

Computer Security

v

Contents

Preface	xvii
About the Author	xxi
Acknowledgments	xxiii
I Software Security	1
1 Linux Security Basics	5
1.1 Users and Groups	6
1.1.1 Users	6
1.1.2 Groups	7
1.2 Permissions and Access Control List	8
1.2.1 The Traditional Permission Model	8
1.2.2 Access Control List	10
1.3 Running Command with Privileges	11
1.3.1 Using <code>sudo</code>	11
1.3.2 Set-UID Programs and Security Issues	12
1.3.3 POSIX Capabilities	12
1.4 Authentication	13
1.4.1 Password Authentication	14
1.4.2 The Shadow File	15
1.5 Summary	16
2 Set-UID Programs	19
2.1 The Need for Privileged Programs	20
2.1.1 The Password Dilemma	20
2.1.2 Different Types of Privileged Programs	21
2.2 The Set-UID Mechanism	22
2.2.1 A Superman Story	22
2.2.2 How It Works	22
2.2.3 An Example of Set-UID Program	23
2.2.4 How to Ensure Its Security	24
2.2.5 The Set-GID Mechanism	24
2.3 What Can Go Wrong: What Happened to Superman	25
2.4 Attack Surfaces of Set-UID Programs	26

2.4.1	User Inputs: Explicit Inputs	26
2.4.2	System Inputs	27
2.4.3	Environment Variables: Hidden Inputs	27
2.4.4	Capability Leaking	28
2.5	Invoking Other Programs	30
2.5.1	Unsafe Approach: Using <code>system()</code>	30
2.5.2	Safe Approach: Using <code>execve()</code>	33
2.5.3	Invoking External Commands in Other Languages	34
2.5.4	Lessons Learned: Principle of Isolation	35
2.6	Principle of Least Privilege	35
2.7	Summary	37
3	Environment Variables and Attacks	39
3.1	Environment Variables	40
3.1.1	How to Access Environment Variables	40
3.1.2	How a Process Gets Its Environment Variables	41
3.1.3	Memory Location for Environment Variables	42
3.1.4	Shell Variables and Environment Variables	43
3.2	Attack Surface	46
3.3	Attacks via Dynamic Linker	47
3.3.1	Static and Dynamic Linking	48
3.3.2	Case Study: <code>LD_PRELOAD</code> and <code>LD_LIBRARY_PATH</code>	49
3.3.3	Case Study: OS X Dynamic Linker	52
3.4	Attack via External Program	52
3.4.1	Two Typical Ways to Invoke External Programs	53
3.4.2	Case Study: the <code>PATH</code> environment variable	53
3.4.3	Reduce Attack Surface	54
3.5	Attack via Library	55
3.5.1	Case Study - Locale in UNIX	55
3.6	Application Code	56
3.6.1	Case Study - Using <code>getenv()</code> in Application Code	56
3.7	<code>Set-UID</code> Approach versus Service Approach	57
3.8	Summary	58
4	Buffer Overflow Attack	61
4.1	Program Memory Layout	62
4.2	Stack and Function Invocation	63
4.2.1	Stack Memory Layout	63
4.2.2	Frame Pointer	64
4.3	Stack Buffer-Overflow Attack	65
4.3.1	Copy Data to Buffer	66
4.3.2	Buffer Overflow	67
4.3.3	Exploiting a Buffer Overflow Vulnerability	68
4.4	Setup for Our Experiment	69
4.4.1	Disable Address Randomization	70
4.4.2	Vulnerable Program	70
4.5	Conduct Buffer-Overflow Attack	71
4.5.1	Finding the Address of the Injected Code	71

4.5.2	Improving the Chance of Guessing	72
4.5.3	Finding the Address Without Guessing	73
4.5.4	Constructing the Input File	75
4.6	Attacks with Unknown Address and Buffer Size	77
4.6.1	Knowing the Range of Buffer Size	77
4.6.2	Knowing the Range of the Buffer Address	78
4.6.3	A General Solution	79
4.7	Buffer Overflow Attacks on 64-bit Programs	80
4.7.1	The Stack Layout	80
4.7.2	A Challenge in Attacks: Zeros in Address	80
4.7.3	Overcoming the Challenge Caused by Zeros	81
4.7.4	Another Challenge in Attacks: Small Buffer	83
4.8	Countermeasures: Overview	84
4.9	Address Randomization	86
4.9.1	Address Randomization on Linux	86
4.9.2	Effectiveness of Address Randomization	88
4.10	StackGuard	89
4.10.1	The Observation and the Idea	89
4.10.2	Manually Adding Code to Function	90
4.10.3	StackGuard Implementation in <code>gcc</code>	91
4.11	Defeating the Countermeasure in <code>bash</code> and <code>dash</code>	94
4.12	Summary	96
5	Return-to-libc Attack and ROP	97
5.1	Introduction: Non-Executable Stack	98
5.2	The Attack Experiment: Setup	99
5.3	Launch the Return-to-libc Attack: Part I	101
5.3.1	Task A: Find the Address of the <code>system()</code> Function	102
5.3.2	Task B: Find the Address of the String <code>"/bin/sh"</code>	102
5.4	Launch the Return-to-libc Attack: Part II	104
5.4.1	Function Prologue	105
5.4.2	Function Epilogue	105
5.4.3	Function Prologue and Epilogue Example	106
5.4.4	Perform Task C	107
5.4.5	Construct Malicious Input	108
5.4.6	Launch the Attack	109
5.4.7	Attacks on 64-bit Programs	110
5.5	Defeating Shell's Countermeasure	111
5.6	Return-Oriented Programming	113
5.6.1	Experiment Setup	113
5.6.2	Tracking the values of the <code>esp</code> and <code>ebp</code> registers	115
5.6.3	Chaining Function Calls Without Arguments	116
5.6.4	Chaining Function Calls With Arguments: Skipping Prologue	117
5.6.5	Chaining Function Calls With Arguments: via <code>leave</code> and <code>ret</code>	121
5.6.6	Chaining Function Calls With Zero in the Argument	125
5.6.7	Use the Chaining Technique to Get Root Shell	126
5.6.8	Further Generalization: Return-Oriented Programming	128
5.7	Summary	129

6	Format String Vulnerability	131
6.1	Functions with Variable Number of Arguments	132
6.1.1	How to Access Optional Arguments	132
6.1.2	How <code>printf()</code> Accesses Optional Arguments	134
6.2	Format String with Missing Optional Arguments	135
6.3	Vulnerable Program and Experiment Setup	137
6.4	Exploiting the Format String Vulnerability	139
6.4.1	Attack 1: Crash Program	139
6.4.2	Attack 2: Print Out Data On the Stack	139
6.4.3	Attack 3: Change the Program's Data in the Memory	140
6.4.4	Attack 4: Change the Program's Data to a Specific Value	141
6.4.5	Attack 4 (Continuation): A Much Faster Approach	142
6.5	Code Injection Attack using Format String Vulnerability	145
6.5.1	The Revised Vulnerable Program	145
6.5.2	The Attack Strategy	146
6.5.3	The Attack Program	147
6.5.4	Reducing the Size of Format String	149
6.5.5	Attacks on 64-bit Programs: Challenges Caused by Zeros	151
6.5.6	A Note on <code>%n</code> 's Length Modifiers	152
6.6	Countermeasures	153
6.6.1	Developer	153
6.6.2	Compiler	153
6.6.3	Address Randomization	154
6.7	Relationship with the Buffer-Overflow Attack	155
6.8	Summary	156
7	Race Condition Vulnerability	157
7.1	The General Race Condition Problem	158
7.2	Race Condition Vulnerability	159
7.3	Experiment Setup	161
7.4	Exploiting Race Condition Vulnerabilities	163
7.4.1	Choose a Target File	163
7.4.2	Launch Attack	163
7.4.3	Monitor the Result	164
7.4.4	Running the Exploit	165
7.4.5	Potential Failure	166
7.5	An Improved Method	166
7.6	Countermeasures	168
7.6.1	Atomic Operation	168
7.6.2	Repeating Check and Use	169
7.6.3	Sticky Symlink Protection	170
7.6.4	Principle of Least Privilege	172
7.7	Summary	173

8	Dirty COW	175
8.1	Memory Mapping using <code>mmap()</code>	176
8.2	<code>MAP_SHARED</code> , <code>MAP_PRIVATE</code> and Copy On Write	177
8.3	Discard the Copied Memory	179
8.4	Mapping Read-Only Files	179
8.5	The Dirty COW Vulnerability	181
8.6	Exploiting the Dirty COW Vulnerability	182
8.6.1	Selecting <code>/etc/passwd</code> as Target File	183
8.6.2	Set Up the Memory Mapping and Threads	183
8.6.3	The <code>write</code> Thread	184
8.6.4	The <code>madvise</code> Thread	185
8.6.5	The Attack Result	185
8.7	Summary	186
9	Shellcode	187
9.1	Introduction	188
9.2	Writing Assembly Code	188
9.3	Writing Shellcode: the Basic Idea	189
9.3.1	Writing Shellcode Using C	190
9.3.2	Writing a Shellcode Using Assembly Code	191
9.4	Approach 1: The Stack Approach	191
9.4.1	Step 1. Setting <code>ebx</code> : getting the address of the shell string	191
9.4.2	Step 2. Setting <code>ecx</code> : getting the address of the argument array	192
9.4.3	Step 3. Setting <code>edx</code>	193
9.4.4	Step 4. Invoking the <code>execve()</code> system call	193
9.4.5	Putting Everything Together	193
9.4.6	Getting Rid of Zeros from Shellcode	194
9.5	Approach 2: the Code Segment Approach	195
9.6	Writing 64-bit Shellcode	197
9.7	A Generic Shellcode	198
9.8	Summary	200
10	Reverse Shell	201
10.1	Introduction	202
10.2	File Descriptor and Redirection	202
10.2.1	File Descriptor	202
10.2.2	Standard IO Devices	204
10.2.3	Redirection	205
10.2.4	Understanding the Syntax of Redirection	206
10.2.5	How To Implement Redirection	207
10.3	Redirecting Input/Output to a TCP Connection	209
10.3.1	Redirecting Output to a TCP Connection	209
10.3.2	Redirecting Input to a TCP Connection	210
10.3.3	Redirecting to TCP Connection From Shell	211
10.4	Reverse Shell	212
10.4.1	Redirecting the Standard Output	212
10.4.2	Redirecting the Standard Input	212
10.4.3	Redirecting the Standard Error	214

10.4.4	Code Injection	214
10.5	Summary	215
II	Web Security	217
11	Web Security Basics	221
11.1	The Web Architecture	222
11.2	Web Browser	222
11.2.1	HTML and Document Object Model (DOM)	222
11.2.2	CSS: Cascading Style Sheets	223
11.2.3	Dynamic Content	223
11.2.4	JavaScript	224
11.3	Web Server: HTTP Server and Web Applications	224
11.3.1	Case Study: Apache Server	225
11.3.2	How HTTP Server Interacts with Web Applications	226
11.4	Browser-Server Communication: The HTTP Protocol	227
11.4.1	Types of HTTP Requests: GET and POST	228
11.4.2	HTTPS	229
11.5	Cookies and Sessions	229
11.5.1	The Stateless Nature	229
11.5.2	Cookies	230
11.5.3	Tracking Using Cookies	230
11.5.4	Sessions and Session Cookies	232
11.6	Sandboxing JavaScript	232
11.6.1	Access Page Data and Document Object Model (DOM)	234
11.6.2	Access Browser Data	235
11.6.3	Access File Systems	235
11.7	Ajax Request and Security	236
11.7.1	Ajax Example	237
11.7.2	Same Origin Policy on Ajax	237
11.7.3	Cross-Domain Ajax Request	238
11.7.4	Case Study: Bypassing Same Origin Policies	238
11.7.5	WebSocket	239
11.8	Summary	240
12	Cross Site Request Forgery	241
12.1	Cross-Site Requests and Its Problems	242
12.2	Cross-Site Request Forgery Attack	243
12.3	CSRF Attacks on HTTP GET Services	244
12.3.1	HTTP GET and POST Services	244
12.3.2	The Basic Idea of CSRF Attacks	245
12.3.3	Attack on Elgg's Add-friend Service	245
12.4	CSRF Attacks on HTTP POST Services	247
12.4.1	Constructing a POST Request Using JavaScript	247
12.4.2	Attack on Elgg's Edit-Profile Service	248
12.5	Countermeasures	250
12.5.1	Using the <code>referer</code> Header	251

12.5.2	Same-Site Cookies	251
12.5.3	Secret Token	253
12.5.4	Case Study: Elgg's Countermeasures	254
12.6	Summary	255
13	Cross-Site Scripting Attack	257
13.1	The Cross-Site Scripting Attack	258
13.1.1	Non-persistent (Reflected) XSS Attack	259
13.1.2	Persistent XSS Attack	260
13.1.3	What damage can XSS cause?	260
13.2	XSS Attacks in Action	261
13.2.1	Prelude: Injecting JavaScript Code	261
13.2.2	Use XSS Attacks to Befriend with Others	262
13.2.3	Use XSS Attacks to Change Other People's Profiles	265
13.3	Achieving Self-Propagation	267
13.3.1	Creating a Self-Propagating XSS Worm: the DOM Approach	268
13.3.2	Create a Self-Propagating Worm: the Link Approach	270
13.4	Preventing XSS attacks	270
13.4.1	Getting Rid of Code from User Inputs	271
13.4.2	Defeating XSS Attacks using Content Security Policy	271
13.4.3	Experimenting with Content Security Policy	274
13.5	JavaScript Code Injection Attacks in General	276
13.5.1	Attack From Third-Party Websites	276
13.5.2	Attacks on Web-Based Mobile Apps	278
13.6	Summary	280
14	SQL Injection Attack	283
14.1	A Brief Tutorial of SQL	284
14.1.1	Log into MySQL	284
14.1.2	Create a Database	284
14.1.3	CREATE a Table	285
14.1.4	INSERT a Row	285
14.1.5	The SELECT Statement	286
14.1.6	WHERE Clause	286
14.1.7	UPDATE SQL Statement	287
14.1.8	Comments in SQL Statements	287
14.2	Interacting with Database in Web Application	288
14.2.1	Getting Data from User	289
14.2.2	Getting Data From Database	290
14.3	Launching SQL Injection Attacks	291
14.3.1	Attack Using cURL	292
14.3.2	Modify Database	293
14.3.3	Multiple SQL Statements	294
14.4	The Fundamental Cause	295
14.5	Countermeasures	297
14.5.1	Filtering and Encoding Data	297
14.5.2	Prepared Statement	298
14.5.3	Defeating SQL Injection Using Prepared Statements	300

14.6 Summary	301
15 Clickjacking Attacks	303
15.1 Prelude	304
15.2 Introduction and Background	304
15.2.1 Overlapping <code>Iframe</code>	304
15.2.2 Opacity	305
15.3 Clickjacking Attacks Using Transparent <code>Iframe</code>	305
15.3.1 Likejacking	305
15.3.2 Hijacking Other Actions	307
15.3.3 Sequence of Clicks	307
15.4 Clickjacking Using Non-Transparent <code>Iframe</code>	308
15.4.1 Likejacking Using Small-Size <code>Iframe</code>	308
15.4.2 Stealing Login Credentials	309
15.5 Countermeasures	310
15.5.1 Framekiller and Framebuster	311
15.5.2 X-Frame-Options	311
15.5.3 Content-Security Policy	313
15.6 Security on Iframes	315
15.6.1 Same Origin Policy	315
15.6.2 Sandboxing Iframes	317
15.7 Summary	318
16 Shellshock Attack	319
16.1 Background: Shell Functions	320
16.2 The Shellshock Vulnerability	322
16.2.1 Vulnerable Version of <code>bash</code>	322
16.2.2 The Shellshock Bug	322
16.2.3 Mistake in the Bash Source Code	323
16.2.4 How Was the Vulnerability Fixed	324
16.2.5 Exploiting the Shellshock vulnerability	325
16.3 Shellshock Attack on <code>Set-UID</code> Programs	325
16.4 Shellshock Attack on CGI Programs	327
16.4.1 Experiment Environment Setup	327
16.4.2 How Web Server Invokes CGI Programs	328
16.4.3 How Attacker Sends Data to <code>Bash</code>	329
16.4.4 Launching the Shellshock Attack	330
16.4.5 Creating Reverse Shell	331
16.5 Remote Attack on PHP	333
16.6 Summary	334
III Hardware Security	337
17 Meltdown Attack	341
17.1 Introduction and Analogy	342
17.1.1 Analogy: The Microsoft Brainteaser Question	342
17.1.2 Stealing A Secret	343

17.1.3	Side Channels	343
17.2	Side Channel Attacks via CPU Cache	344
17.2.1	Time Difference When Accessing Cache vs. Memory	344
17.2.2	Using CPU Cache as a Side Channel	346
17.3	The Room Holding Secret: The Kernel	349
17.3.1	Secret Data in Kernel Space	349
17.3.2	The Guard: Preventing Direct Access to Kernel Memory	351
17.3.3	Avoid Getting Killed: Handling Error/Exceptions in C	351
17.4	Passing the Guard: Out-of-Order Execution by CPU	354
17.5	The Meltdown Attack	356
17.5.1	A Naive Approach	356
17.5.2	Improve the Attack by Getting the Secret Data Cached	357
17.5.3	Improve the Attack Using Assembly Code	358
17.5.4	Improve the Attack Using Statistic Approach	359
17.6	Countermeasures	361
17.7	Summary	361
18	Spectre Attack	363
18.1	Introduction	364
18.2	Out-of-Order Execution and Branch Prediction	364
18.2.1	An Experiment	365
18.2.2	Experiment Results	367
18.3	The Spectre Attack	367
18.3.1	The Setup for the Experiment	368
18.3.2	The Program Used in the Experiment	369
18.4	Improve the Attack Using Statistic Approach	371
18.5	Spectre Variant and Mitigation	373
18.6	Summary	374
IV	Cryptography	375
19	Secret-Key Encryption	379
19.1	Introduction	380
19.2	Substitution Cipher	380
19.2.1	Monoalphabetic Substitution Cipher	380
19.2.2	Breaking Monoalphabetic Substitution Cipher	381
19.2.3	Polyalphabetic Substitution Cipher	385
19.2.4	The Enigma Machine	386
19.3	DES and AES Encryption Algorithms	387
19.3.1	DES: Data Encryption Standard	387
19.3.2	AES: Advanced Encryption Standard	388
19.4	Encryption Modes	389
19.4.1	Encryption Modes	390
19.4.2	Electronic Codebook (ECB) Mode	390
19.4.3	Cipher Block Chaining (CBC) Mode	390
19.4.4	Cipher Feedback (CFB) Mode	392
19.4.5	Output Feedback (OFB) Mode	394

19.4.6	Counter (CTR) Mode	394
19.4.7	Modes for Authenticated Encryption	395
19.4.8	Padding	396
19.5	Initialization Vector and Attacks	398
19.5.1	Mistake: Using the Same IV	398
19.5.2	Mistake: Using a Predictable IV	400
19.6	The Padding Oracle Attack	403
19.6.1	The Experiment Setup	404
19.6.2	How the Attack Works: the Main Idea	404
19.6.3	Finding the Value of $D_2[15]$	405
19.6.4	Finding the Value of $D_2[14]$	407
19.6.5	Finding the Value of $D_2[13]$	407
19.7	Programming using Cryptography APIs	408
19.8	Authenticated Encryption and the GCM Mode	410
19.8.1	The GCM Mode	411
19.8.2	Programming using the GCM Mode	412
19.9	Summary	414
20	One-Way Hash Function	415
20.1	Introduction	416
20.2	Concept and Properties	416
20.2.1	Cryptographic Properties	416
20.2.2	Replay the Number Game	417
20.3	Algorithms and Programs	417
20.3.1	The MD (Message Digest) Series	418
20.3.2	The SHA (Secure Hash Algorithm) Series	418
20.3.3	How Hash Algorithm Works	419
20.3.4	One-Way Hash Commands	419
20.3.5	Computing One-Way Hash in Programs	420
20.3.6	Performance of One-Way Hash Functions	422
20.4	Applications of One-Way Hash Functions	422
20.4.1	Integrity Verification	422
20.4.2	Committing a Secret Without Telling It	423
20.4.3	Password Verification	424
20.4.4	Trusted Timestamping	427
20.5	Message Authentication Code (MAC)	428
20.5.1	Constructing MAC and Potential Attacks	429
20.5.2	Launching the Length Extension Attack	430
20.5.3	Case Study: Length Extension Attack on Flickr	432
20.5.4	The Keyed-Hash MAC (HMAC) Algorithm	433
20.6	Blockchain and Bitcoins	434
20.6.1	Hash Chain and Blockchain	434
20.6.2	Make Chaining Difficult	435
20.6.3	Adding Incentives and Bitcoin	437
20.7	Hash Collision Attacks	437
20.7.1	Security Impact of Collision Attacks	438
20.7.2	Generating Two Different Files with the Same MD5 Hash	439
20.7.3	Generating Two Programs with the Same MD5 Hash	441

20.7.4	Making the Two Programs Behave Differently	443
20.7.5	Hash-Colliding X.509 Certificates	446
20.8	Summary	446
21	Public Key Cryptography	449
21.1	Introduction	450
21.2	Diffie-Hellman Key Exchange	450
21.2.1	Diffie-Hellman Key Exchange	451
21.2.2	Turn DH Key Exchange into a Public-Key Encryption Algorithm	452
21.3	The RSA Algorithm	453
21.3.1	Math Background: Modulo Operation	454
21.3.2	Math Background: Euler's Theorem	454
21.3.3	Math Background: Extended Euclidean Algorithm	455
21.3.4	The RSA Algorithm	456
21.3.5	Exercise: Small Number	457
21.3.6	Exercise: Large Number	458
21.3.7	Performance	460
21.3.8	Hybrid Encryption	461
21.3.9	Other Public-Key Encryption Algorithms	462
21.4	Using OpenSSL Tools to Conduct RSA Operations	462
21.4.1	Generating RSA keys	462
21.4.2	Extracting the public key	464
21.4.3	Encryption and Decryption	464
21.5	Paddings for RSA	464
21.5.1	Attacks Against Textbook RSA	465
21.5.2	Paddings: PKCS#1 v1.5 and OAEP	465
21.6	Digital Signature	467
21.6.1	Digital Signature using RSA	467
21.6.2	DSA and Other Digital Signature Algorithms	469
21.7	Programming Using Public-Key Cryptography APIs	469
21.7.1	Key Generation	470
21.7.2	Encryption and Decryption	471
21.7.3	Digital Signature	472
21.8	Applications	474
21.8.1	Authentication	474
21.8.2	HTTPS and TLS/SSL	476
21.8.3	Chip Technology Used in Credit Cards	476
21.9	Summary and Further Learning	478
22	Public Key Infrastructure	481
22.1	Attack on Public Key Cryptography	482
22.1.1	Man-in-the-Middle (MITM) Attack	482
22.1.2	Defeating MITM Attacks	483
22.1.3	Public Key Infrastructure	483
22.2	Public Key Certificates	484
22.2.1	X.509 Digital Certificate	484
22.2.2	Get Certificate from a Real Server	485
22.3	Certificate Authority (CA)	486

22.3.1	The Core Functionalities of CA	486
22.3.2	Becoming a CA and Setup	487
22.3.3	Generating Keys and Certificates	488
22.4	Getting Certificate from CA	488
22.4.1	Generating Public/Private Keys	488
22.4.2	Generating Certificate Signing Request	489
22.4.3	Adding Alternative Names	490
22.4.4	Asking CA to Sign the Request	491
22.5	Using Public Key Certificate to Secure Web Servers	491
22.5.1	OpenSSL's Built-in Server	492
22.5.2	Apache Setup for HTTPS	493
22.6	Root and Intermediate Certificate Authorities	494
22.6.1	Root CAs and Self-Signed Certificate	494
22.6.2	Intermediate CAs and Chain of Trust	495
22.6.3	Creating Certificates for Intermediate CA	496
22.6.4	Apache Setup	497
22.6.5	Trusted CAs in the Real World	498
22.7	How PKI Defeats the MITM Attack	498
22.7.1	Attacker Forwards the Authentic Certificate	498
22.7.2	Attacker Creates a Fake Certificate	499
22.7.3	Attackers Send Their Own Certificates	499
22.7.4	The Man-In-The-Middle Proxy	500
22.8	Attacks on the Public-Key Infrastructure	501
22.8.1	Attack on CA's Verification Process	502
22.8.2	Attack on CA's Signing Process	503
22.8.3	Attacks on the Algorithms	503
22.8.4	Attacks on User Confirmation	504
22.9	Types of Digital Certificates	505
22.9.1	Domain Validated Certificates (DV)	505
22.9.2	Organizational Validated Certificates (OV)	506
22.9.3	Extended Validated Certificates (EV)	506
22.10	Summary	507