

Inhalt

Materialien zum Buch	13
Geleitwort	15
1 Einleitung	17
1.1 Über dieses Buch	17
1.2 Zielgruppe	19
1.3 Wie Sie mit dem Buch arbeiten	20
1.4 Die Autoren	20
2 Exploit! So schnell führt ein Programmierfehler zum Root-Zugriff	21
2.1 Das Szenario	21
2.2 Die Vorbereitungsarbeiten, Informationssammlung	22
2.3 Analyse und Identifikation von Schwachstellen	23
2.4 Ausnutzung der XSS-Schwachstelle	25
2.5 Analyse und Identifikation weiterer Schwachstellen	26
2.6 Zugriff auf das interne Netzwerk	31
2.7 Angriff auf das interne Netz	35
2.8 Privilege Escalation am Entwicklungsserver	39
2.9 Analyse des Angriffs	42
3 Einführung in die sichere Softwareentwicklung	43
3.1 Ein Prozessmodell für sichere Softwareentwicklung	44
3.2 Die Praktiken der sicheren Softwareentwicklung	46

3.2.1	Code-Review	47
3.2.2	Architectural Risk Analysis	47
3.2.3	Risk-Based Security-Tests	48
3.2.4	Penetration Testing	49
3.2.5	Abuse Cases	50
3.2.6	Security Operations	51
3.3	Fachwissen für sichere Softwareentwicklung	51
3.3.1	Injection	52
3.3.2	Broken Authentication	53
3.3.3	Sensitive Data Exposure	55
3.3.4	XML External Entities (XXE)	56
3.3.5	Broken Access Control	57
3.3.6	Security Misconfiguration	59
3.3.7	Cross Site Scripting (XSS)	60
3.3.8	Insecure Deserialization	62
3.3.9	Using Components with Known Vulnerabilities	64
3.3.10	Insufficient Logging & Monitoring	65
4	Grundlagenwissen für sicheres Programmieren	67
4.1	Praktiken der agilen Softwareentwicklung	68
4.2	Die Programmiersprache C	69
4.2.1	Ein einfaches C-Programm	70
4.2.2	Automatisierung des Build-Prozesses	72
4.2.3	Die Erstellung und Verwendung von Bibliotheken in C	73
4.3	Die Programmiersprache Java	76
4.3.1	Ein einfaches Java-Programm	76
4.3.2	Automatisierung des Build-Prozesses	77
4.3.3	Die Erstellung und Verwendung von Bibliotheken in Java	79
4.4	Versionierung von Quellcode	82
4.5	Debugging und automatisiertes Testen	84
4.5.1	Debugging	85
4.5.2	Automatisiertes Testen	85
4.6	Continuous Integration	91
4.7	Beispiele auf GitHub und auf rheinwerk-verlag.de	94

5	Reverse Engineering	97
<hr/>		
5.1	Analyse von C-Applikationen	97
5.1.1	Die x86-Architektur	97
5.1.2	Die x86-64-Architektur	100
5.1.3	Speicheraufteilung von C-Applikationen	101
5.1.4	Der GNU Debugger	102
5.1.5	Die Verwendung des Heap	106
5.1.6	Die Verwendung des Stacks	112
5.1.7	Reverse Engineering: Ein konkretes Beispiel	120
5.2	Analyse von Java-Applikationen	129
5.2.1	Classdateiformat	130
5.2.2	Speicheraufteilung von Java-Applikationen	132
5.2.3	Befehlssatz	133
5.2.4	Der Java-Disassembler	135
5.2.5	Class Loader	136
5.2.6	Garbage Collectors	139
5.2.7	Just-in-Time Compiler	141
5.2.8	Reverse Engineering Java: Ein konkretes Beispiel	143
5.3	Code Obfuscation	148
6	Sichere Implementierung	151
<hr/>		
6.1	Reduzieren Sie die Sichtbarkeit von Daten und Code	151
6.1.1	Zugriffskontrollen in Java	152
6.1.2	Objekte als Parameter	153
6.1.3	Object Serialization	157
6.1.4	Immutable Objects	158
6.1.5	Java Reflection API	159
6.2	Der sichere Umgang mit Daten	160
6.2.1	Repräsentation von Daten	161
6.2.2	Input-Validierung	170
6.2.3	Output-Codierung	174
6.3	Der richtige Umgang mit Fehlern	176
6.3.1	Fehlercodes	176
6.3.2	Exceptions	177
6.3.3	Logging	179

6.4	Kryptografische APIs richtig einsetzen	182
6.4.1	Java Cryptography Architecture	183
6.4.2	Sichere Datenspeicherung	185
6.5	Statische Codeanalyse	211
6.5.1	Manuelles Code-Review	212
6.5.2	Automatisiertes Code-Review	213
6.5.3	Analyse von Bibliotheken	223

7 Sicheres Design 227

7.1	Architekturbasierte Risikoanalyse	227
7.1.1	Analyse der Angriffsfläche	228
7.1.2	Bedrohungsmodellierung	229
7.2	Designprinzipien für sichere Softwaresysteme	232
7.3	Das HTTP-Protokoll	235
7.3.1	HTTP-Transaktionen	236
7.3.2	Cookies	239
7.3.3	HTTPS	240
7.3.4	Interception Proxy	242
7.4	Sicheres Design von Webapplikationen	244
7.4.1	Clientseitige Kontrolle	244
7.4.2	Zugriffskontrolle	248
7.4.3	Authentifizierung	249
7.4.4	Session-Management	254
7.4.5	Autorisierung	257
7.4.6	Datenspeicherung	263
7.4.7	Verhinderung von Browserangriffen	271

8 Kryptografie 281

8.1	Verschlüsselung	281
8.1.1	Grundbegriffe	281
8.1.2	Symmetrische Verschlüsselung: Designideen von AES	290
8.1.3	Asymmetrische Verschlüsselung: Hinter den Kulissen von RSA	298

8.1.4	RSA: Security-Überlegungen	304
8.1.5	Diffie-Hellman Key Exchange	305
8.2	Hash-Funktionen	309
8.2.1	Grundlegende Eigenschaften von Hash-Funktionen	309
8.2.2	Angriffe auf Hash-Funktionen	313
8.2.3	Ausgewählte Architekturen von Hash-Funktionen	319
8.3	Message Authentication Codes und digitale Signaturen	321
8.3.1	Message Authentication Codes	322
8.3.2	Digitale Signaturen	324
8.4	NIST-Empfehlungen	327

9 Sicherheitslücken finden und analysieren 329

9.1	Installation der Windows-Testinfrastruktur	329
9.1.1	Installation des i.Ftp-Clients	330
9.1.2	Installation der Debugging-Umgebung	332
9.1.3	Installation der PyWin-Umgebung	334
9.2	Manuelle Analyse der Anwendung	335
9.2.1	Identifikation von Datenkanälen in die Anwendung	336
9.2.2	Analyse der XML-Konfigurationsdateien	339
9.3	Automatische Schwachstellensuche mittels Fuzzing	340
9.4	Analyse des Absturzes im Debugger	343
9.5	Alternativen zum Fuzzing	344
9.6	Tools zur Programmanalyse	344
9.6.1	Ollly Debug	345
9.6.2	Immunity Debugger	348
9.6.3	WinDebug	348
9.6.4	x64dbg	349
9.6.5	IDA Pro	351
9.6.6	Hopper	351
9.6.7	GDB – Gnu Debugger	353
9.6.8	EDB – Evan’s Debugger	353
9.6.9	Radare2	354
9.6.10	Zusammenfassung der Tools	355

10	Buffer Overflows ausnutzen	357
<hr/>		
10.1	Die Crash-Analyse des i.Ftp-Clients	357
10.2	Offsets ermitteln	360
10.3	Eigenen Code ausführen	363
10.4	Umgang mit Bad Characters	368
10.5	Shellcode generieren	372
10.5.1	Bind Shellcode	372
10.5.2	Reverse Shellcode	375
10.6	Exception Handling ausnutzen	377
10.7	Analyse unterschiedlicher Buffer-Längen	379
10.8	Buffer Offsets berechnen	382
10.9	SEH-Exploits	382
10.10	Heap Spraying	386
11	Schutzmaßnahmen einsetzen	391
<hr/>		
11.1	ASLR	391
11.2	Stack Cookies	393
11.3	SafeSEH	395
11.4	SEHOP	396
11.5	Data Execution Prevention	396
11.6	Schutz gegen Heap Spraying	399
12	Schutzmaßnahmen umgehen	401
<hr/>		
12.1	Was sind Reliable Exploits?	401
12.2	Bypass von ASLR	402
12.2.1	Review des i.Ftp-Exploits	402
12.2.2	Verwendung von ASLR-freien Modulen	404
12.2.3	Verwendung von ASLR-freien Modulen – i.Ftp.exe	405
12.2.4	Partielles Überschreiben der Sprungadresse	408

12.2.5	Egg Hunting	411
12.2.6	Verwendung von ASLR-freien Modulen – Lgi.dll	416
12.2.7	Brute Force einer Sprungadresse	417
12.2.8	Partielles Überschreiben der Return-Adresse II	417
12.2.9	Ermitteln der Adressen aus dem laufenden Programm	417
12.2.10	Fehlertolerante Sprungbereiche	418
12.3	Bypass von Stack Cookies	418
12.4	Bypass von SafeSEH	419
12.5	Bypass von SEHOP	420
12.6	Data Execution Prevention (DEP) – Bypass mittels Return Oriented Programming	423
12.6.1	DEP unter Windows	424
12.6.2	Return Oriented Programming	429
12.7	DEP Bypass mittels Return-to-libc	449
13	Format String Exploits	451
<hr/>		
13.1	Formatstrings	451
13.2	Die fehlerhafte Anwendung	454
13.3	Aufbau der Analyseumgebung	457
13.4	Analyse des Stack-Inhalts	459
13.5	Speicherstellen mit %n überschreiben	463
13.6	Die Exploit-Struktur	466
13.7	Die Ermittlung von Adressen und Offsets	468
13.8	Die Verifikation der Adressen im Debugger	471
13.9	Die erste lauffähige Version des Exploits	473
13.10	ASLR Bypass	477
14	Real Life Exploitation	485
<hr/>		
14.1	Heartbleed	485
14.2	SSL OpenFuck	488

Inhalt

14.3 Shellshock	493
14.4 Eternal Blue	495
14.5 Spectre und Meltdown	504
14.6 Stagefright	509
Index	511