumber potensial malware yang mematikan! Bahwa memang kasus ini dibuktikan dengan sangat meyakinkan oleh Karsten Nohl dan Jacob Lell pada konferensi Black Hat 2014:

  https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf

  Eksploitasi ini dinamai BadUSB oleh penemunya. Diperkirakan sekitar setengah dari perangkat USB di luar sana rentan terhadap eksploitasi BadUSB.

- Jika Anda membaca makalah Nohl dan Lell yang disebutkan di atas, Anda juga harus membaca laporan berikut oleh Stephanie Blanchet Hoareau, Erwan Le Disez, David Boucher, dan Benoit Poulo-Cazajou berikut ini:

  http://www.bertin-it.com/brochure/WP-BadUSB_an-unpatchable-flaw-by-Bertin-IT.pdf

  Salah satu hal yang saya nikmati dari laporan yang ditulis dengan baik ini adalah konteks historis yang diberikannya untuk eksploitasi BadUSB. Melalui laporan inilah saya mengetahui bahwa, pada tahun 2011, Angelos Stavrou dan Zhaohui Wang memberikan ceramah dalam konferensi Black Hat tahun itu yang bertajuk “Memanfaatkan Smart-Phone USB

Connectivity For Fun And Proft, ”di mana mereka menunjukkan bagaimana ponsel Android yang terhubung ke komputer sebagai perangkat USB dapat diemulasi untuk bertindak seperti keyboard untuk memasukkan perintah yang tidak bersahabat ke dalam host.

- Penting untuk disadari bahwa BadUSB adalah tidak tentang file malware apa pun dalam memori flash stik USB. [Itu mungkin untuk dideteksi yang dibuat oleh perangkat lunak anti-virus dan, dalam kasus terburuk, Anda selalu dapat memformat ulang memory stick untuk menyingkirkannya malware yang dicurigai yang berada dalam memori abu tongkat.] BadUSB adalah tentang ancaman malware yang berada di mikrokontroler firmware yang mengontrol cara perangkat beroperasi. Alat saat ini untuk mendeteksi malware tidak dapat mengidentifikasi ancaman berbasis firmware ini. Seseorang dapat membuat argumen bahwa sifat malware ini adalah sedemikian rupa sehingga ia tidak akan dapat dideteksi oleh alat pemindaian virus, saat ini atau di masa mendatang. Lihat akhirnya dari bagian ini untuk argumen ini. [BadUSB juga tidak tentang "Propagasi USB Mode ”untuk malware yang dijelaskan di Kuliah 22. Seperti yang diingat oleh pembaca, jika mesin Windows memilikinya "AutoRun" diaktifkan, sebuah file bernama autorun.inf di perangkat USB akan secara otomatis dijalankan saat file perangkat dicolokkan ke komputer. Salinan yang terinfeksi dari file ini di perangkat dapat menginfeksi komputer dengan file malware.]

- Karsten Nohl dan Jacob Lell memilih untuk tidak mempublikasikan perangkat lunak untuk eksploitasi mereka. Pembicaraan tersebut dilanjutkan dengan presentasi berjudul “Making BadUSB Work For You” pada konferensi Derbycon 2014 oleh Adam Caudill dan Brandon Wilson dimana mereka menunjukkan bahwa mereka telah berhasil mengembangkan implementasi BadUSB exploit mereka sendiri. Mereka punya

membuat kode mereka tersedia di GitHub.

- Sekarang kucing sudah keluar dari tas dan orang-orang sudah mulai memposting kode di web yang memungkinkan eksploitasi ini, Anda mungkin ingin lebih berhati-hati ketika Anda menempelkan memory stick Anda di komputer orang lain atau menempelkan tongkat orang lain.
  memory stick Anda sendiri. [Saat di hotel, siapa yang belum download yang kost melewati dari situs web maskapai penerbangan ke dalam memory stick pribadi dan dibawa ke komputer hotel untuk mencetaknya! Mengingat eksploitasi BadUSB, Anda mungkin tidak ingin melakukannya lagi. (Di masa depan, kamu mungkin hanya ingin mengunduh boarding pass ke smartphone Anda secara langsung). Dengan segala macam penyumbatan memori mereka menempel ke komputer hotel di lobi, selalu ada kemungkinan itu, sengaja atau secara tidak sengaja, seseorang mungkin menggunakan eksploitasi BadUSB untuk menanam malware di komputer tersebut. Bayangkan saja Akibatnya setelah memory stick Anda terinfeksi dengan cara ini, Anda mencolokkannya ke memory stick Anda sendiri komputer!]

- Untuk memahami eksploitasi BadUSB, yang terbaik adalah meninjau kembali alasan utama standar USB dibuat pada pertengahan 1990-an. Apa yang mendorong pengembangan standar ini adalah semakin banyaknya pilihan perangkat yang dapat dihubungkan dengan komputer mereka: keyboard, mouse, webcam, pemutar musik, drive eksternal, dan sebagainya. Dirasakan bahwa jika satu jenis konektor dapat dirancang untuk semua periferal semacam itu, itu akan sangat menyederhanakan dukungan perangkat keras yang perlu dimasukkan ke dalam komputer untuk koneksi transfer data.
  dengan periferal yang berbeda. [Standar USB telah memenuhi tujuan itu. Akronim USB adalah singkatan dari "Universal Serial Bus". Salah satu alasan popularitas USB untuk menghubungkan perangkat portabel ke komputer adalah Anda dapat menghubungkan dan memutuskan perangkat tanpa harus me-reboot host

komputer. Artinya, perangkat USB cenderung hot-swappable.]

- Mempertimbangkan bahwa begitu banyak jenis perangkat yang berbeda dapat menampilkan dirinya sendiri ke komputer Anda melalui koneksi USB, Pernahkah Anda bertanya-tanya tentang bagaimana komputer dapat membedakan antara, katakanlah, keyboard dan thumb drive jika keduanya menampilkan diri ke komputer Anda melalui port perangkat keras yang sama?

- Saat Anda memasukkan perangkat USB ke komputer Anda, hal pertama yang dilakukan OS di komputer Anda adalah menentukan "kelas USB" dari perangkat itu. Standar USB mendefinisikan sejumlah besar kelas (lebih dari 20), beberapa yang paling umum digunakan adalah:

Perangkat Antarmuka Manusia (HID): Perangkat USB yang termasuk dalam kelas ini digunakan untuk menghubungkan perangkat penunjuk (mouse komputer, joystick), keypad, keyboard, dll.

Gambar: Perangkat USB yang termasuk dalam kategori ini digunakan untuk menghubungkan webcam, pemindai, dll., ke komputer.

Printer: Seperti yang bisa Anda tebak dari namanya, perangkat USB yang termasuk dalam kelas ini adalah digunakan untuk menghubungkan berbagai jenis printer ke komputer.

Penyimpanan Massal (MSC): Perangkat USB yang termasuk dalam kelas ini digunakan untuk abu drive memori, pemutar audio digital, kamera, dll. [Seperti yang mungkin sudah Anda duga, singkatan MSC adalah singkatan dari "Mass Storage Class". Nama lain untuk kelas ini adalah UMS untuk "Penyimpanan Massal USB".]

Hub USB: Perangkat semacam itu digunakan untuk memperluas satu port USB ke beberapa port lainnya. [Beberapa laptop paling ringan hanya dilengkapi dengan satu port USB. Jika Anda ingin menghubungkan banyak

perangkat ke laptop seperti itu, Anda memerlukan Hub USB. Selain itu, jika mesin memiliki beberapa port USB, biasanya satu USB Hub yang dibangun secara internal yang diperluas menjadi beberapa port yang Anda lihat di bagian luar laptop Anda (daripada memiliki sirkuit USB independen untuk setiap port terpisah).]

Kartu pintar : Jenis perangkat USB ini dapat digunakan untuk membaca kartu pintar.

dan beberapa lainnya

- Setiap kelas perangkat USB diberi kode numerik dalam standar USB. Misalnya, kode numerik yang terkait dengan kelas HID adalah 0x03, kode yang terkait dengan kelas MSC 0x08.

- Seperti yang disebutkan sebelumnya, segera setelah OS di komputer host mendeteksi perangkat USB, ia menanyakan perangkat USB untuk kelas perangkat tersebut. Perangkat USB merespons kembali dengan kode numerik kelas. OS kemudian memuat driver perangkat lunak yang sesuai dengan kelas perangkat tersebut. [Selanjutnya, semua komunikasi antara komputer host dan perangkat USB berbentuk paket. Byte pertama dari setiap paket adalah byte pengenal paket, yang menyatakan tujuan paket. Misalnya, paket mungkin jabat tangan paket, atau paket bantalan data, atau mungkin kesalahan atau paket pesan status, dll.]

- Dengan asumsi perangkat USB dari kelas MSC, driver perangkat lunak di komputer host kemudian berinteraksi dengan firmware di mikrokontroler di perangkat USB untuk mentransfer data antara komputer host dan memori flash di memori USB. tongkat. [Mikrokontroler hanyalah komputer chip tunggal kecil yang murah, dengan CPU, RAM, dan I / O, itu, untuk perangkat USB, diberi daya oleh arus yang ditarik melalui port USB dari komputer host.

Dan firmware terdiri dari program yang disimpan dalam EEPROM (Electrically Erasable Read Only Memory) itu dijalankan di CPU mikrokontroler.]

- "Mini-review" dari perangkat USB yang disajikan sejauh ini menjelaskan bagaimana perangkat tersebut bekerja dalam kondisi normal. Sekarang mari pertimbangkan aspek berikut dari firmaware yang berada di mikrokontroler perangkat semacam itu yang dapat mengubah memory stick menjadi sumber malware yang berbahaya.

- Untuk memungkinkan perbaikan bug dilakukan pada perangkat lunak dalam mikrokontroler USB dan juga memungkinkan perangkat lunak ditingkatkan versinya, produsen USB mengizinkan alat pihak ketiga untuk mengubah perangkat lunaknya. Bahkan, Anda dapat mengunduh file pabrikan-konsorsium mendukung alat sumber terbuka yang disebut ”Alat Peningkatan Firmware Perangkat USB” untuk tujuan ini dari

  https://admin.fedoraproject.org/pkgdb/package/dfu-util/

  Ini adalah vendor- dan perangkat-independen Peningkatan Firmware Perangkat (DFU) alat untuk memutakhirkan firmware di perangkat USB. Anda dapat menggunakan alat ini untuk mengunduh firmware yang saat ini ada di perangkat USB dan untuk mengunggah versi baru dari perangkat lunak tersebut ke perangkat.

- Fakta bahwa seseorang dapat mengganti perangkat lunak pabrikan di mikrokontroler USB membukanya untuk eksploitasi untuk menyebarkan infeksi malware. Inilah bagaimana itu bisa terjadi:

Anda mengambil memory stick (yang biasanya termasuk dalam kelas MCS) dan Anda mengubah firmware-nya sehingga, setelah dimasukkan ke komputer host, ia melaporkan ke OS bahwa kelasnya adalah HID. Itu akan memungkinkan stik USB untuk bertindak sebagai keyboard vis-a-vis komputer host yang terhubung dengannya. Setiap penekanan tombol yang dikirim oleh USB yang menyamar sebagai keyboard dapat digunakan untuk menjalankan perintah yang menginstal malware dari situs jarak jauh. Perintah yang dijalankan dengan cara ini juga dapat menginstal malware yang akan disimpan secara permanen di host dan diinstal di semua stik memori USB yang dicolokkan ke host di masa mendatang.

- Apa yang membuat eksploit ini sangat berbahaya adalah karena tidak terdeteksi oleh alat pemindaian virus apa pun. Alat-alat ini tidak dimaksudkan untuk memeriksa firmware di perangkat periferal yang terhubung ke komputer.

- Jelas, reaksi pertama Anda terhadap keadaan penerbangan yang dijelaskan di butir sebelumnya kemungkinan besar adalah: Mengapa tidak menambah alat pemindaian virus untuk juga melihat firmware di perangkat periferal yang terhubung ke host? Anda mungkin berpikir tentang alat pemindaian yang ditempatkan pada pembuangan OS sehingga ketika OS pertama kali mendeteksi perangkat USB, itu membuat titik pemeriksaan firmware sebelum mengizinkan pertukaran data apa pun dengan perangkat. Namun ada masalah dengan skenario itu: Bagaimana alat ini membedakan antara USB yang secara sah dimiliki oleh kelas HID dan perangkat yang menyamar sebagai milik

sama?

Kembali ke Daftar Isi

32.9 IP SELULER

- Katakanlah Anda di rumah dan ingin menggunakan ponsel cerdas Anda untuk mengirim pesan teks ke teman Anda yang tinggal di kota yang sama dengan Anda, tetapi yang saat ini kebetulan sedang menikmati minuman lokal di Starbucks di tempat yang jauh. negara. Anggap saja ponsel cerdas teman Anda terhubung ke WiFi Starbuck itu.

- Fakta bahwa pesan teks Anda akan sampai ke smartphone teman Anda terlepas dari di mana tepatnya dia berada di muka bumi sungguh menakjubkan. Pernahkah Anda bertanya-tanya bagaimana bisa operator telepon seluler di akhir Anda
  Tautan komunikasi tahu bagaimana merutekan paket Anda ke telepon teman Anda terlepas dari lokasi telepon itu? [Itu
  Masalah komunikasi yang terlibat di sini lebih kompleks dari yang Anda kira. Di masa lalu, saat semuanya
  telepon memiliki nomor tetap, pertukaran telepon pada ujung sumber dari hubungan komunikasi bisa
  segera cari tahu cara merutekan panggilan telepon hanya dengan memeriksa kode negara, kode area, dll.,
  terkait dengan nomor keluar. Namun hal itu jelas tidak berlaku untuk ponsel berbasis ponsel modern
  komunikasi. Anda mungkin berpikir bahwa smartphone saat ini terhubung ke internet dalam beberapa jarak jauh
  negara memiliki alamat IP yang ditetapkan oleh ISP di lokasi terpencil itu. (Itu pasti akan menjadi
  kasus untuk perangkat non-seluler seperti laptop.) Jika memang demikian, bagaimana jaringan di ujung sumber
  tahu bagaimana merutekan paket ke telepon jarak jauh jika itu adalah sumber yang memulai koneksi?]

- Jawaban atas pertanyaan yang diajukan di atas terletak pada konsep

apa yang dikenal sebagai Dukungan Mobilitas IP sebagaimana didefinisikan dalam RFC 5944. Apa yang dijabarkan dalam RFC 5944 juga secara informal disebut sebagai IP Seluler .

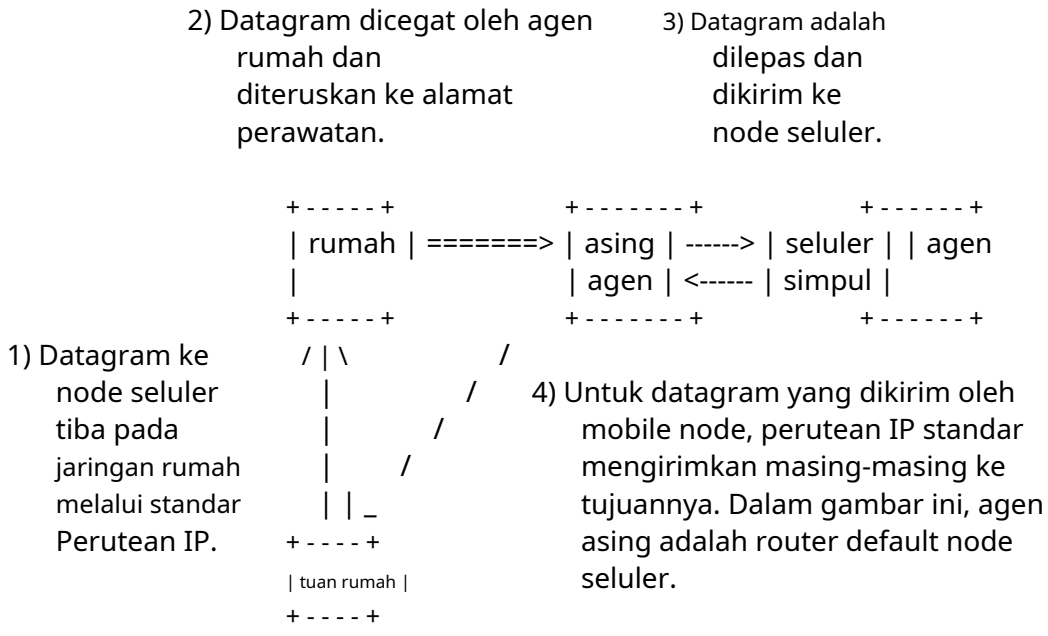
- Menurut standar RFC 5944, setiap "node" seluler dalam jaringan adalah selalu diidentifikasi olehnya alamat IP rumah , terlepas dari lokasi node saat ini. Saat jauh dari rumah, node seluler juga memiliki alamat IP lain yang terkait dengannya; alamat IP kedua ini dikenal sebagai perawatan alamat IP . [Pikirkan tentang alamat IP rumah sebagai pengenal permanen untuk telepon pintar. Saat smartphone tidak ada dari jaringan asalnya, ia memerlukan alamat IP, alamat IP rumah, dan IP perawatan alamat, untuk beroperasi sesuai dengan RFC 5944.]

- Sedangkan mobile node secara unik dikenali olehnya alamat IP rumah , itu perawatan alamat IP , jika ada, adalah node seluler saat ini titik-keterikatan dengan internet.

- Secara informal, operator telepon seluler tempat alamat IP rumah untuk node seluler terdaftar disebut sebagai agen rumah di RFC 5944. Dan operator telepon seluler pada titik keterikatan simpul seluler saat ini dikenal sebagai simpul agen asing . [Untuk definisi resmi: Agen Rumah: Router di rumah node seluler jaringan yang menyalurkan datagram untuk dikirimkan ke node seluler saat jauh dari rumah, dan memelihara informasi lokasi saat ini untuk node seluler. Agen Asing: Router di ponsel jaringan yang dikunjungi node yang menyediakan layanan perutean ke node seluler saat terdaftar. Orang asing agent melepas dan mengirimkan ke datagram node seluler yang disalurkan oleh node seluler agen rumah. Untuk datagram yang dikirim oleh node seluler, agen asing dapat berfungsi sebagai router default untuk

node seluler terdaftar.]

- Terlepas dari titik lampiran saat ini untuk node seluler, jika ponsel pintar Anda ingin mengirim paket ke node seluler itu, ia mengirimkan paket ke agen rumah node seluler. Agen rumah terowongan paket ke agen asing node seluler saat ini, yang, pada gilirannya, mengarahkan paket ke tujuan akhir mereka dengan menggunakan alamat IP yang tepat. Ini diilustrasikan oleh diagram berikut yang diambil dari RFC 5944:



Pengoperasian Mobile IPv4 (dari RFC 5944)

- Dalam diagram yang ditunjukkan di atas, "host" di bagian bawah diagram bisa jadi adalah ponsel pintar Anda dan "node seluler" adalah ponsel pintar teman Anda di lokasi terpencil mana pun di bumi

jika ada jangkauan telepon seluler.

- Yang paling menarik dari diagram perutean yang ditunjukkan di atas adalah jalur yang diambil oleh paket dari ponsel jarak jauh kembali ke ponsel pintar Anda. Seperti yang ditunjukkan oleh panah diagonal, jalur kembali untuk paket melewati agen rumah.

- Poin penting lainnya yang terkait dengan paket pengembalian adalah bahwa alamat IP sumber dalam paket tersebut adalah alamat IP rumah node seluler. Sejauh menyangkut "host" di bagian bawah diagram, paket yang diterimanya dari node seluler yang berlokasi jauh terlihat seolah-olah node seluler tersebut berlokasi di jaringan rumahnya.

- Mari kembali ke perihal smartphone Anda mengirimkan paket ke smartphone teman Anda yang saat ini berada di lokasi yang jauh. Data yang datang dari ponsel cerdas Anda tidak akan terlihat berbeda dari saat ponsel teman Anda dicolokkan ke jaringan rumah. Ini adalah tugas perute di jaringan rumah untuk menyalurkan paket yang datang dari telepon Anda ke perute pada titik pemasangan telepon teman Anda saat ini. Tunneling berarti bahwa perute rumah menempatkan paket yang datang dari ponsel cerdas Anda dalam muatan data dari paket yang dikirim ke perute tempat ponsel cerdas teman Anda saat ini berada. Router itu melepas paket dan mengirimkannya ke smartphone teman Anda.