

OMAR SANTOS
RON TAYLOR

Cert Guide

Learn, prepare, and practice for exam success



CompTIA®

PenTest+

PEARSON IT
CERTIFICATION

Save 10%
on Exam
Voucher

See Inside

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



CompTIA® PenTest+

Cert Guide

Omar Santos
Ron Taylor



CompTIA® PenTest+ Cert Guide

Omar Santos

Ron Taylor

Copyright © 2019 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-7897-6035-7

ISBN-10: 0-7897-6035-5

Library of Congress Control Number: 2018956261

01 18

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson IT Certification cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

MICROSOFT® WINDOWS®, AND MICROSOFT OFFICE® ARE REGISTERED TRADEMARKS OF THE MICROSOFT CORPORATION IN THE U.S.A. AND OTHER COUNTRIES. THIS BOOK IS NOT SPONSORED OR ENDORSED BY OR AFFILIATED WITH THE MICROSOFT CORPORATION.

Warning and Disclaimer

This book is designed to provide information about the CompTIA PenTest+ exam. Every effort has been made to make this book as complete and accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the supplemental online content or programs accompanying it.

MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE INFORMATION CONTAINED IN THE DOCUMENTS AND RELATED GRAPHICS PUBLISHED AS PART OF THE SERVICES FOR ANY PURPOSE. ALL SUCH DOCUMENTS AND RELATED GRAPHICS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS INFORMATION, INCLUDING ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF INFORMATION AVAILABLE FROM THE SERVICES.

THE DOCUMENTS AND RELATED GRAPHICS CONTAINED HEREIN COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED HEREIN AT ANY TIME. PARTIAL SCREEN SHOTS MAY BE VIEWED IN FULL WITHIN THE SOFTWARE VERSION SPECIFIED.

Editor-in-Chief

Mark Taub

Product Line Manager

Brett Bartow

Acquisitions Editor

Paul Carlstroem

Managing Editor

Sandra Schroeder

Development Editor

Christopher Cleveland

Project Editor

Mandie Frank

Copy Editor

Kitty Wilson

Technical Editors

Chris McCoy

Benjamin Taylor

Editorial Assistant

Vanessa Evans

Designer

Chuti Prasertsith

Composition

codemantra

Indexer

Erika Millen

Proofreader

Christopher Morris

Contents at a Glance

Introduction	xxiii
CHAPTER 1	Introduction to Ethical Hacking and Penetration Testing 3
CHAPTER 2	Planning and Scoping a Penetration Testing Assessment 25
CHAPTER 3	Information Gathering and Vulnerability Identification 63
CHAPTER 4	Social Engineering Attacks 121
CHAPTER 5	Exploiting Wired and Wireless Networks 143
CHAPTER 6	Exploiting Application-Based Vulnerabilities 207
CHAPTER 7	Exploiting Local Host and Physical Security Vulnerabilities 277
CHAPTER 8	Performing Post-Exploitation Techniques 333
CHAPTER 9	Penetration Testing Tools 361
CHAPTER 10	Understanding How to Finalize a Penetration Test 471
CHAPTER 11	Final Preparation 505
APPENDIX A	Answers to the “Do I Know This Already?” Quizzes and Q&A Sections 511
Index	541

Contents

Introduction xxiii

Chapter 1 Introduction to Ethical Hacking and Penetration Testing 3

“Do I Know This Already?” Quiz 3

Understanding Ethical Hacking and Penetration Testing 6

 What Is the Difference Between Ethical Hacking and Nonethical Hacking? 6

 Why Do We Need to Do Penetration Testing? 7

 Understanding the Current Threat Landscape 7

 Ransomware 8

 IoT 8

 Threat Actors 9

 Exploring Penetration Testing Methodologies 10

 Why Do We Need to Follow a Methodology for Penetration Testing? 10

 Penetration Testing Methods 11

 Surveying Penetration Testing Methodologies 13

 Building Your Own Lab 16

 Requirements and Guidelines for Penetration Testing Labs 18

 What Tools Should You Use in Your Lab? 18

 What if You Break Something? 19

 Review All Key Topics 20

 Define Key Terms 20

 Q&A 21

Chapter 2 Planning and Scoping a Penetration Testing Assessment 25

“Do I Know This Already?” Quiz 25

Explaining the Importance of the Planning and Preparation Phase 29

 Understanding the Target Audience 29

 Rules of Engagement 30

 Communication Escalation Path 31

 Confidentiality of Findings 32

 Budget 32

 Point-in-Time Assessment 33

Impact Analysis and Remediation Timelines	34
Disclaimers	38
Technical Constraints	39
Support Resources	40
Understanding the Legal Concepts of Penetration Testing	41
Contracts	41
<i>Written Authorization</i>	42
SOW	42
MSA	42
NDA	43
Export Restrictions	43
Corporate Policies	43
Learning How to Scope a Penetration Testing Engagement Properly	44
Scope Creep	44
Types of Assessment	45
Special Scoping Considerations	45
Target Selection	46
Strategy	47
Risk Acceptance, Tolerance, and Management	47
<i>Understanding Risk Management</i>	48
<i>Risk Acceptance</i>	48
<i>Risk Mitigation</i>	48
<i>Risk Transfer, Avoidance, and Sharing</i>	49
<i>Risk Appetite and Tolerance</i>	49
Learning the Key Aspects of Compliance-Based Assessments	50
Rules for Completing Compliance-Based Assessments	50
<i>Regulations in the Financial Sector</i>	50
<i>Regulations in the Healthcare Sector</i>	52
<i>Payment Card Industry Data Security Standard (PCI DSS)</i>	53
<i>Key Technical Elements in Regulations You Should Consider</i>	56
Limitations When Performing Compliance-Based Assessments	57
Review All Key Topics	58
Define Key Terms	59
Q&A	59

Chapter 3 Information Gathering and Vulnerability Identification 63

“Do I Know This Already?” Quiz	63
Understanding Information Gathering and Reconnaissance	67
Understanding Active Reconnaissance vs. Passive Reconnaissance	70
Understanding Active Reconnaissance	71
Nmap Scan Types	73
TCP Connect Scan (-sT)	73
UDP Scan (-sU)	74
TCP FIN Scan (-sF)	76
Ping scan (-sn)	77
Exploring the Different Types of Enumeration	78
Host Enumeration	78
User Enumeration	80
Group Enumeration	81
Network Share Enumeration	82
Web Page Enumeration/Web Application Enumeration	83
Service Enumeration	85
Exploring Enumeration via Packet Crafting	85
Understanding Passive Reconnaissance	87
Domain Enumeration	88
Packet Inspection and Eavesdropping	90
Understanding Open Source Intelligence (OSINT) Gathering	90
Exploring Reconnaissance with <i>Recon-ng</i>	90
Understanding the Art of Performing Vulnerability Scans	103
How a Typical Automated Vulnerability Scanner Works	103
Understanding the Types of Vulnerability Scans	104
Unauthenticated Scans	104
Authenticated Scans	105
Discovery Scans	106
Full Scans	106
Stealth Scans	108
Compliance Scans	109
Challenges to Consider When Running a Vulnerability Scan	110
Considering the Best Time to Run a Scan	110

<i>Determining What Protocols Are in Use</i>	110
<i>Network Topology</i>	110
<i>Bandwidth Limitations</i>	111
<i>Query Throttling</i>	111
<i>Fragile Systems/Nontraditional Assets</i>	111
Understanding How to Analyze Vulnerability Scan Results	112
<i>US-CERT</i>	113
<i>The CERT Division of Carnegie Mellon University</i>	113
<i>NIST</i>	114
<i>JPCERT</i>	114
<i>CAPEC</i>	114
<i>CVE</i>	114
<i>CWE</i>	115
How to Deal with a Vulnerability	115
Review All Key Topics	116
Define Key Terms	117
Q&A	117
Chapter 4 Social Engineering Attacks	121
“Do I Know This Already?” Quiz	121
Understanding Social Engineering Attacks	125
Phishing	126
Pharming	126
Malvertising	127
Spear Phishing	128
SMS Phishing	134
Voice Phishing	135
Whaling	135
Elicitation, Interrogation, and Impersonation (Pretexting)	135
Social Engineering Motivation Techniques	137
Shoulder Surfing	137
USB Key Drop and Social Engineering	138
Review All Key Topics	138
Define Key Terms	139
Q&A	139

Chapter 5 Exploiting Wired and Wireless Networks 143

“Do I Know This Already?” Quiz	143
Exploiting Network-Based Vulnerabilities	148
Exploring Windows Name Resolution and SMB Attacks	148
<i>NetBIOS Name Service and LLMNR</i>	148
SMB Exploits	151
DNS Cache Poisoning	155
SNMP Exploits	157
SMTP Exploits	159
<i>SMTP Open Relays</i>	160
<i>Useful SMTP Commands</i>	160
<i>Using Known SMTP Server Exploits</i>	163
FTP Exploits	166
Pass-the-Hash Attacks	168
Kerberos and LDAP-Based Attacks	169
Understanding Man-in-the-Middle Attacks	173
<i>Understanding ARP Spoofing and ARP Cache Poisoning</i>	173
<i>Downgrade Attacks</i>	175
Route Manipulation Attacks	175
Understanding Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks	176
<i>Direct DoS Attacks</i>	176
<i>Reflected DDoS Attacks</i>	178
<i>Amplification DDoS Attacks</i>	178
Network Access Control (NAC) Bypass	179
VLAN Hopping	181
DHCP Starvation Attacks and Rogue DHCP Servers	183
Exploiting Wireless and RF-Based Attacks and Vulnerabilities	185
Installing Rogue Access Points	185
Evil Twin Attacks	185
Deauthentication Attacks	186
Attacking the Preferred Network Lists	189
Jamming Wireless Signals and Causing Interference	189
War Driving	190

Initialization Vector (IV) Attacks and Unsecured Wireless Protocols	190
<i>Attacking WEP</i>	190
<i>Attacking WPA</i>	192
<i>KRACK Attacks</i>	196
<i>Attacking Wi-Fi Protected Setup (WPS)</i>	197
KARMA Attacks	197
Fragmentation Attacks	197
Credential Harvesting	199
Bluejacking and Bluesnarfing	199
Radio-Frequency Identification (RFID) Attacks	200
Review All Key Topics	200
Define Key Terms	202
Q&A	202
Chapter 6 Exploiting Application-Based Vulnerabilities	207
“Do I Know This Already?” Quiz	207
Overview of Web Applications for Security Professionals	213
The HTTP Protocol	213
Understanding Web Sessions	221
How to Build Your Own Web Application Lab	224
Understanding Injection-Based Vulnerabilities	227
Exploiting SQL Injection Vulnerabilities	228
<i>A Brief Introduction to SQL</i>	228
<i>SQL Injection Categories</i>	232
<i>Fingerprinting a Database</i>	234
<i>Surveying the UNION Exploitation Technique</i>	235
<i>Using Booleans in SQL Injection Attacks</i>	237
<i>Understanding Out-of-Band Exploitation</i>	237
<i>Exploring the Time-Delay SQL Injection Technique</i>	239
<i>Surveying a Stored Procedure SQL Injection</i>	239
<i>Understanding SQL Injection Mitigations</i>	240
HTML Injection Vulnerabilities	241
Command Injection Vulnerabilities	241
Exploiting Authentication-Based Vulnerabilities	242

Exploring Credential Brute Forcing	243
Understanding Session Hijacking	245
Understanding Redirect Attacks	249
Taking Advantage of Default Credentials	249
Exploiting Kerberos Vulnerabilities	250
Exploiting Authorization-Based Vulnerabilities	250
Understanding Parameter Pollution	250
Exploiting Insecure Direct Object Reference Vulnerabilities	251
Understanding Cross-Site Scripting (XSS) Vulnerabilities	252
Reflected XSS Attacks	253
Stored XSS Attacks	255
DOM-Based XSS Attacks	256
XSS Evasion Techniques	257
XSS Mitigations	258
Understanding Cross-Site Request Forgery Attacks	260
Understanding Clickjacking	261
Exploiting Security Misconfigurations	262
Exploiting Directory Traversal Vulnerabilities	262
Understanding Cookie Manipulation Attacks	263
Exploiting File Inclusion Vulnerabilities	264
Local File Inclusion Vulnerabilities	264
Remote File Inclusion Vulnerabilities	264
Exploiting Insecure Code Practices	265
Comments in Source Code	265
Lack of Error Handling and Overly Verbose Error Handling	266
Hard-Coded Credentials	266
Race Conditions	266
Unprotected APIs	267
Hidden Elements	270
Lack of Code Signing	270
Review All Key Topics	271
Define Key Terms	272
Q&A	273

Chapter 7 Exploiting Local Host and Physical Security Vulnerabilities 277

“Do I Know This Already?” Quiz	277
Exploiting Local Host Vulnerabilities	281
Insecure Service and Protocol Configurations	281
Local Privilege Escalation	285
<i>Understanding Linux Permissions</i>	286
<i>Understanding SUID or SGID and Unix Programs</i>	291
<i>Insecure SUDO Implementations</i>	294
<i>Ret2libc Attacks</i>	298
Windows Privileges	299
<i>CPassword</i>	299
<i>Clear-Text Credentials in LDAP</i>	300
<i>Kerberoasting</i>	301
<i>Credentials in Local Security Authority Subsystem Service (LSASS)</i>	301
<i>SAM Database</i>	302
<i>Understanding Dynamic Link Library Hijacking</i>	303
<i>Exploitable Services</i>	304
<i>Insecure File and Folder Permissions</i>	305
<i>Understanding Windows Group Policy</i>	305
<i>Keyloggers</i>	306
<i>Scheduled Tasks</i>	307
<i>Escaping the Sandbox</i>	308
<i>Virtual Machine Escape</i>	310
<i>Understanding Container Security</i>	310
Mobile Device Security	314
<i>Understanding Android Security</i>	316
<i>Understanding Apple iOS Security</i>	323
Understanding Physical Security Attacks	326
<i>Understanding Physical Device Security</i>	326
<i>Protecting Your Facilities Against Physical Security Attacks</i>	327
Review All Key Topics	328
Define Key Terms	329
Q&A	329

Chapter 8 Performing Post-Exploitation Techniques 333

- “Do I Know This Already?” Quiz 333
- Maintaining Persistence After Compromising a System 337
- Creating Reverse and Bind Shells 338
 - Command and Control (C2) Utilities 344
 - Creating and Manipulating Scheduled Jobs and Tasks 346
 - Creating Custom Daemons, Processes, and Additional Backdoors 346
 - Creating New Users 346
- Understanding How to Perform Lateral Movement 347
- Post-Exploitation Scanning 347
 - Using Remote Access Protocols 348
 - Using Windows Legitimate Utilities 349
 - Using PowerShell for Post-Exploitation Tasks* 349
 - Using PowerSploit* 351
 - Using the Windows Management Instrumentation for Post-Exploitation Tasks* 354
 - Using Sysinternals and PSExec* 355

Understanding How to Cover Your Tracks and Clean Up Systems After a Penetration Testing Engagement 356

- Review All Key Topics 357
- Define Key Terms 358
- Q&A 358

Chapter 9 Penetration Testing Tools 361

- “Do I Know This Already?” Quiz 361
- Understanding the Different Use Cases of Penetration Testing Tools and How to Analyze Their Output 365
- Penetration Testing–Focused Linux Distributions 365
 - Kali Linux* 366
 - Parrot* 367
 - BlackArch Linux* 367
 - CAINE* 369
 - Security Onion* 369
- Common Tools for Reconnaissance and Enumeration 370
- Tools for Passive Reconnaissance* 370
 - Tools for Active Reconnaissance* 390

Common Tools for Vulnerability Scanning	400
Common Tools for Credential Attacks	420
<i>John the Ripper</i>	420
<i>Cain and Abel</i>	424
<i>Hashcat</i>	425
<i>Hydra</i>	428
<i>RainbowCrack</i>	429
<i>Medusa</i> and <i>Ncrack</i>	430
<i>CeWL</i>	431
<i>Mimikatz</i>	432
<i>Patator</i>	432
Common Tools for Persistence	433
Common Tools for Evasion	434
<i>Veil</i>	434
<i>Tor</i>	438
<i>Proxychains</i>	439
<i>Encryption</i>	439
<i>Encapsulation and Tunneling Using DNS and Other Protocols Like NTP</i>	440
Exploitation Frameworks	442
<i>Metasploit</i>	442
<i>BeEF</i>	449
Common Decompilation, Disassembling, and Debugging Tools	450
<i>The GNU Project Debugger (GDB)</i>	450
<i>Windows Debugger</i>	452
<i>OllyDbg</i>	452
<i>edb Debugger</i>	452
<i>Immunity Debugger</i>	454
<i>IDA</i>	454
<i>Objdump</i>	455
Common Tools for Forensics	457
Common Tools for Software Assurance	458
<i>Findbugs, Findsecbugs, and SonarQube</i>	458
<i>Fuzzers and Fuzz Testing</i>	458
<i>Peach</i>	459

<i>Mutiny Fuzzing Framework</i>	459
<i>American Fuzzy Lop</i>	459
Wireless Tools	459
Leveraging Bash, Python, Ruby, and PowerShell in Penetration Testing Engagements	460
Introducing the Bash Shell	460
A Brief Introduction to Python	461
A Brief Introduction to Ruby	461
A Brief Introduction to PowerShell	462
Review All Key Topics	462
Define Key Terms	465
Q&A	465
Chapter 10 Understanding How to Finalize a Penetration Test	471
“Do I Know This Already?” Quiz	471
Explaining Post-Engagement Activities	474
Surveying Report Writing Best Practices	475
Understanding the Importance of a Quality Report	475
Discussing Best Practices of Writing a Penetration Testing Report	476
<i>Knowing Your Audience</i>	476
<i>Avoiding Cutting and Pasting</i>	477
<i>Relating the Findings to the Environment</i>	477
<i>Starting the Report While You Are Testing</i>	478
Exploring Tools for Collecting and Sharing Information	478
<i>Using Dradis for Effective Information Sharing and Reporting</i>	478
<i>Steps in Using the Dradis Framework CE on Kali Linux</i>	479
Exploring the Common Report Elements	490
PCI Data Security Standard Reporting Guidelines	491
Expanding on the Common Report Elements	493
<i>Executive Summary</i>	493
<i>Methodology</i>	494
<i>Finding Metrics and Measurements</i>	494
<i>Findings and Recommendations for Remediation</i>	495

Understanding Report Handling and Communications Best Practices	499
Understanding Best Practices in Report Handling	499
<i>Correctly Classifying Report Contents</i>	499
<i>Controlling Distribution Method and Media</i>	499
Explaining the Importance of Appropriate Communication	500
Review All Key Topics	501
Define Key Terms	502
Q&A	502
Chapter 11 Final Preparation	505
Tools for Final Preparation	505
Pearson Cert Practice Test Engine and Questions on the Website	505
<i>Accessing the Pearson Test Prep Software Online</i>	506
<i>Accessing the Pearson Test Prep Software Offline</i>	506
Customizing Your Exams	507
Updating Your Exams	508
<i>Premium Edition</i>	508
Chapter-Ending Review Tools	509
Suggested Plan for Final Review/Study	509
Summary	509
Appendix A Answers to the “Do I Know This Already?” Quizzes and Q&A Sections	511
Index	541

About the Authors

Omar Santos is a principal engineer in the Cisco Product Security Incident Response Team (PSIRT) within Cisco’s Security Research and Operations. He mentors and leads engineers and incident managers during the investigation and resolution of security vulnerabilities in all Cisco products, including cloud services. Omar has been working with information technology and cybersecurity since the mid-1990s. He has designed, implemented, and supported numerous secure networks for Fortune 100 and 500 companies and the U.S. government. Prior to his current role, he was a technical leader within the Worldwide Security Practice and the Cisco Technical Assistance Center (TAC), where he taught, led, and mentored many engineers within both organizations.

Omar is an active member of the security community, where he leads several industrywide initiatives and standards bodies. His active role helps businesses, academic institutions, state and local law enforcement agencies, and other participants that are dedicated to increasing the security of the critical infrastructure.

Omar often delivers technical presentations at many cybersecurity conferences. He is the author of more than 20 books and video courses. You can follow Omar on any of the following:

- Personal website: omarsantos.io and theartofhacking.org
- Twitter: @santosomar
- LinkedIn: <https://www.linkedin.com/in/santosomar>

Ron Taylor has been in the information security field for almost 20 years, 10 of which were spent in consulting. In 2008, he joined the Cisco Global Certification Team as an SME in information assurance. In 2012, he moved into a position with the Security Research & Operations group, where his focus was mostly on penetration testing of Cisco products and services. He was also involved in developing and presenting security training to internal development and test teams globally. In addition, he provided consulting support to many product teams as an SME on product security testing. He then spent some time as a consulting systems engineer specializing in Cisco’s security product line. In his current role, he works in the Cisco Product Security Incident Response Team (PSIRT). He has held a number of industry certifications, including GPEN, GWEB, GCIA, GCIH, GWAPT, RHCE, CCSP, CCNA, CISSP, and MCSE. Ron is also a Cisco Security Blackbelt, SANS mentor, cofounder and president of the Raleigh BSides Security Conference, and an active member of the Packet Hacking Village team at Defcon.

You can follow Ron on any of the following:

- Twitter: @Gu5G0rman
- LinkedIn: www.linkedin.com/in/-RonTaylor

Dedication

I would like to dedicate this book to my lovely wife, Jeannette, and my two beautiful children, Hannah and Derek, who have inspired and supported me throughout the development of this book.

I also dedicate this book to my father, Jose, and to the memory of my mother, Generosa. Without their knowledge, wisdom, and guidance, I would not have the goals that I strive to achieve today.

—Omar

The most important thing in life is family:

To my wife of 17 years: Kathy, without your support and encouragement, I would not be where I am today.

To my kids, Kaitlyn, Alex, and Grace: You give me the strength and motivation to do what I do.

To my parents: It was your example that instilled in me the drive and work ethic that has gotten me this far.

—Ron

Acknowledgments

This book is a result of concerted efforts of various individuals whose help brought this book to reality. We would like to thank the technical reviewers, Chris McCoy and Ben Taylor, for their significant contributions and expert guidance.

We would also like to express our gratitude to Chris Cleveland, Kitty Wilson, Mandie Frank, Paul Carlstroem, and Brett Bartow for their help and continuous support throughout the development of this book.

About the Technical Reviewers

Chris McCoy is a technical leader in the Cisco Advanced Security Initiatives Group (ASIG). He has more than 20 years of experience in the networking and security industry. He has a passion for computer security, finding flaws in mission-critical systems, and designing mitigations to thwart motivated and resourceful adversaries. He was formerly with Spirent Communications and the U.S. Air Force. Chris is CCIE certified in the Routing & Switching and Service Provider tracks, which he has held for more than 10 years. You can follow Chris on Twitter@chris_mccoy.

Benjamin Taylor is a security researcher currently working in the Cisco Security and Trust Organization. He has worked in the security industry for more than 10 years. His work spans numerous architectures and operating systems. His background and experience include security evaluations, penetration testing, security architecture reviews, product security compliance, digital forensics, and reverse engineering.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Credits

Cover: GlebSStock/Shutterstock

NIST Computer Security Resource Center defines the term Hacker
Sun Tzu, The Art of War

High Level Organization of the Standard by The Penetration Testing Execution Standard

PCI Security Standard council, Information Supplement: Penetration Testing Guidance

Penetration Testing Framework 0.59 by VulnerabilityAssessment.co.uk

Open Source Security Testing Methodology Manual (OSSTMM), Contemporary Security testing and analysis

GLBA (12 U.S.C. § 1843(k))

NY DFS Cybersecurity Regulation

Covered Entities and Business Associates, The HIPAA Rules apply to covered entities and business associates.

Payment Card Industry (PCI) Data Security Standard (DSS) and Payment Application Data Security Standard (PA-DSS), April 2016.

Elaine Barker, NIST Special Publication 800-57 Part 1 Revision 4 Recommendation for Key Management Part 1: General, January 2016.

Figure Credits

Figure 2-1 Screenshot of Gantt Chart © 2018 Microsoft Corporation

Figure 3-2 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-4 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-6 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-8 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-10 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-12 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-13 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-14 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-15 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-16 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-17 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-18 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-19 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-20 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-21 Screenshot of Kali Linux © 2018 Kali Linux

Figure 3-22 Screenshot of Kali Linux © 2018 Kali Linux

- Figure 3-23 Screenshot of Google © 2018 Google, LLC.
- Figure 3-24 Screenshot of DNSdumpster © 2018 Hacker Target Pty Ltd
- Figure 3-25 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-26 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-27 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-28 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-29 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-30 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-31 Screenshot of Shodan © 2013-2018 Shodan®
- Figure 3-32 Screenshot of Shodan © 2013-2018 Shodan®
- Figure 3-33 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-34 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-35 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-36 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-37 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-38 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 3-39 Omar Santos
- Figure 3-41 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-3 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-4 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-5 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-6 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-7 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-8 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-9 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-10 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 4-11 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-15 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-18 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-19 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-20 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-23 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-24 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-25 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 5-26 Screenshot of Kali Linux © 2018 Kali Linux
- Figure 6-3 Screenshot of Wireshark © The Wireshark team
- Figure 6-4 Screenshot of Wireshark © The Wireshark team
- Figure 6-7 Screenshot of WebGoat © OWASP
- Figure 6-9 Screenshot of W3school © 1999-2018 by Refsnes Data
- Figure 6-11 Screenshot of WebGoat © OWASP
- Figure 6-12 Screenshot of WebGoat © OWASP
- Figure 6-13 Screenshot of WebGoat © OWASP
- Figure 6-14 Screenshot of DVWA © 2014-2017 Dewhurst Security

- Figure 6-16 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 6-18 Screenshot of Wireshark © The Wireshark team
Figure 6-21 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 6-22 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 6-23 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 6-24 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 6-25 Screenshot of DVWA © 2014-2017 Dewhurst Security
Figure 7-5 Screenshot of Unix Permission Calculator © 2017 Dan's Tools
Figure 7-6 Screenshot of Unix Permission Calculator © 2017 Dan's Tools
Figure 7-7 Screenshot of Visudo Command Man Page © Visudo
Figure 7-9 Screenshot of Microsoft Excel © 2018 Microsoft Corporation
Mobile Top 10 2016-Top 10 by OWASP
Figure 7-13 Screenshot of Android Studio © Google, LLC.
Figure 7-14 Screenshot of Android Studio © Google, LLC.
Figure 7-15 Screenshot of Kali Linux © 2018 Kali Linux
Figure 8-4 Screenshot of Kali Linux © 2018 Kali Linux
Figure 8-6 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-1 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-2 Screenshot of Parrot Linux © 2013-2018 Lorenzo Faletra
Figure 9-3 Screenshot of BlackArch Linux © 2013-2018 BlackArch Linux
Figure 9-4 Screenshot of BlackArch Linux © 2013-2018 BlackArch Linux
Figure 9-5 Screenshot of Caine © Caine
Figure 9-6 Screenshot of Security Onion © Security Onion Solutions, LLC
Figure 9-7 Screenshot of Shodan © 2013-2018 Shodan®
Figure 9-8 Screenshot of Maltego © Paterva
Figure 9-9 Screenshot of Maltego © Paterva
Figure 9-10 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-11 Screenshot of Censys © 2018 Censys
Figure 9-12 Screenshot of Zenmap © Nmap
Figure 9-13 Screenshot of Zenmap © Nmap
Category: Vulnerability Scanning Tools by OWASP
Figure 9-14 Screenshot of Greenbone © 2017 Greenbone Networks
Figure 9-15 Screenshot of Greenbone © 2017 Greenbone Networks
Figure 9-16 Screenshot of Greenbone © 2017 Greenbone Networks
Figure 9-17 Screenshot of Greenbone © 2017 Greenbone Networks
OWASP Zed Attack Proxy Project by OWASP
Figure 9-18 Screenshot of OWASP ZAP © OWASP
Figure 9-19 Screenshot of OWASP ZAP © OWASP
Figure 9-20 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-21 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-22 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-23 Screenshot of Kali Linux © 2018 Kali Linux

- Figure 9-24 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-25 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-26 Screenshot of the art of hacking © 2018 Omar Santos
Figure 9-27 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-28 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-29 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-30 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-31 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-32 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-33 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-34 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-35 Screenshot of BeEF Exploitation Framework © Beef
Figure 9-36 Screenshot of OllyDbg © 2000-2014 Oleh Yusluk
Figure 9-37 Screenshot of Kali Linux © 2018 Kali Linux
Figure 9-38 Screenshot of IDA © 2017 Hex-Rays SA.
Figure 9-39 Screenshot of IDA © 2017 Hex-Rays SA.
Figure 10-1 Screenshot of Kali Linux © 2018 Kali Linux
Figure 10-2 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-3 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-4 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-5 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-6 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-7 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-8 Screenshot of Kali Linux © 2018 Kali Linux
Figure 10-9 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-10 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-11 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-12 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-13 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-14 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-15 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-16 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-17 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-18 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-19 Screenshot of Dradis © 2012-2018 Dradis Framework
Figure 10-21 Screenshot of SQLMap © 2006-2018 by Bernardo Damele
Assumpcao Guimaraes
Figure 10-22 Screenshot of SQLMap © 2006-2018 by Bernardo Damele
Assumpcao Guimaraes ISO 31000
Chapter Opener images: Charlie Edwards/Gettyimages

Introduction

CompTIA PenTest+ is a security penetration testing certification that focuses on performance-based and multiple-choice questions, as well as simulations that require a candidate to demonstrate the hands-on ability to complete a penetration testing engagement. PenTest+ candidates must demonstrate their skills in planning and scoping a penetration testing engagement. Candidates are also required to know how to mitigate security weaknesses and vulnerabilities, as well as how to exploit them.

CompTIA PenTest+ is an intermediate-level cybersecurity career certification. Historically, the only intermediate-level cybersecurity certification was the CompTIA Cybersecurity Analyst (CySA+). Today, PenTest+ provides an alternate path from those who want to specialize in security penetration testing (ethical hacking).

CompTIA PenTest+ and CySA+ can be taken in any order. Either exam typically follows the skills learned in Security+. The main difference between CySA+ and PenTest+ is that CySA+ focuses on defensive security (including incident detection and response), whereas PenTest+ focuses on offensive security (ethical hacking or penetration testing).

NOTE CompTIA PenTest+ is a globally recognized certification that demonstrates the holder's knowledge and skills across a broad range of security topics.

The Goals of the CompTIA PenTest+ Certification

The CompTIA PenTest+ certification was created and is managed by one of the most prestigious organizations in the world and has a number of stated goals. Although not critical for passing the exam, having knowledge of the organization and of these goals is helpful in understanding the motivation behind the creation of the exam.

Sponsoring Bodies

The Computing Technology Industry Association (CompTIA) is a vendor-neutral IT certification body that is recognized worldwide. CompTIA has been in existence for more than 20 years. It develops certificate programs for IT support, networking, security, Linux, cloud, and mobility. CompTIA is a nonprofit trade association.

PenTest+ is one of a number of security-related certifications offered by CompTIA. Other certifications offered by this organization include the following:

- CompTIA Security+
- CompTIA Cybersecurity Analyst (CySA+)
- CompTIA Advanced Security Practitioner (CASP)

CompTIA offers certifications in other focus areas, including the following:

- CompTIA IT Fundamentals
- CompTIA A+
- CompTIA Network+
- CompTIA Cloud Essentials
- CompTIA Cloud+
- CompTIA Linux+
- CompTIA Server+
- CompTIA Project+
- CompTIA CTT+

Stated Goals

The goal of CompTIA in its administration of the PenTest+ certification is to provide a reliable instrument to measure an individual's knowledge of cybersecurity penetration testing (ethical hacking). This knowledge is not limited to technical skills alone but extends to all aspects of a successful penetration testing engagement.

The Exam Objectives (Domains)

The CompTIA PenTest+ exam is broken down into five major domains. This book covers all the domains and the subtopics included in them. The following table lists the breakdown of the domains represented in the exam:

Domain	Percentage of Representation in Exam
1.0 Planning and Scoping	15%
2.0 Information Gathering and Vulnerability Identification	22%
3.0 Attacks and Exploits	30%
4.0 Penetration Testing Tools	17%
5.0 Reporting and Communication	16%
	Total 100%

1.0 Planning and Scoping

The Planning and Scoping domain, which is covered in Chapter 2, discusses the importance of good planning and scoping in a penetration testing or ethical hacking

engagement. Comprising 15% of the exam, it covers several key legal concepts and the different aspects of compliance-based assessment. It Covers topics including the following:

- Explain the importance of planning for an engagement.
- Explain key legal concepts.
- Explain the importance of scoping an engagement properly.
- Explain the key aspects of compliance-based assessments.

2.0 Information Gathering and Vulnerability Identification

The Information Gathering and Vulnerability Identification domain, which is covered in Chapter 3, starts out by discussing in general what reconnaissance is and the difference between passive and active reconnaissance methods. It touches on some of the common tools and techniques used. From there it covers the process of vulnerability scanning and how vulnerability scanning tools work, including how to analyze vulnerability scanning results to provide useful deliverables and the process of leveraging the gathered information in the exploitation phase. Finally, it discusses some of the common challenges to consider when performing vulnerability scans. This domain accounts for 22% of the exam. Topics include the following:

- Given a scenario, conduct information gathering using appropriate techniques.
- Given a scenario, perform a vulnerability scan.
- Given a scenario, analyze vulnerability scan results.
- Explain the process of leveraging information to prepare for exploitation.
- Explain weaknesses related to specialized systems.

3.0 Attacks and Exploits

The Attacks and Exploits domain is covered throughout Chapters 4 through 8. These chapters include topics such as social engineering attacks, exploitation of wired and wireless networks, application-based vulnerabilities, local host and physical security vulnerabilities, and post-exploitation techniques. It encompasses 30% of the exam. Topics include the following:

- Compare and contrast social engineering attacks.
- Given a scenario, exploit network-based vulnerabilities.
- Given a scenario, exploit wireless and RF-based vulnerabilities.
- Given a scenario, exploit application-based vulnerabilities.

- Given a scenario, exploit local host vulnerabilities.
- Summarize physical security attacks related to facilities.
- Given a scenario, perform post-exploitation techniques.

4.0 Penetration Testing Tools

The Penetration Testing Tools domain is covered in Chapter 9. In this chapter, you will learn different use cases for penetration testing tools. You will also learn how to analyze the output of some of the most popular penetration testing tools to make informed assessments. At the end of the chapter, you will learn how to leverage the bash shell, Python, Ruby, and PowerShell to perform basic scripting. This domain accounts for 17% of the exam. The topics include the following:

- Given a scenario, use Nmap to conduct information gathering exercises.
- Compare and contrast various use cases of tools.
- Given a scenario, analyze tool output or data related to a penetration test.
- Given a scenario, analyze a basic script (limited to bash, Python, Ruby, and PowerShell).

5.0 Reporting and Communication

The Reporting and Communication domain is covered in Chapter 10, which starts out by discussing post-engagement activities, such as cleanup of any tools or shells left on systems that were part of the test. From there it covers report writing best practices, including the common report elements as well as findings and recommendations. Finally, it touches on report handling and proper communication best practices. This domain makes up 16% of the exam. Topics include the following:

- Given a scenario, use report writing and handling best practices.
- Explain post-report delivery activities.
- Given a scenario, recommend mitigation strategies for discovered vulnerabilities.
- Explain the importance of communication during the penetration testing process.

Steps to Earning the PenTest+ Certification

To earn the PenTest+ certification, a test candidate must meet certain prerequisites and follow specific procedures. Test candidates must qualify for and sign up for the exam.

Recommended Experience

There are no prerequisites for the PenTest+ certification. However, CompTIA recommends that candidates possess Network+, Security+, or equivalent knowledge.

NOTE Certifications such as Cisco CCNA CyberOps can help candidates and can be used as an alternative to Security+.

CompTIA also recommends a minimum of three to four years of hands-on information security or related experience.

Signing Up for the Exam

The steps required to sign up for the PenTest+ exam are as follows:

1. Create a Pearson Vue account at pearsonvue.com and schedule your exam.
2. Complete the examination agreement, attesting to the truth of your assertions regarding professional experience and legally committing to the adherence to the testing policies.
3. Review the candidate background questions.
4. Submit the examination fee.

The following website presents the CompTIA certification exam policies:
<https://certification.comptia.org/testing/test-policies>.

Facts About the PenTest+ Exam

The PenTest+ exam is a computer-based test that focuses on performance-based and multiple-choice questions. There are no formal breaks, but you are allowed to bring a snack and eat it at the back of the test room; however, any time used for breaks counts toward 165 minutes allowed for the test. You must bring a government-issued identification card. No other forms of ID will be accepted. You may be required to submit to a palm vein scan.

About the CompTIA® PenTest+ Cert Guide

This book maps to the topic areas of the CompTIA® PenTest+ exam and uses a number of features to help you understand the topics and prepare for the exam.

Objectives and Methods

This book uses several key methodologies to help you discover the exam topics on which you need more review, to help you fully understand and remember those details, and to help you prove to yourself that you have retained your knowledge of those topics. This book does not try to help you pass the exam only by memorization; it seeks to help you truly learn and understand the topics. This book is designed to help you pass the PenTest+ exam by using the following methods:

- Helping you discover which exam topics you have not mastered
- Providing explanations and information to fill in your knowledge gaps
- Supplying exercises that enhance your ability to recall and deduce the answers to test questions
- Providing practice exercises on the topics and the testing process via test questions on the companion website

Customizing Your Exams

In the exam settings screen, you can choose to take exams in one of three modes:

- **Study mode:** Allows you to fully customize your exams and review answers as you are taking the exam. This is typically the mode you would use first to assess your knowledge and identify information gaps.
- **Practice Exam mode:** Locks certain customization options, as it is presenting a realistic exam experience. Use this mode when you are preparing to test your exam readiness.
- **Flash Card mode:** Strips out the answers and presents you with only the question stem. This mode is great for late-stage preparation, when you really want to challenge yourself to provide answers without the benefit of seeing multiple-choice options. This mode does not provide the detailed score reports that the other two modes do, so it will not be as helpful as the other modes at helping you identify knowledge gaps.

In addition to choosing among these three modes, you will be able to select the source of your questions. You can choose to take exams that cover all the chapters, or you can narrow your selection to just a single chapter or the chapters that make up specific parts in the book. All chapters are selected by default. If you want to narrow your focus to individual chapters, simply deselect all the chapters and then select only those on which you wish to focus in the Objectives area.

You can also select the exam banks on which to focus. Each exam bank comes complete with a full exam of questions that cover topics in every chapter. The two exams printed in the book are available to you, as are two additional exams of unique questions. You can have the test engine serve up exams from all four banks or just from one individual bank by selecting the desired banks in the exam bank area.

There are several other customizations you can make to your exam from the exam settings screen, such as the time of the exam, the number of questions served up, whether to randomize questions and answers, whether to show the number of correct answers for multiple-answer questions, and whether to serve up only specific types of questions. You can also create custom test banks by selecting only questions that you have marked or questions on which you have added notes.

Updating Your Exams

If you are using the online version of the Pearson Test Prep software, you should always have access to the latest version of the software as well as the exam data. If you are using the Windows desktop version, every time you launch the software while connected to the Internet, it checks whether there are any updates to your exam data and automatically downloads any changes made since the last time you used the software.

Sometimes, due to many factors, the exam data may not fully download when you activate your exam. If you find that figures or exhibits are missing, you may need to manually update your exams. To update a particular exam you have already activated and downloaded, simply click the **Tools** tab and click the **Update Products** button. Again, this is only an issue with the desktop Windows application.

If you wish to check for updates to the Pearson Test Prep exam engine software, Windows desktop version, simply click the **Tools** tab and click the **Update Application** button. By doing so, you ensure that you are running the latest version of the software engine.

This page intentionally left blank

Exploiting Local Host and Physical Security Vulnerabilities

In this chapter you will learn about exploiting local host vulnerabilities, as well as physical security flaws. This chapter provides details on how to take advantage of insecure services and protocol configurations during a penetration testing engagement. You will also learn how to perform local privilege escalation attacks as part of penetration testing. This chapter provides details to help you gain an understanding of Set-UID, Set-GID, and Unix programs, as well as ret2libc attacks. This chapter also covers privilege escalation attacks against Windows systems and the security flaws of Android and Apple iOS mobile devices. In this chapter you will also gain an understanding of physical security attacks such as piggybacking, tailgating, fence jumping, dumpster diving, lock picking, and badge cloning.

“Do I Know This Already?” Quiz

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. Table 7-1 lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in Appendix A, “Answers to the ‘Do I Know This Already?’ Quizzes and Q&A Sections.”

Table 7-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section	Questions
Exploiting Local Host Vulnerabilities	1–8
Understanding Physical Security Attacks	9–10

CAUTION The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as incorrect for purposes of the self-assessment. Giving yourself credit for an answer you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following is not an insecure service or protocol?
 - a. Cisco Smart Install
 - b. Telnet
 - c. Finger
 - d. Windows PowerSploit
2. Consider the following example:

```
omar@ares:~$ ls -l topsecret.txt
-rwxrwxr-- 1 omar omar 15 May 26 21:15 topsecret.txt
```

What permissions does the user omar have in the topsecret.txt file?
 - a. Read only
 - b. Write only
 - c. Read, write, execute
 - d. Write, execute
3. Which of the following is not true about sticky bits?
 - a. A restricted deletion flag, or sticky bit, is a single bit whose interpretation depends on the file type.
 - b. For directories, the sticky bit prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory; this is called the restricted deletion flag for the directory, and is commonly found on world-writable directories such as /tmp.
 - c. If the sticky bit is set on a directory, files inside the directory cannot be renamed or removed by the owner of the file, the owner of the directory, or the superuser (even though the modes of the directory might allow such an operation).
 - d. For regular files on some older systems, the sticky bit saves the program's text image on the swap device so it will load more quickly when run.

4. Which of the following is a type of attack in which a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the executable memory of the process?
 - a. Ret2libc
 - b. ASLR bypass
 - c. CPassword
 - d. Sticky-bit attack
5. Which of the following is a component of Active Directory's Group Policy Preferences that allows administrators to set passwords via Group Policy?
 - a. Ret2libc
 - b. CPassword
 - c. Sticky-bit
 - d. GPO crack
6. Which of the following tools allows an attacker to dump the LSASS process from memory to disk?
 - a. John the Ripper
 - b. SAMsploit
 - c. Sysinternals ProcDump
 - d. Windows PowerShell
7. The SELinux and AppArmor security frameworks include enforcement rules that attempt to prevent which of the following attacks?
 - a. Lateral movement
 - b. Sandbox escape
 - c. Cross-site request forgery (CSRF)
 - d. Cross-site scripting (XSS)

- 8.** Which of the following is not one of the top mobile security threats and vulnerabilities?
 - a.** Cross-site request forgery (CSRF)
 - b.** Insecure data storage
 - c.** Insecure communication
 - d.** Insecure authentication
- 9.** Which of the following is an attack in which the attacker tries to retrieve encryption keys from a running operating system after using a system reload?
 - a.** Hot-boot
 - b.** Rowhammer
 - c.** Cold boot
 - d.** ASLR bypass
- 10.** Which of the following is the term for an unauthorized individual following an authorized individual to enter a restricted building or facility?
 - a.** Lockpicking
 - b.** Dumpster diving
 - c.** Badge cloning
 - d.** Tailgating

Foundation Topics

Exploiting Local Host Vulnerabilities

Threat actors take advantage of numerous local host vulnerabilities to carry out different attacks. In this section, you will learn about exploits against local host vulnerabilities such as taking advantage of specific operating system flaws, escalating local privileges, stealing credentials, installing key loggers, and abusing physical device security. You will also learn about different virtual machine and container vulnerabilities, and you will learn about cold boot attacks, JTAG debugging, and different attacks that can be carried out over the serial console of a device.

Insecure Service and Protocol Configurations

Key Topic

Many attacks materialize because unused or insecure protocols, services, and associated ports, which are low-hanging fruit opportunities for attackers. In addition, many organizations don't patch vulnerabilities for the services, protocols, and ports they don't use—despite the fact that vulnerabilities may still be present for months or even years.

TIP A best practice is to clearly define and document the services, protocols, and ports that are necessary for business. An organization should ensure that all other services, protocols, and ports are disabled or removed. As a penetration tester, you should always go after insecure protocols, services, and associated ports.

Some protocols should never be used, such as Telnet and Cisco Smart Install. Telnet is a clear-text protocol that exposes the entire contents of any session to anyone who can gain access to the traffic. Secure Shell (SSH) should be used instead. If a switch is running the Cisco Smart Install protocol, any unauthenticated attacker can modify the configuration and fully compromise the switch.

NOTE You can obtain more information about Smart Install and related features from the following Cisco security advisory: <https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180409-smi>.

Other protocols, like Telnet, transfer sensitive data in clear text. Examples of these clear-text protocols include SNMP (versions 1 and 2), HTTP, syslog, IMAP, POP3, and FTP.

TIP In some cases, there is no secure alternative to otherwise insecure management protocols. In such a case, it is very important to understand what is at risk and what mitigation techniques could be implemented.

All insecure protocols are subject to man-in-the-middle (MITM) attacks or to IP traffic capture (sniffing). Example 7-1 shows how easy it is to capture a password from an FTP transaction by just sniffing the traffic using the Linux Tcpdump tool.

Example 7-1 Capturing Passwords and Sniffing Traffic from Clear-Text Protocols by Using Tcpdump

```
root@kubel:~# tcpdump -nnXSs 0 host 10.1.1.12
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens160, link-type EN10MB (Ethernet), capture size 262144
bytes

22:50:23.958387 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [S], seq
314242458, win 29200, options [mss 1460,sackOK,TS val 1523378506 ecr
0,nop,wscale 7], length 0
    0x0000: 4500 003c 1cd0 4000 4006 07d4 0a01 010c E..<..@.0.....
    0x0010: 0a01 010b c664 0015 12ba f59a 0000 0000 .....d.....
    0x0020: a002 7210 acf1 0000 0204 05b4 0402 080a ..r.....
    0x0030: 5acc e94a 0000 0000 0103 0307         Z..J.....
22:50:23.958455 IP 10.1.1.11.21 > 10.1.1.12.50788: Flags [S.], seq
4230935771, ack 314242459, win 28960, options [mss 1460,sackOK,TS val
1523511322 ecr 1523378506,nop,wscale 7], length 0
    0x0000: 4500 003c 0000 4000 4006 24a4 0a01 010b E..<..@.0.$....
    0x0010: 0a01 010c 0015 c664 fc2e f4db 12ba f59b .....d.....
    0x0020: a012 7120 1647 0000 0204 05b4 0402 080a ..q..G.....
    0x0030: 5ace f01a 5acc e94a 0103 0307         Z...Z..J....
22:50:23.958524 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [., ack
4230935772, win 229, options [nop,nop,TS val 1523378506 ecr 1523511322],
length 0
    0x0000: 4500 0034 1cd1 4000 4006 07db 0a01 010c E..4..@.0.....
    0x0010: 0a01 010b c664 0015 12ba f59b fc2e f4dc .....d.....
    0x0020: 8010 00e5 10e4 0000 0101 080a 5acc e94a .....Z..J
    0x0030: 5ace f01a                           Z...
22:50:23.961422 IP 10.1.1.11.21 > 10.1.1.12.50788: Flags [P.], seq
4230935772:4230935792, ack 314242459, win 227, options [nop,nop,TS val
1523511323 ecr 1523378506], length 20: FTP: 220 (vsFTPd 3.0.3)
    0x0000: 4500 0048 04c6 4000 4006 1fd2 0a01 010b E..H..@.0.....
    0x0010: 0a01 010c 0015 c664 fc2e f4dc 12ba f59b .....d.....
    0x0020: 8018 00e3 1653 0000 0101 080a 5ace f01b .....S.....Z...
    0x0030: 5acc e94a 3232 3020 2876 7346 5450 6420 Z..J220.(vsFTPD.
    0x0040: 332e 302e 3329 0d0a                   3.0.3)...
```

```
22:50:23.961485 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [.], ack  
4230935792, win 229, options [nop,nop,TS val 1523378507 ecr 1523511323],  
length 0  
0x0000: 4510 0034 1cd2 4000 4006 07ca 0a01 010c E..4..@.0.....  
0x0010: 0a01 010b c664 0015 12ba f59b fc2e f4f0 .....d.....  
0x0020: 8010 00e5 10ce 0000 0101 080a 5acc e94b .....Z..K  
0x0030: 5ace f01b Z...  
22:50:26.027005 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [P.], seq  
314242459:314242470, ack 4230935792, win 229, options [nop,nop,TS val  
1523379024 ecr 1523511323], length 11: FTP: USER omar  
0x0000: 4510 003f 1cd3 4000 4006 07be 0a01 010c E..?..@.0.....  
0x0010: 0a01 010b c664 0015 12ba f59b fc2e f4f0 .....d.....  
0x0020: 8018 00e5 6a32 0000 0101 080a 5acc eb50 ....j2.....Z..P  
0x0030: 5ace f01b 5553 4552 206f 6d61 720d 0a Z...USER.omar..  
22:50:26.027045 IP 10.1.1.11.21 > 10.1.1.12.50788: Flags [.], ack  
314242470, win 227, options [nop,nop,TS val 1523511839 ecr 1523379024],  
length 0  
0x0000: 4500 0034 04c7 4000 4006 1fe5 0a01 010b E..4..@.0.....  
0x0010: 0a01 010c 0015 c664 fc2e f4f0 12ba f5a6 .....d.....  
0x0020: 8010 00e3 163f 0000 0101 080a 5ace f21f .....?.....Z...  
0x0030: 5acc eb50 Z..P  
22:50:26.027343 IP 10.1.1.11.21 > 10.1.1.12.50788: Flags [P.], seq  
4230935792:4230935826, ack 314242470, win 227, options [nop,nop,TS val  
1523511839 ecr 1523379024], length 34: FTP: 331 Please specify the  
password.  
0x0000: 4500 0056 04c8 4000 4006 1fc2 0a01 010b E..V..@.0.....  
0x0010: 0a01 010c 0015 c664 fc2e f4f0 12ba f5a6 .....d.....  
0x0020: 8018 00e3 1661 0000 0101 080a 5ace f21f .....a.....Z...  
0x0030: 5acc eb50 3333 3120 506c 6561 7365 2073 Z..P331.Please.s  
0x0040: 7065 6369 6679 2074 6865 2070 6173 7377 pecify.the.  
0x0050: 6f72 642e 0d0a password...  
22:50:26.027393 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [.], ack  
4230935826, win 229, options [nop,nop,TS val 1523379024 ecr 1523511839],  
length 0  
0x0000: 4510 0034 1cd4 4000 4006 07c8 0a01 010c E..4..@.0.....  
0x0010: 0a01 010b c664 0015 12ba f5a6 fc2e f512 .....d.....  
0x0020: 8010 00e5 0c98 0000 0101 080a 5acc eb50 .....Z..P  
0x0030: 5ace f21f Z...  
22:50:30.053380 IP 10.1.1.12.50788 > 10.1.1.11.21: Flags [P.], seq  
314242470:314242485, ack 4230935826, win 229, options [nop,nop,TS val  
1523380030 ecr 1523511839], length 15: FTP: PASS badpass1  
0x0000: 4510 0043 1cd5 4000 4006 07b8 0a01 010c E..C..@.0.....  
0x0010: 0a01 010b c664 0015 12ba f5a6 fc2e f512 .....d.....  
0x0020: 8018 00e5 c455 0000 0101 080a 5acc ef3e .....U.....Z..>  
0x0030: 5ace f21f 5041 5353 2062 6164 7061 7373 Z...PASS.badpass  
0x0040: 310d 0a 1..
```

```

22:50:30.085058 IP 10.1.1.11.21 > 10.1.1.12.50788: Flags [P.], seq
4230935826:4230935849, ack 314242485, win 227, options [nop,nop,TS val
1523512854 ecr 1523380030], length 23: FTP: 230 Login successful.

0x0000: 4500 004b 04c9 4000 4006 1fcc 0a01 010b E..K..@.0..... .
0x0010: 0a01 010c 0015 c664 fc2e f512 12ba f5b5 .....d.......
0x0020: 8018 00e3 1656 0000 0101 080a 5ace f616 .....V.....Z...
0x0030: 5acc ef3e 3233 3020 4c6f 6769 6e20 7375 z..>230.Login.
0x0040: 6363 6573 7366 756c 2e0d 0a                     successful...

```

In Example 7-1 a host at IP address 10.1.1.12 initiates an FTP connection to an FTP server with IP address 10.1.1.11. In the packet capture, you can see the initial login transaction where the user (omar) successfully logs in using the password (bad-pass1), as demonstrated in the highlighted lines in Example 7-1. It is possible to use similar utilities, such as Tshark, to capture data from a live network (see <https://www.wireshark.org/docs/man-pages/tshark.html>).

The following are also some of the services that are considered insecure:

- **Rlogin:** <https://linux.die.net/man/1/rlogin>
- **Rsh:** <https://linux.die.net/man/1/rsh>
- **Finger:** <https://linux.die.net/man/1/finger>

The following services should be carefully implemented and not exposed to untrusted networks:

- **Authd (or Identd):** <https://linux.die.net/man/3/ident>
- **Netdump:** <https://linux.die.net/man/8/netdump>
- **Netdump-server:** <https://linux.die.net/man/8/netdump-server>
- **Nfs:** <https://linux.die.net/man/5/nfs>
- **Rwhod:** <https://linux.die.net/man/8/rwhod>
- **Sendmail:** <https://linux.die.net/man/8/sendmail.sendmail>
- **Samba:** <https://linux.die.net/man/7/samba>
- **Yppasswdd:** <https://linux.die.net/man/8/yppasswdd>
- **Ypserv:** <https://linux.die.net/man/8/ypserv>
- **Ypxfrd:** <https://linux.die.net/man/8/ypxfrd>

TIP RedHat provides a great resource that goes over Linux server security; see https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Security_Guide/ch-server.html.

Key Topic

Local Privilege Escalation

Privilege escalation is the process of elevating the level of authority (privileges) of a compromised user or a compromised application. This is done to further perform actions on the affected system or any other systems in the network, typically post-exploitation (that is, after gaining a foothold in the target system and exploiting a vulnerability).

NOTE In Chapter 8, “Performing Post-Exploitation Techniques,” you will learn about additional post-exploitation methodologies and tactics.

The main focus of the post-exploitation phase is to maintain access to the compromised systems and move around in the network while remaining undetected. In many cases, privilege escalation is required to perform those tasks.

It is possible to perform privilege escalation in a few different ways. An attacker may be able to compromise a system by logging in with a non-privileged account. Subsequently, the attacker can go from that unprivileged (or less privileged) account to another account that has greater authority, as shown in Figure 7-1.

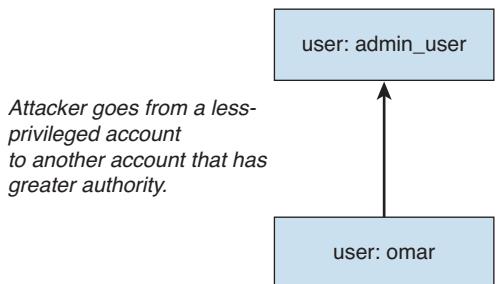


FIGURE 7-1 Privilege Escalation from One Account to Another

It is also possible to perform privilege escalation by “upgrading,” or elevating, the privileges of the same account, as shown in Figure 7-2.



The same account is used, but the attacker manipulates the system to increase the account privilege.

FIGURE 7-2 Privilege Escalation Using the Same Account

In Figure 7-2, the user (omar) belongs to the engineering group (eng) and does not have administrative rights on the system. The attacker then exploits a vulnerability and is able to manipulate the system to put the same user (omar) in the admin group, subsequently giving the user administrative rights on the system.

Understanding Linux Permissions

This book assumes that you have familiarity with Linux and user accounts. As a refresher, in some cases users must be able to accomplish tasks that require privileges (for example, when installing a program or adding another user). This is why **sudo** exists. Example 7-2 shows the first few lines and description of the **sudo** man page.

Example 7-2 The Linux **sudo** Command

```
sudo, sudoedit – execute a command as another user

SYNOPSIS
    sudo -h | -K | -k | -V
    sudo -v [-AknS] [-a type] [-g group] [-h host] [-p prompt] [-u user]
    sudo -l [-AknS] [-a type] [-g group] [-h host] [-p prompt] [-U user]
    [-u user] [command]
    sudo [-AbEHnPS] [-a type] [-C num] [-c class] [-g group] [-h host]
    [-p prompt] [-r role] [-t type] [-u user] [VAR=value] [-i | -s] [command]
    sudoedit [-AknS] [-a type] [-C num] [-c class] [-g group] [-h host]
    [-p prompt] [-u user] file ...

DESCRIPTION
    sudo allows a permitted user to execute a command as the superuser
    or another user, as specified by the security policy. The invoking user's
    real (not effective) user ID is used to determine the user name with which
    to query the security policy.

    sudo supports a plugin architecture for security policies and input/
    output logging. Third parties can develop and distribute their own policy
    and I/O logging plug-ins to work seamlessly with the sudo front end. The
    default security policy is sudoers, which is configured via the file /etc/
    sudoers, or via LDAP. See the Plugins section for more information.

    The security policy determines what privileges, if any, a user has
    to run sudo. The policy may require that users authenticate themselves
    with a password or another authentication mechanism. If authentication
    is required, sudo will exit if the user's password is not entered within
    a configurable time limit. This limit is policy-specific; the default
    password prompt timeout for the sudoers security policy is unlimited.

    Security policies may support credential caching to allow the user
    to run sudo again for a period of time without requiring authentication.
    The sudoers policy caches credentials for 15 minutes, unless overridden
    in sudoers(5). By running sudo with the -v option, a user can update the
    cached credentials without running a command.
```

When invoked as sudoedit, the -e option (described below), is implied.

Security policies may log successful and failed attempts to use sudo. If an I/O plugin is configured, the running command's input and output may be logged as well.

. . . <output omitted for brevity>. . .

On Unix-based systems, you can use the **chmod** command to set permissions values on files and directories.

NOTE You can set permissions of a file or directory (folder) to a given user, a group of users, and others.

Key Topic

With Linux you can set three basic permissions:

- Read (r)
- Write (w)
- Execute (x)

You can apply these permissions to any type of files or to directories. Example 7-3 shows the permissions of a file called omar_file.txt. The user executes the **ls -l** command, and in the portion of the output on the left, you see **-rw-rw-r--**, which indicates that the current user (omar) has read and write permissions.

Example 7-3 Linux File Permissions

```
omar@dionysus:~$ ls -l omar_file.txt
-rw-rw-r-- 1 omar omar 15 May 26 23:45 omar_file.txt
```

Figure 7-3 explains the Linux file permissions.

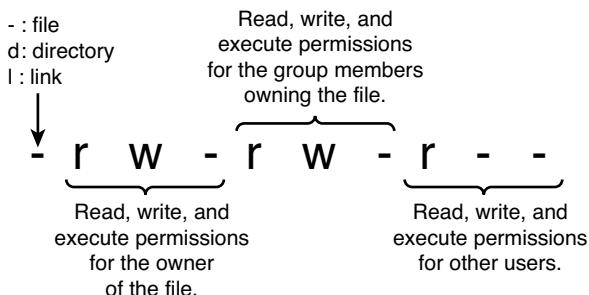


FIGURE 7-3 Explaining Linux File Permissions

Example 7-4 shows how a user belonging to any group can change the permissions of the file to be read, write, executable by using the **chmod 0777** command.

Key Topic
Example 7-4 Changing File Permissions

```
omar@dionysus:~$ chmod 0777 omar_file.txt
omar@dionysus:~$ ls -l omar_file.txt
-rwxrwxrwx 1 omar omar 15 May 26 23:45 omar_file.txt
omar@dionysus:~$
```

As documented in the **chmod** man pages, the restricted deletion flag, or sticky bit, is a single bit whose interpretation depends on the file type. For directories, the sticky bit prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory; this is called the restricted deletion flag for the directory, and it is commonly found on world-writable directories such as /tmp. For regular files on some older systems, the sticky bit saves the program's text image on the swap device so it will load more quickly when run.

TIP The sticky bit is obsolete with files, but it is used for directories to indicate that files can be unlinked or renamed only by their owner or the superuser. Sticky bits were used with files in very old Unix machines due to memory restrictions. If the sticky bit is set on a directory, files inside the directory may be renamed or removed only by the owner of the file, the owner of the directory, or the superuser (even though the modes of the directory might allow such an operation); on some systems, any user who can write to a file can also delete it. This feature was added to keep an ordinary user from deleting another's files from the /tmp directory.

There are two ways that you can use the **chmod** command:

- Symbolic (text) method
- Numeric method

When you use the symbolic method, the structure includes who has access and the permission given. The indication of who has access to the file is as follows:

- **u:** The user that owns the file
- **g:** The group that the file belongs to
- **o:** The other users (that is, everyone else)
- **a:** All of the above (that is, use **a** instead of **ugo**)

Example 7-5 shows how to remove the execute permissions for all users by using the **chmod a-x omar_file.txt** command.

Example 7-5 Symbolic Method Example

```
omar@dionysus:~$ ls -l omar_file.txt
-rwxrwxrwx 1 omar omar 15 May 26 23:45 omar_file.txt
omar@dionysus:~$ chmod a-x omar_file.txt
omar@dionysus:~$ ls -l omar_file.txt
-rw-rw-rw- 1 omar omar 15 May 26 23:45 omar_file.txt
omar@dionysus:~$
```

The **chmod** command allows you to use + to add permissions and - to remove permissions. The **chmod** command clears the set-group-ID (SGID or setgid) bit of a regular file if the file's group ID does not match the user's effective group ID or one of the user's supplementary group IDs, unless the user has appropriate privileges. Additional restrictions may cause the set-user-ID (SUID or setuid) and set-group-ID bits of MODE or FILE to be ignored. This behavior depends on the policy and functionality of the underlying **chmod** system call. When in doubt, check the underlying system behavior. This is clearly explained in the man page of the **chmod** command (**man chmod**). In addition, the **chmod** command retains a directory's SUID and SGID bits unless you explicitly indicate otherwise.

You can also use numbers to edit the permissions of a file or directory (for the owner, group, and others), as well as the SUID, SGID, and sticky bits. Example 7-4 shows the numeric method. The three-digit number specifies the permission, where each digit can be anything from 0 to 7. The first digit applies to permissions for the owner, the second digit applies to permissions for the group, and the third digit applies to permissions for all others.

Figure 7-4 demonstrates how the numeric method works.

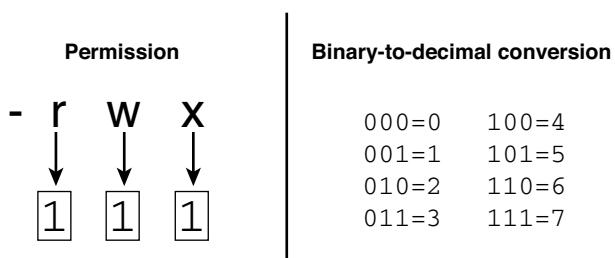


FIGURE 7-4 Explaining the Linux File Permission Numeric Method

As shown in Figure 7-4, a binary number 1 is put under each permission granted and a 0 under each permission not granted. On the right in Figure 7-4, the binary-to-decimal conversion is done. This is why in Example 7-4, the numbers 777 make the file omar_file.txt world-writable (which means any user has read, write, and execute permissions).

A great online tool that you can use to practice setting the different parameters of Linux permissions is the Permissions Calculator, at <http://permissions-calculator.org> (see Figure 7-5).

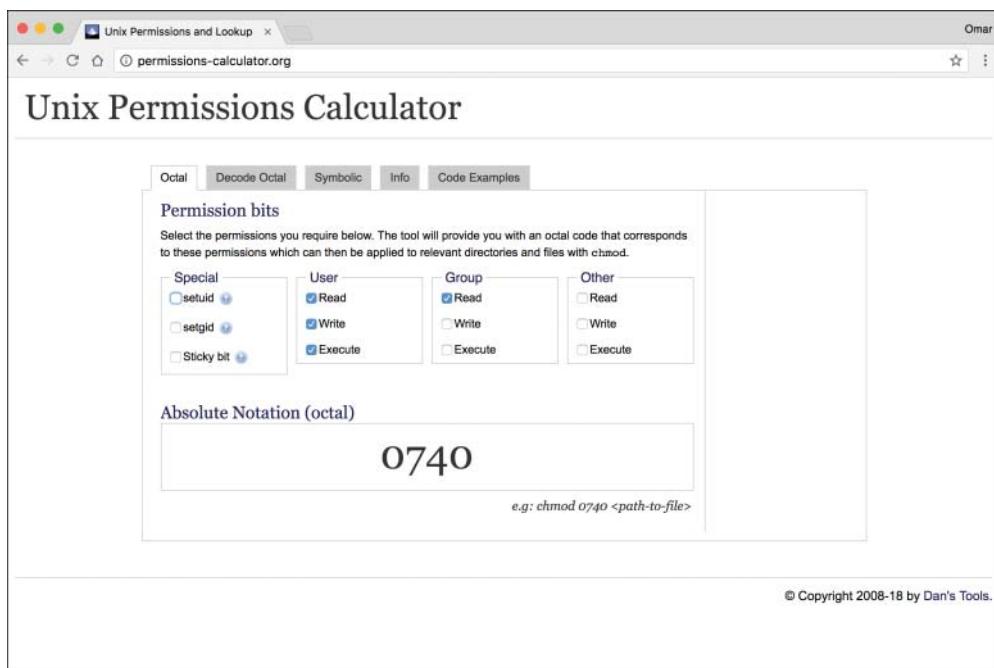


FIGURE 7-5 Permissions Calculator Online Tool

The Permissions Calculator website also provides several examples using PHP, Python, and Ruby to change file and directory permissions programmatically, as shown in Figure 7-6.

The screenshot shows a web browser window titled "Unix Permissions and Lookup" with the URL "permissions-calculator.org/examples/". The page has tabs for Octal, Decode Octal, Symbolic, Info, and Code Examples, with "Code Examples" selected. The main content area is titled "Code Examples" and contains a note: "Here's a few simple code examples for changing file and directory permissions programmatically." Below this, there are two sections: "Chmod with PHP" and "Chmod with Python".

Chmod with PHP

Format:

```
bool chmod(string $path, int $mode);
```

Note: in PHP that the mode must be supplied as octal so always prefix with 0. For more info see <http://php.net/manual/en/function.chmod.php>.

```
<?php
chmod("/somedir/somefile", 755); // decimal; probably incorrect
chmod("/somedir/somefile", "u+rwx,g+rx"); // string; incorrect
chmod("/somedir/somefile", 0755); // octal; correct value of mode
?>
```

```
<?php
// SGID bit set, Everything for user, and group and read and exec
// for other
chmod("/somedir/somefile", 02775);
?>
```

Chmod with Python

Format:

```
os.chmod(path, mode)
```

In python it's possible to use octal values or use values defined in the stat module. For more info see <http://docs.python.org/library/os.html#os.chmod>.

```
import os

# Everything for user and group and read and exec for other
os.chmod("/somedir/somefile", 0775)

# SGID bit set, Everything for user and group and read and exec for other
```

FIGURE 7-6 Changing Permissions Programmatically

Understanding SUID or SGID and Unix Programs

A program or a script in which the owner is root (by setting its Set-UID bit) will execute with superuser (root) privileges. This introduces a security problem: If the system is compromised and that program is manipulated (as in the case of monolithic embedded devices), an attacker may be able to run additional executions as superuser (root).

Key Topic

Modern Unix and Linux-based systems ignore the SUID and SGID bits on shell scripts for this reason.

TIP An example of a SUID-based attack is the vulnerability that existed in the program /usr/lib/preserve (or /usr/lib/ex3.5preserve). This program, which is used by the vi and ex editors, automatically made a backup of the file being edited if the user was unexpectedly disconnected from the system before writing out changes to the file. The system wrote the changes to a temporary file in a special directory. The system also sent an email to the user using /bin/mail with a notification that the file had been saved. Because users could have been editing a file that was private or confidential, the directory used by the older version of the Preserve program was not accessible by most users on the system. Consequently, to let the Preserve program write into this directory and let the recovery program read from it, these programs were made SUID root.

You can find all the SUID and SGID files on your system by using the command shown in Example 7-6.

Example 7-6 Finding All the SUID and SGID Files on a System

```
omar@dionysus:~$ sudo find / \(-perm -004000 -o -perm -002000 \)
-type f -print
[sudo] password for omar: ****
find: '/proc/3491/task/3491/fdinfo/6':/usr/sbin/postqueue
/usr/sbin/postdrop
/usr/lib/eject/dmcrypt-get-device
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/x86_64-linux-gnu/utempter/utempter
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/snapd/snap-confine
/usr/lib/openssh/ssh-keysign
/usr/bin/dotlock.mailutils
/usr/bin/pkexec
/usr/bin/chfn
/usr/bin/screen
/usr/bin/newgrp
/usr/bin/crontab
/usr/bin/at
/usr/bin/chsh
/usr/bin/ssh-agent
/usr/bin/gpasswd
/usr/bin/expiry
```

```
/usr/bin/wall
/usr/bin/sudo
/usr/bin/bsd-write
/usr/bin/mlocate
/usr/bin/newgidmap
/usr/bin/chage
/usr/bin/newuidmap
find: '/proc/3491/fdinfo/5': No such file or directory
/sbin/mount.cifs
/sbin/unix_chpwd
/sbin/pam_extrousers_chpwd
/sbin/mount.ecryptfs_private
/bin/fusermount
/bin/ping6
/bin/mount
/bin/umount
/bin/ntfs-3g
/bin/su
/bin/ping
```

In Example 7-6, the **find** command starts in the root directory (/) and looks for all files that match mode 002000 (SGID) or mode 004000 (SUID). The **-type f** option limits the search to files only.

TIP Security Enhanced Linux (SELinux) is a collection of kernel modifications and user-space tools that are now part of several Linux distributions. It supports access control security policies, including mandatory access controls. SELinux aims to provide enforcement of security policies and simplify the amount of software required to accomplish such enforcement. Access can be constrained on variables such as which users and applications can access which resources. In addition, SELinux access controls are determined by a policy loaded on the system that cannot be changed by uneducated users or insecure applications. SELinux also allows you to configure more granular access control policies. For instance, SELinux lets you specify who can unlink, append only, or move a file instead of only being able to specify who can read, write, or execute a file. It also allows you to configure access to many other resources in addition to files. For example, it allows you to specify access to network resources and interprocess communication (IPC).



Insecure SUDO Implementations

Sudo, which stands for “super user do,” is a Linux utility that allows a system administrator to give certain users or groups of users the ability to run some or all commands as root or superuser. The Sudo utility operates on a per-command basis, and it is not a replacement for the shell. You can also use the Sudo utility to restrict the commands a user can run on a per-host basis, to restrict logging of each command to have an audit trail of who did what, and to restrict the ability to use the same configuration file on different systems.

Example 7-7 shows the Linux command **groups** being used. The command shows the group that the user omar belongs to. You can see in this example that sudo is one of the groups that the user omar belongs to.

Example 7-7 The **groups** Command

```
omar@dionysus:~$ groups
omar adm cdrom sudo dip plugdev lxd sambashare lpadmin
```

Another command you can use to see the groups a user belongs to is the **id** command, as shown in Example 7-8.

Example 7-8 The **id** Command

```
omar@dionysus:~$ id
uid=1000(omar) gid=1000(omar) groups=1000(omar),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),110(lxd),113(sambashare),117(lpadmin)
```

Example 7-9 shows the same commands used when a different user (ron) is logged in. In this case, you can see that ron belongs only to the group ron.

Example 7-9 The Groups to Which User ron Belongs

```
ron@dionysus:~$ groups
ron
ron@dionysus:~$ id
uid=1001(ron) gid=1001(ron) groups=1001(ron)
ron@dionysus:~$
```

Certain Linux systems call this group the “wheel” group. If you want to add an existing user to the wheel (or sudo) group, you can use the **usermod** command with the

-G option. You might also want to use the **-a** option, to avoid removing the user from other groups to which he or she belongs, as shown in Example 7-10.

Example 7-10 The **usermod** Command

```
$ sudo usermod -a -G wheel ron
```

You can also add a user account to the wheel group as you create it, as shown in Example 7-11.

Example 7-11 Adding a User to the wheel Group at Creation

```
$ sudo useradd -G wheel chris
```

In many different Linux systems, you can also use the **visudo** command. Figure 7-7 shows the first few lines of the description of the **visudo** man page (**man visudo**).

VISUDO(8) BSD System Manager's Manual VISUDO(8)

NAME
visudo – edit the sudoers file

SYNOPSIS
visudo [-chqsV] [-f sudoers] [-x output_file]

DESCRIPTION
visudo edits the sudoers file in a safe fashion, analogous to vipw(8). visudo locks the sudoers file against multiple simultaneous edits, provides basic sanity checks, and checks for parse errors. If the sudoers file is currently being edited you will receive a message to try again later.

There is a hard-coded list of one or more editors that visudo will use set at compile-time that may be overridden via the editor sudoers Default variable. This list defaults to /usr/bin/editor. Normally, visudo does not honor the VISUAL or EDITOR environment variables unless they contain an editor in the aforementioned editors list. However, if visudo is configured with the --with-env-editor option or the env_editor Default variable is set in sudoers, visudo will use any the editor defines by VISUAL or EDITOR. Note that this can be a security hole since it allows the user to execute any program they wish simply by setting VISUAL or EDITOR.

visudo parses the sudoers file after the edit and will not save the changes if there is a syntax error. Upon finding an error, visudo will print a message stating the line number(s) where the error occurred and the user will receive the “What now?” prompt. At this point the user may enter ‘e’ to re-edit the sudoers file, ‘x’ to exit without saving the changes, or ‘Q’ to quit and save changes. The ‘Q’ option should be used with extreme care because if visudo believes there to be a parse error, so will sudo and no one will be able to run sudo again until the error is fixed. If ‘e’ is typed to edit the sudoers file after a parse error has been detected, the cursor will be placed on the line where the error occurred (if the editor supports this feature).

Manual page visudo(8) line 1 (press h for help or q to quit)

FIGURE 7-7 The **visudo** Command Man Page

Example 7-12 shows the contents of the sudoers file after the **visudo** command is invoked.

Example 7-12 The sudoers File

```
# This file MUST be edited with the 'visudo' command as root.  
#  
# Please consider adding local content in /etc/sudoers.d/ instead of  
# directly modifying this file.  
#  
# See the man page for details on how to write a sudoers file.  
#  
Defaults env_reset  
Defaults mail_badpass  
Defaults secure_path="/usr/local/sbin:/usr/local/bin:/usr/  
sbin:/usr/bin:/sbin:/bin:/snap/bin"  
# Host alias specification  
  
# User alias specification  
  
# Cmnd alias specification  
  
# User privilege specification  
root    ALL=(ALL:ALL) ALL  
  
# Members of the admin group may gain root privileges  
%admin  ALL=(ALL) ALL  
  
# Allow members of group sudo to execute any command  
%sudo   ALL=(ALL:ALL) ALL  
  
# See sudoers(5) for more information on "#include" directives:  
#includedir /etc/sudoers.d
```

The first highlighted line in Example 7-12 means that the root user can execute commands from ALL terminals, acting as ALL (that is, any) users, and can run the ALL command (any commands). The second highlighted line specifies that members of the admin group may gain root privileges and can also execute commands from all terminals, acting as ALL (any) users, and can run the ALL command (any commands). The third highlighted line specifies the same for any members of the group sudo.

A huge mistake that some people make is to copy and paste the root privileges and assign them to a user, as shown in Example 7-13.

Example 7-13 Improper sudoers File Entry

```
ben    ALL=(ALL:ALL)  ALL
```

In Example 7-13 the user ben has been assigned all the privileges of root. Attackers can take advantage of misconfigured sudoers files, like this one, to cause severe negative effects on a system. In most cases, you probably want a specific user to power off the system or just execute certain commands that will be required for the user to do certain tasks. Example 7-14 shows a better setup than Example 7-13: Because ben only needs to be able to power off the system, he has only been given that sudo capability.

Example 7-14 Allowing ben to Power Off the System

```
ben  ALL= /sbin/poweroff
```

As demonstrated in Example 7-15, you can also create aliases for users (User_Alias), run commands as other users (Runas_Alias), specify the host or network from which they can log in (Host_Alias), and specify the command (Cmnd_Alias).

Example 7-15 sudoers File Using Aliases

```
User_Alias      COOLGUYS = ben, chris, ron
Runas_Alias     LESSCOOL = root, operator
Host_Alias      COOLNET = 192.168.78.0/255.255.255.0
Cmnd_Alias      PRINT = /usr/sbin/lpc, /usr/bin/lprm

omar ALL=(LESSCOOL)  ALL
# The user omar can run any command from any terminal as any user in
# the LESSCOOL group (root or operator).

trina COOLNET=(ALL)  ALL
# The user trina may run any command from any machine in the COOLNET
# network, as any user.

ben ALL=PRINT
# The user ben may run lpc and lprm from any machine.
```

In Example 7-15 the alias COOLGUYS includes the users ben, chris, and ron. The alias LESSCOOL includes the users root and operator. The alias COOLNET includes the network 192.168.78.0/24, and the command alias PRINT includes the commands lpc and lprm.

TIP Sudo has been affected by several vulnerabilities that allow users to overwrite system configurations, run additional commands that should not be authorized, among other things. You can stay informed of any new vulnerabilities in Sudo at <https://www.sudo.ws/security.html>.

Key Topic

Ret2libc Attacks

A “return-to-libc” (or ret2libc) attack typically starts with a buffer overflow. In this type of attack, a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the executable memory of the process. This is done to potentially bypassing the no-execute (NX) bit feature and allow the attacker to inject his or her own code.

Operating systems that support non-executable stack help protect against code execution after a buffer overflow vulnerability is exploited. On the other hand, non-executable stack cannot prevent a ret2libc attack because in this attack, only existing executable code is used. Another technique, called *stack-smashing protection*, can prevent or obstruct code execution exploitation because it can detect the corruption of the stack and can potentially “flush out” the compromised segment.

A technique called *ASCII armoring* can be used to mitigate ret2libc attacks. When you implement ASCII armoring, the address of every system library (such as libc) contains a NULL byte (0x00) that you insert in the first 0x01010101 bytes of memory. This is typically a few pages more than 16 MB and is called the *ASCII armor region* because every address up to (but not including) this value contains at least one NULL byte. When this methodology is implemented, an attacker cannot place code containing those addresses using string manipulation functions such as `strcpy()`.

Of course, this technique doesn’t protect the system if the attacker finds a way to overflow NULL bytes into the stack. A better approach is to use the address space layout randomization (ASLR) technique, which mitigates the attack on 64-bit systems. When you implement ASLR, the memory locations of functions are random. ASLR is not very effective in 32-bit systems, though, because only 16 bits are available for randomization, and an attacker can defeat such a system by using brute-force attacks.

Windows Privileges

The following sections cover several methodologies and attacks for performing privilege escalation in Windows systems.

CPassword

Key Topic

Legacy Windows operating systems were susceptible to CPassword attacks. CPassword was a component of Active Directory's Group Policy Preferences that allowed administrators to set passwords via Group Policy. Microsoft patched this vulnerability in MS14-025 (see <https://docs.microsoft.com/en-us/security-updates/securitybulletins/2014/ms14-025>). Microsoft also released a document explaining the vulnerability details, as well as well-known mitigations (see <https://support.microsoft.com/en-us/help/2962486/ms14-025-vulnerability-in-group-policy-preferences-could-allow-elevati>).

If administrators use CPassword to perform common tasks (such as changing the local administrator account), any user with basic read rights to the SYSVOL directory can obtain the authentication key and crack it by using tools such as John the Ripper and Hashcat.

TIP A CPassword attack is also referred to as a *GPP attack*. To test and find vulnerable systems, you can just perform a keyword search for “cpassword” through all the files in the SYSVOL directory and modify or remove any Group Policy Objects (GPOs) that reference them. A GPO is a virtual compilation of policy settings. Each GPO is configured with a unique name, such as a GUID. You can obtain more information about GPOs at [https://msdn.microsoft.com/en-us/library/aa374162\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa374162(v=vs.85).aspx). Microsoft has also published an article describing the SYSVOL implementation at <https://social.technet.microsoft.com/wiki/contents/articles/24160.active-directory-back-to-basics-sysvol.aspx>.

You can automatically decrypt passwords that are stored in the Group Policy Preferences by using Metasploit, and you can use the Meterpreter post-exploitation module to obtain and decrypt CPassword from files stored in the SYSVOL directory. In addition, a number of PowerShell scripts can be used to perform this type of attack, such as the ones at <https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Get-GPPPPassword.ps1>.



Clear-Text Credentials in LDAP

Unfortunately, many organizations still configure their Windows domain controllers to receive credentials in clear text over the network. One easy way to determine whether a system is affected by sending credentials in the clear is to look for event IDs 2886 and 2887 in the Active Directory Service log. Example 7-16 shows an example of Event 2886.

Example 7-16 Directory Service Event 2886

```
Log Name: Directory Service
Source: Microsoft-Windows-ActiveDirectory_DomainService
Date: 6/12/2018 3:08:11 AM
Event ID: 2886
Task Category: LDAP Interface
Level: Warning
Keywords: Classic
User: hacker
Computer: omar_workstation.sd.lan
Description:
The security of this directory server can be significantly enhanced by configuring the server to reject SASL (Negotiate, Kerberos, NTLM, or Digest) LDAP binds that do not request signing (integrity verification) and LDAP simple binds that are performed on a cleartext (non-SSL/TLS-encrypted) connection. Even if no clients are using such binds, configuring the server to reject them will improve the security of this server.

Some clients may currently be relying on unsigned SASL binds or LDAP simple binds over a non-SSL/TLS connection, and will stop working if this configuration change is made. To assist in identifying these clients, if such binds occur this directory server will log a summary event once every 24 hours indicating how many such binds occurred. You are encouraged to configure those clients to not use such binds. Once no such events are observed for an extended period, it is recommended that you configure the server to reject such binds.
```

If any domain controller has the 2886 event present, this indicates that LDAP signing is not being enforced by the domain controller, and it is possible to perform a simple (clear-text) LDAP bind over a non-encrypted connection.

TIP The tool at <https://github.com/russelltomkins/Active-Directory/blob/master/Query-InsecureLDAPBinds.ps1> can be used to query logs for insecure LDAP binds and clear-text passwords. Furthermore, the following post includes additional information about how such an attack could be performed: <https://www.harmj0y.net/blog/powershell/kerberoasting-without-mimikatz>.

Kerberoasting

Key Topic

Kerberoast is a series of tools for attacking Microsoft Kerberos implementations and Windows service accounts. The tool can be obtained from <https://github.com/nidem/kerberoast>.

TIP The post <https://www.blackhillsinfosec.com/a-toast-to-kerberoast/> provides step-by-step instructions for remotely running a Kerberoast attack over an established Meterpreter session to a command and control server and cracking the ticket offline using Hashcat.

You will learn more about Meterpreter and Hashcat in Chapter 9, “Penetration Testing Tools.”

Credentials in Local Security Authority Subsystem Service (LSASS)

Key Topic

Another attack commonly performed against Windows systems involves obtaining user and application credentials from the Local Security Authority Subsystem Service (LSASS). It is possible to dump the LSASS process from memory to disk by using tools such as Sysinternals ProcDump. Attackers have been successful using ProcDump because it is a utility digitally signed by Microsoft. Therefore, this type of attack can evade many antivirus programs. ProcDump creates a minidump of the target process. An attacker can then use tools such as Mimikatz to mine user credentials.

TIP You can use the VMware tool vmss2core to dump memory from a suspended virtual machine (VM). You can easily identify a suspended VM by the file extension .vmss. You can also use the VMware tool vmss2core to dump memory from snapshotted VMs (*.vmsn). You can then use the Volatility Framework to extract the hashes. For more information about the Volatility Framework, see <http://www.volatilityfoundation.org>.

The following are additional resources related to the aforementioned attacks:

- **ProcDump and Windows Sysinternals:** <https://docs.microsoft.com/en-us/sysinternals/downloads/procdump>
- **Mimikatz:** <http://blog.gentilkiwi.com/mimikatz>
- **The Volatility Foundation:** <http://www.volatilityfoundation.org>
- **Vmss2core:** <https://labs.vmware.com/flings/vmss2core>
- **VMware Snapshot and Saved State Analysis:** <http://volatility-labs.blogspot.be/2013/05/movp-ii-13-vmware-snapshot-and-saved.html>

SAM Database



Microsoft Active Directory plays an important role in many organizations. Active Directory provides a directory service for managing and administering different domain activities. Active Directory is based on a client/server architecture. Understanding how Active Directory works and the underlying architecture is very important for any pen tester tasked with testing Windows environments.

Of course, one of the common tasks in a penetration testing engagement is to retrieve passwords from a Windows system and ultimately try to get domain administrator access. In Chapter 5, “Exploiting Wired and Wireless Networks,” you learned about the pass-the-hash attack technique and other attacks against Windows systems. As a refresher, Windows stores password hashes in three places:

- The Security Account Manager (SAM) database
- The LSASS
- The Active Directory database

All versions of Windows store passwords as hashes, in a file called the Security Accounts Manager (SAM) database.

NOTE The SAM database stores only hashes the passwords. Windows itself does not know what the passwords are.

The SAM database stores usernames and NT hashes in a %SystemRoot%/system32/config/SAM file. This file contains all the hash values for accounts that are local to the computer.

Microsoft created its own hash process for its Windows operating systems. This is where the NT LAN Manager (NTLM) comes into play. NTLM is a suite of

Microsoft security protocols that have been proven to be vulnerable and used by many penetration testers as well as threat actors to compromise machines. Because password hashes cannot be reversed, instead of trying to figure out a user's password, you (or an attacker) can just use a password hash collected from a compromised system and then use the same hash to log in to another client or server system. This technique, called *pass-the-hash*, is illustrated in Figure 7-8.

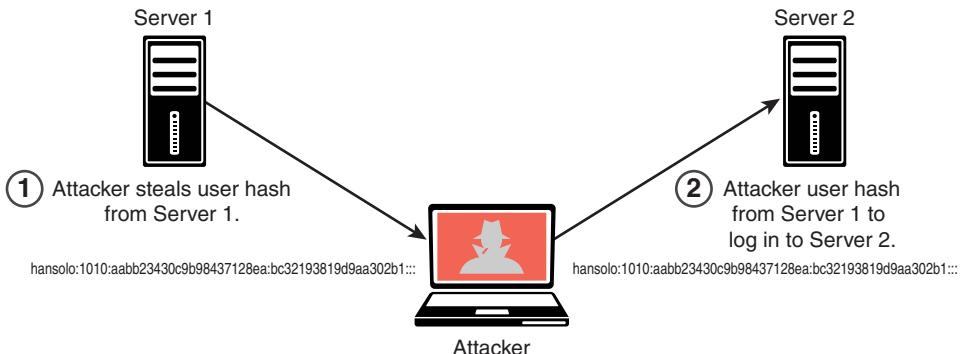


FIGURE 7-8 Pass-the-Hash Attack Example

Microsoft now uses Kerberos in Windows domains, but NTLM is still used when the client is authenticating to a server in a different Active Directory forest that has a legacy NTLM trust instead of a transitive inter-forest trust. NTLM is also used when the client is authenticating to a server that doesn't belong to a domain and when Kerberos is blocked by a firewall or a similar device.

Understanding Dynamic Link Library Hijacking

Key Topic

Dynamic link libraries (DLLs) are common components in all versions of Windows. Some DLLs are loaded into applications when they start (if needed). DLLs interact with APIs and other operating system procedures. If you tamper with a system in order to control which DLL an application loads, you may be able to insert a malicious DLL during the DLL loading process to compromise the system. An application can decide the order of the directories to be searched for a DLL to load, depending on the configuration of the system. The following list shows the order of the Windows DLL search process:

- Step 1.** Windows searches the working directory from which the application is loaded.
- Step 2.** Windows searches the current directory (from which the user is working).

- Step 3.** Windows searches the system directory (typically \Windows\System32\). The **GetSystemDirectory** function is called to obtain this directory.
- Step 4.** Windows searches the 16-bit system directory.
- Step 5.** Windows searches the Windows directory. The **GetWindowsDirectory** function is called to obtain this directory.
- Step 6.** Windows searches directories that are listed in the PATH environment variable.

In this process, the attack relies on a program making a decision to load a DLL from the current directory (step 2). An attacker can manipulate that step and perform a DLL hijacking attack. For instance, if the user is opening an Excel spreadsheet, Microsoft Office attempts to load its DLL component from the location of that document file. An attacker can put a malicious DLL in that directory. Subsequently, Microsoft Office can carelessly load the malicious DLL.

TIP DLL hijack attacks are not as effective as they used to be. This is because Microsoft has released several patches and features that help prevent these types of attacks. The following article explains some of the mitigations: <https://docs.microsoft.com/en-us/windows/desktop/dlls/dynamic-link-library-search-order>.



Exploitable Services

You as a pen tester can take advantage of exploitable services such as the following:

- **Unquoted service paths:** If an executable (application binary) is enclosed in quotation marks (""), Windows knows where to find it. On the contrary, if the path where the application binary is located doesn't contain any quotation marks, Windows will try to locate it and execute it inside every folder of this path until it finds the executable file. An attacker can abuse this functionality to try to elevate privileges if the service is running under SYSTEM privileges. A service is vulnerable if the path to the executable has a space in the filename and the filename is not wrapped in quotation marks; exploitation requires write permissions to the path before the quotation mark.
- **Writable services:** Administrators often configure Windows services that run with SYSTEM privileges. This could lead to a security problem because an attacker may obtain permissions over the service or over the folder where

the binary of the service is stored (or both). Services configured this way are also often found in third-party software (TPS) and may be used for privilege escalation.

Insecure File and Folder Permissions



An attacker can take advantage of unsecured and misconfigured file and folder permissions. Files and folders in Windows can have read and write permissions. These permissions are established strictly to specific users or groups. In contrast, Unix and Linux-based systems grant file and folder permissions to the owner, the group owner, or everybody. Windows uses specific permissions to allow users to access folder content. Windows does not use execute permissions on files. Windows uses the filename extension to determine whether a file (including a script file) can be run.

TIP For details on how Windows file security and access rights work, see <https://docs.microsoft.com/en-us/windows/desktop/fileio/file-security-and-access-rights>. Microsoft has also published a detailed document explaining Windows access control lists at <https://docs.microsoft.com/en-us/windows/desktop/secauthz/access-control-lists>.

Table 7-2 compares the permissions between Unix/Linux systems and Windows.

Table 7-2 A Comparison Between Permissions for Unix/Linux-Based Systems and Windows Systems

Unix/Linux	Windows
Read and write permissions on a folder in Unix is the same as the read and write permissions in Windows.	
The read and execute permissions on a file in Unix are the same as the read and execute permissions in Windows.	
Write permission on a file	Modify permission on a file
Execute permission on a folder	List Folder Contents permission
Read, write, and execute permissions on a file or folder	Full Control permission

Understanding Windows Group Policy

In Windows, Group Policy is a centralized administration feature for systems belonging to a Windows domain. This functionality allows you to create policies in Active Directory and assign them to users or systems. You create policies to

configure specific settings and permissions within the Windows operating system. The item inside Active Directory that contains these settings is called a Group Policy Object (GPO). GPOs can be used for user accounts, for client computer settings, or for configuring policies in servers. Typically, the goal is to configure GPOs in such a way that they cannot be overridden by users.

TIP Microsoft provides a series of spreadsheets and other documentation to help manage GPOs; see <http://www.microsoft.com/en-us/download/details.aspx?id=25250>. These spreadsheets list the policy settings for computer and user configurations that are included in the Administrative template files delivered with the specified Windows operating system. You can configure these policy settings when you edit GPOs. A brief example of one of these spreadsheets is shown in Figure 7-9.

C1	A	B	C	
1	Status	Policy Path	Policy Name	Supported O
2		Computer Configuration\Windows Settings\Account Policies\Password Policy	Enforce password history	At least Windows
3		Computer Configuration\Windows Settings\Account Policies\Password Policy	Maximum password age	At least Windows
4		Computer Configuration\Windows Settings\Account Policies\Password Policy	Minimum password age	At least Windows
5		Computer Configuration\Windows Settings\Account Policies\Password Policy	Minimum password length	At least Windows
6		Computer Configuration\Windows Settings\Account Policies\Password Policy	(Password must meet complexity requirement)	At least Windows
7		Computer Configuration\Windows Settings\Account Policies\Password Policy	Store passwords using reversible encryption for all users in the domain	At least Windows
8		Computer Configuration\Windows Settings\Account Policies\Account Lockout Policy	Account lockout duration	At least Windows
9		Computer Configuration\Windows Settings\Account Policies\Account Lockout Policy	Account lockout threshold	At least Windows
10		Computer Configuration\Windows Settings\Account Policies\Account Lockout Policy	Reset lockout counter after	At least Windows
11		Computer Configuration\Windows Settings\Local Policies\Kerberos Policy	Enforce user logon restrictions	At least Windows
12		Computer Configuration\Windows Settings\Local Policies\Kerberos Policy	Maximum lifetime for service ticket	At least Windows
13		Computer Configuration\Windows Settings\Local Policies\Kerberos Policy	Maximum lifetime for user ticket	At least Windows
14		Computer Configuration\Windows Settings\Local Policies\Kerberos Policy	Maximum lifetime for user ticket renewal	At least Windows
15		Computer Configuration\Windows Settings\Local Policies\Kerberos Policy	Maximum tolerance for computer clock synchronization	At least Windows
16		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit account logon events	At least Windows
17		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit account management	At least Windows
18		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit directory service access	At least Windows
19		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit logon events	At least Windows
20		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit object access	At least Windows
21		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit policy change	At least Windows
22		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit privilege use	At least Windows
23		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit process tracking	At least Windows
24		Computer Configuration\Windows Settings\Local Policies\Audit Policy	Audit system events	At least Windows
25		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Access this computer from the network	At least Windows
26		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Access Credential Manager as a trusted caller	At least Windows
27		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Act as part of the operating system	At least Windows
28		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Add workstations to a domain	At least Windows
29		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Adjust memory quotas for a process	At least Windows
30		Computer Configuration\Windows Settings\Local Policies\User Rights Assignment	Allow log on locally	At least Windows

FIGURE 7-9 Group Policy Settings Reference for Windows and Windows Server

Keyloggers

An attacker may use a keylogger to capture every key stroke of a user in a system and steal sensitive data (including credentials). There are two main types of keyloggers: keylogging hardware devices and keylogging software. A hardware (physical) keylogger is usually a small device that can be placed between a user's keyboard and the main system. Software keyloggers are dedicated programs designed to track and log user keystrokes.

NOTE Keyloggers are legal in some countries and designed to allow employers to oversee the use of their computers. However, recent regulations like GDPR have made keyloggers a very sensitive and controversial topic. Threat actors use keyloggers for the purpose of stealing passwords and other confidential information.

There are several categories of software-based keyloggers:

- **Kernel-based keylogger:** A program on the machine obtains root access to hide itself in the operating system and intercepts keystrokes that pass through the kernel. This method is difficult both to write and to combat. Such keyloggers reside at the kernel level, which makes them difficult to detect, especially for user-mode applications that don't have root access. They are frequently implemented as rootkits that subvert the operating system kernel to gain unauthorized access to the hardware. This makes them very powerful. A keylogger using this method can act as a keyboard device driver, for example, and thus gain access to any information typed on the keyboard as it goes to the operating system.
- **API-based keylogger:** With this type of keylogger, compromising APIs reside inside a running application. Different types of malware have taken advantage of Windows APIs, such as `GetAsyncKeyState()` and `GetForegroundWindow()`, to perform keylogging activities.
- **Hypervisor-based keylogger:** This type of keylogger is effective in virtual environments, where the hypervisor could be compromised to capture sensitive information.
- **Web form-grabbing keylogger:** Keyloggers can steal data from web form submissions by recording the web browsing on submit events.
- **JavaScript-based keylogger:** Malicious JavaScript tags can be injected into a web application and then capture key events (for example, the `onKeyUp()` JavaScript function).
- **Memory-injection-based keylogger:** This type of keylogger tampers with the memory tables associated with the browser and other system functions.

Scheduled Tasks



Threat actors can take advantage of the Windows Task Scheduler to bypass User Account Control (UAC) if the user has access to its graphical interface. This is possible because the security option runs with the system's highest privileges. When a Windows user creates a new task, the system typically doesn't require the user to

authenticate with an administrator account. You can also use this functionality for post-exploitation and persistence.

NOTE You can access the scheduled tasks of a Windows system by navigating to **Start -> Programs -> Accessories -> System Tools -> Scheduled Tasks**.



Escaping the Sandbox

The term *sandbox* can mean different things depending on to the field. In cybersecurity, a sandbox allows you to isolate running applications to minimize the risk of software vulnerabilities spreading from one application to another. Figure 7-10 illustrates this sandboxing concept.

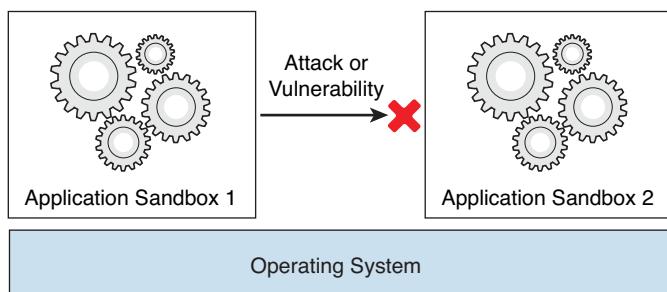


FIGURE 7-10 Sandboxes

Sandboxes can also be used to run untested or untrusted software from unverified or untrusted third parties, suppliers, users, or websites. In addition, they can be used to test malware without allowing the software to compromise the host system.

TIP Sandbox implementations typically operate and provide a controlled set of resources for guest applications to run in. These resources include a “scratch space” on disk and memory. Typically, network access is disallowed or highly restricted.

In web development, a sandbox is a mirrored production environment that developers use to create an application before migrating it to a production environment. Companies like Amazon, Google, and Microsoft, among others, provide sandboxing services.

NOTE For the purpose of this book, we of course concentrate on sandboxes related to cybersecurity.

The following are examples of sandbox implementations:

- **A jail:** This implementation is commonly used in mobile devices where there is restricted filesystem namespace and rule-based execution to not allow untrusted applications to run in the system. This is where the term jail-breaking comes in. Users may “jail-break” their phones to be able to install games and other applications. With a jail-broken phone, an attacker can more easily impersonate applications and deliver malware to the user because a jail-broken device does not have the security controls in place to prevent malware from running on the system.
- **Rule-based execution in SELinux and AppArmor security frameworks:** This implementation restricts control over what processes are started, spawned by other applications, or allowed to inject code into the system. These implementations can control what programs can read and write to the file system.
- **Virtual machines:** Virtual machines can be used to restrict a guest operating system to run sandboxed so that the applications do not run natively on the host system and can only access host resources through the hypervisor.
- **Sandboxing on native hosts:** Security researchers may use sandboxing to analyze malware behavior. Even commercial solutions such as Cisco’s ThreatGrid use sandbox environments that mimic or replicate the victim system to evaluate how malware infects and compromises such a system.
- **Secure Computing Mode (seccomp) and seccomp-bpf (seccomp extension):** These are sandboxes built in the Linux kernel to only allow the `write()`, `read()`, `exit()`, and `sigreturn()` system calls.
- **Software fault isolation (SFI):** This implementation uses sandboxing methods in all store, read, and jump assembly instructions to isolated segments of memory.
- **Web browsers:** Browsers provide sandboxing capabilities to isolate extensions and plugins.
- **HTML5:** HTML5 has a `sandbox` attribute for use with iframes.
- **Java virtual machines:** These VMs include a sandbox to restrict the actions of untrusted code, such as a Java applet.
- **.NET Common Language Runtime:** This implementation enforces restrictions on untrusted code.
- **Adobe Reader:** This implementation runs PDF files in a sandbox to prevent them from escaping the PDF viewer and tampering with the rest of the computer.
- **Microsoft Office:** Office has a sandbox mode to prevent unsafe macros from harming the system.

If an attacker finds a way to bypass (escape) the sandbox, he or she can then compromise other applications and potentially implement a full system compromise. Several sandbox escape vulnerabilities in the past have allowed attackers to do just that.

Key Topic

Virtual Machine Escape

In the previous section, you learned that VMs can be used to restrict a guest operating system to run sandboxed. This is because the applications do not run natively on the host system and can only access host resources through the hypervisor.

If an attacker finds a way to escape the VM, he or she can then compromise other VMs and potentially compromise the hypervisor. This is catastrophic in cloud environments, where multiple customers can be affected by these types of attacks. A VM escape attack is illustrated in Figure 7-11.

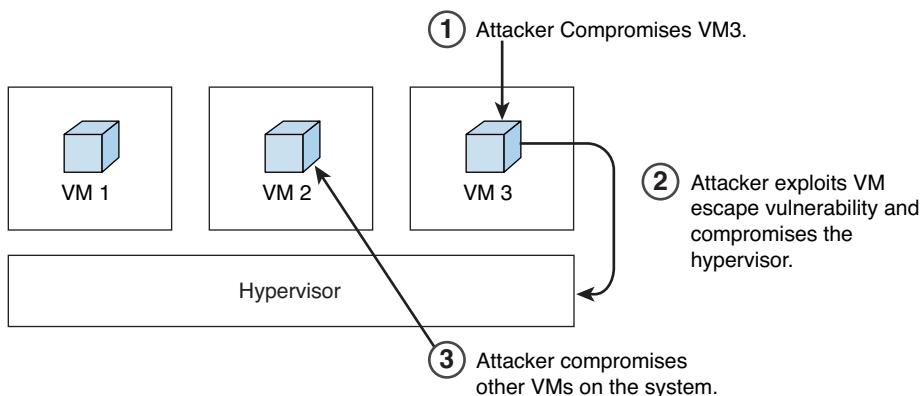


FIGURE 7-11 VM Escape

Understanding Container Security

A lot of people immediately think about Docker when they hear the word *containers*, but there are other container technologies out there. Linux Containers (LXC) is a well-known set of tools, templates, and library and language bindings for Linux containers. It's pretty low level and very flexible, and it covers just about every containment feature supported by the upstream kernel.

NOTE You can learn more about LXC at <https://linuxcontainers.org>.

Docker is really an extension of LXC's capabilities. A high-level API provides a lightweight virtualization solution to run different processes in isolation. Docker was developed in the Go language and utilizes LXC, cgroups, and the Linux kernel itself.

NOTE You can learn more about Docker at <https://www.docker.com>.

Another popular container technology or package is rkt (or Rocket). rkt aims to provide a feature and capability that its creators call “secure-by-default.” It includes a number of security features such as support for SELinux, TPM measurement, and running app containers in hardware-isolated VMs.

NOTE You can learn more about Rocket at <https://github.com/rkt/rkt>.

Cri-o is a lightweight container technology used and designed with Kubernetes. It provides support for containers based on the Open Container Initiative specifications (see <https://www.opencontainers.org>), a set of two specifications: the Runtime Specification (`runtime-spec`) and the Image Specification (`image-spec`). The `runtime-spec` outlines how to run a filesystem bundle that is unpacked on disk.

NOTE You can learn more about Cri-o at <http://cri-o.io>.

Another container package is called OpenVz. It is not as popular as Docker or Rocket, but it is making the rounds.

NOTE You can learn more about OpenVz at <https://openvz.org>.

What is a container? A container image is a lightweight, standalone, executable package of a piece of software that includes everything you need to run it, including code, the runtime, system tools, system libraries, and settings. Containers are available for Linux, Mac OS X, and Windows applications.

NOTE Containerized software will always run the same, regardless of the environment.

Containers isolate software from its surroundings and help reduce conflicts between teams running different software on the same infrastructure.

So what is the difference between a container and a virtual machine? Figure 7-12 provides a comparison.

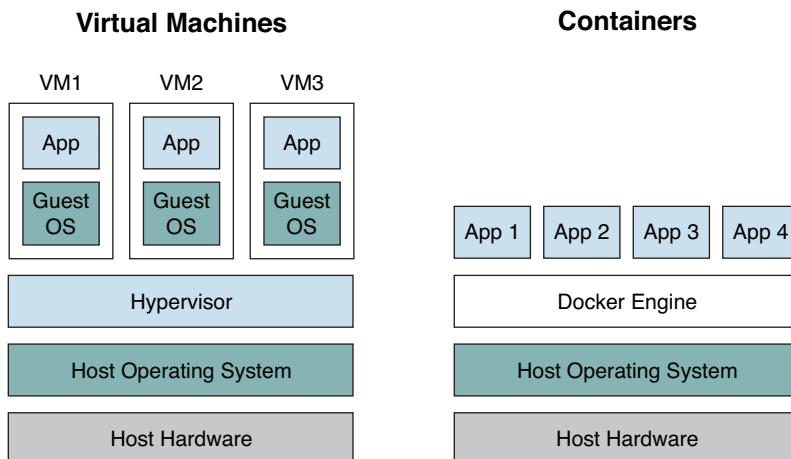


FIGURE 7-12 VMs vs. Containers

Figure 7-12 shows the architectural differences between container and VM environments. A VM generally includes an entire operating system along with the application. It also needs a hypervisor running along with it to control the VM. VMs tend to be pretty big in size, since they include whole operating systems. Because of this, they take up several minutes to boot up the operating system and initialize the application they are hosting. Containers are much smaller; they perform much better than VMs and can start almost instantly.

One of the biggest advantages of container technologies is that containers can be created much faster than VM instances. Their lightweight footprint means less overhead in terms of performance and size. Containers increase developer productivity by removing cross-service dependencies and conflicts. Each container can be seen as a different microservice, and you can very easily upgrade them independently.

Each image of a container can be version controlled, so you can track different versions of a container. Containers encapsulate all the relevant details, such as application dependencies and operating systems. This makes them extremely portable across systems.

Docker and container technologies are supported by all major cloud providers, including Amazon Web Services (AWS), Google Cloud Platform, and Microsoft Azure. In addition, Docker can be integrated with tools like Ansible, Chef, Puppet, Jenkins, Kubernetes, OpenStack, Vagrant, and dozens of other tools and infrastructures.



TIP Of course, this is not a book about Docker and containers. However, if you have never played with containers, you can easily download your favorite Linux distribution and install Docker. For example, in Ubuntu or even Kali Linux, you can simply install Docker with the `apt install docker.io` command.

Some of the most challenging issues with containers and DevOps are operational in nature. For example, due to the convenience and agility that containers bring to the table, developers often pull Docker containers from community repositories and stores not knowing what vulnerabilities they are inheriting in those containers. Asset discovery and container vulnerability management are therefore very important.

The following are a few examples of tools and solutions that have been developed throughout the years for container security:

- **Anchore:** Anchore is used to analyze container images for the presence of known security vulnerabilities and against custom security policies. It has both open source and commercial versions. You can obtain the open source code and more information about it from <https://github.com/anchore/anchore-engine>.
- **Aqua Security:** This is a commercial tool for securing container-based applications (see <https://www.aquasec.com>).
- **Bane:** This is an AppArmor profile generator for Docker containers. You can download it from <https://github.com/genuine-tools/bane>.
- **CIS Docker Benchmark:** This tool provides an automated way to test containers against well-known security best practices. You can download the CIS Docker Benchmark from <https://github.com/dev-sec/cis-docker-benchmark>.
- **Dev-Sec.io:** This tool allows you to automatically apply hardening best practices to different types of servers (see <https://dev-sec.io>).
- **Clair:** This is an open source static analysis for Docker containers from Core-OS. You can download Clair from <https://github.com/coreos/clair>.
- **Dagda:** This is another tool for performing static analysis of known vulnerabilities. You can download Dagda from <https://github.com/eliasgranderubio/dagda>.
- **docker-bench-security:** This script, created by Docker, checks for common security best practices when deploying Docker containers in production. You can download this tool from <https://github.com/docker/docker-bench-security>.

- **docker-explorer:** This tool was created by Google to help analyze offline Docker file systems. It can be useful when performing forensic analysis of Docker containers. You can download it from <https://github.com/google/docker-explorer>.
- **Notary:** This open source project includes a server and a client for running and interacting with trusted containers. Notary is maintained by The Update Framework (TUF). You can obtain more information about Notary from <https://github.com/theupdateframework/notary> and information about TUF from <https://theupdateframework.github.io>.
- **oscap-docker:** OpenSCAP (created by RedHat) includes the oscap-docker tool, which is used to scan Docker containers and images. OpenSCAP and the oscap-docker tool can be downloaded from <https://github.com/OpenSCAP/openscap>.



Mobile Device Security

Mobile device security is a hot topic today. Individuals and organizations are increasingly using mobile devices for personal use and to conduct official business. Because of this, the risk in mobile devices and applications continues to increase.

The OWASP organization created the Mobile Security Project to provide mobile application and platform developers, as well as security professionals, resources to understand cybersecurity risks and to build and maintain secure mobile applications. The OWASP Mobile Security Project website can be accessed at https://www.owasp.org/index.php/OWASP_Mobile_Security_Project.

OWASP often performs studies of the top mobile security threats and vulnerabilities. According to OWASP, the top 10 mobile security risks at the time of this writing are:

- Improper platform usage
- Insecure data storage
- Insecure communication
- Insecure authentication
- Insufficient cryptography
- Insecure authorization
- Client code quality
- Code tampering
- Reverse engineering
- Extraneous functionality

Mobile applications (apps) run either directly on a mobile device, on a mobile device web browser, or both. Mobile operating systems (such as Android and Apple iOS) offer software development kits (SDKs) for developing applications (such as those for games, productivity, business, and more). These mobile apps, referred to as *native apps*, typically provide the fastest performance with the highest degree of reliability and adhere to platform-specific design principles.

Mobile web apps are basically websites designed to look and feel like native apps. These apps are accessed by a user via a device's browser and are usually developed in HTML5 and responsive mobile frameworks. Another option, a hybrid app, executes like a native app, but a majority of its processes rely on web technologies.

A lot of attacks against mobile apps start with reverse engineering and then move into tampering with the mobile app. Reverse engineering involves analyzing the compiled app to extract information about its source code. The goal of reverse engineering is to understand the underlying code and architecture. Tampering is the process of changing a mobile app (either the compiled app or the running process) or its environment to affect its behavior. In order to perform good reverse engineering of mobile apps, you should become familiar with the mobile device processor architecture, the app executable format, and the programming language used to develop a mobile app.

Modern apps often include controls that hinder dynamic analysis. Certificate pinning and end-to-end (E2E) encryption sometimes prevent you from intercepting or manipulating traffic with a proxy. Root detection could prevent an app from running on a rooted device, preventing you from using advanced testing tools.

NOTE Mobile apps that implement the protections specified in the Mobile AppSec Verification Standard (MASVS) Anti-Reversing Controls should withstand reverse engineering to a certain degree. Details about MASVS can be accessed at https://www.owasp.org/images/6/61/MASVS_v0.9.4.pdf.

There are a few basic tampering techniques:

- **Binary patching (“modding”):** This involves changing the compiled app in binary executables or tampering with resources. Modern mobile operating systems such as iOS and Android enforce code signing to mitigate binary tampering.
- **Code injection:** This allows you to explore and modify processes at runtime. Several tools, including Cydia Substrate (<http://www.cydiasubstrate.com>), Frida (<https://www.frida.re>), and XPosed (<https://github.com/rovo89/XposedInstaller>), give you direct access to process memory and important structures such as live objects instantiated by the app.

- **Static and dynamic binary analysis:** This is done using disassemblers and decompilers to translate an app's binary code or bytecode back into a more understandable format. By using these techniques on native binaries, you can obtain assembler code that matches the architecture for which the app was compiled.
- **Debugging and tracing:** It is possible to identify and isolate problems in a program as part of the software development life cycle. The same tools used for debugging are valuable to reverse engineers even when identifying bugs is not their primary goal. Debuggers enable program suspension at any point during runtime, inspection of the process's internal state, and even register and memory modification.



Understanding Android Security

Android is a Linux-based open source platform developed by Google as a mobile operating system. Android is not only used in mobile phones and tablets but also in wearable products, TVs, and many other smart devices. Android-based solutions come with many pre-installed (“stock”) apps and support installation of third-party apps through the Google Play store and other marketplaces.

Android's software stack is composed of several different layers (see <https://source.android.com/devices/architecture>). Each layer defines interfaces and offers specific services. At the lowest level, Android is based on a variation of the Linux kernel. On top of the kernel, the Hardware Abstraction Layer (HAL) defines a standard interface for interacting with built-in hardware components. Several HAL implementations are packaged into shared library modules that the Android system calls when required. This is how applications interact with the device's hardware (for instance, how a phone uses the camera, microphone, and speakers).

Android apps are usually written in Java and compiled to Dalvik bytecode, which is somewhat different from the traditional Java bytecode. The current version of Android executes this bytecode on the Android runtime (ART). ART is the successor to Android's original runtime, the Dalvik virtual machine. The key difference between Dalvik and ART is the way the bytecode is executed (see <https://source.android.com/devices/tech/dalvik/>).

Android apps do not have direct access to hardware resources, and each app runs in its own sandbox (see <https://source.android.com/security/app-sandbox>). The Android runtime controls the maximum number of system resources allocated to apps, preventing any one app from monopolizing too many resources.

Even though the Android operating system is based on Linux, it doesn't implement user accounts in the same way other Unix-like systems do. In Android, the multiuser support of the Linux kernel extends to sandbox apps: With a few exceptions, each app runs as though under a separate Linux user, effectively isolated from other apps and the rest of the operating system.

TIP The file `android_filesystem_config.h` includes a list of the predefined users and groups to which system processes are assigned. User IDs (UIDs) for other applications are added as they are installed.

Android apps interact with system services such as the Android Framework and related APIs. Most of these services are invoked via normal Java method calls and are translated to IPC calls to system services that are running in the background. Examples of system services include the following:

- Network connectivity, including Wi-Fi, Bluetooth, and NFC
- Cameras
- Geolocation (GPS)
- Device microphone

The framework also offers common security functions, such as cryptography.

The Android Package Kit (APK) file is an archive that contains the code and resources required to run the app it comes with. This file is identical to the original signed app package created by the developer. The installed Android apps are typically located at `/data/app/[package-name]`.

The following are some key Android files:

- **AndroidManifest.xml:** This file contains the definition of the app's package name, target, and minimum API version, app configuration, components, and user-granted permissions.
- **META-INF:** This file contains the application's metadata and the following three files:
 - **MANIFEST.MF:** This file stores hashes of the app resources.
 - **CERT.RSA:** This file stores the app's certificate(s).
 - **CERT.SF:** This file lists resources and the hash of the corresponding lines in the MANIFEST.MF file.

- **assets:** This directory contains app assets (files used within the Android app, such as XML files, JavaScript files, and pictures), which the AssetManager can retrieve.
- **classes.dex:** This directory contains classes compiled in the DEX file format that the Dalvik virtual machine/Android runtime can process. DEX is Java bytecode for the Dalvik virtual machine, and it is optimized for small devices.
- **lib:** This directory contains native compiled libraries that are part of the APK, such as the third-party libraries that are not part of the Android SDK.
- **res:** This directory contains resources that haven't been compiled into resources.arsc.
- **resources.arsc:** This file contains precompiled resources, such as XML files for layout.

AndroidManifest.xml is encoded into binary XML format, which is not readable with a text editor. However, you can unpack an Android app by using Apktool. When you run Apktool with the default command-line flags, it automatically decodes the manifest file to text-based XML format and extracts the file resources. The following are the typical decoded and extracted files:

- **AndroidManifest.xml:** This is the decoded manifest file, which can be opened and edited in a text editor.
- **apktool.yml:** This file contains information about the output of Apktool.
- **original:** This folder contains the MANIFEST.MF file, which stores information about the files contained in the JAR file.
- **res:** This directory contains the app's resources.
- **smalidea:** This is a Smali language plugin. Smali is a human-readable representation of the Dalvik executable. Every app also has a data directory for storing data created during runtime. Additional information about smalidea can be obtained from <https://github.com/JesusFreke/smali/wiki/smalidea>.
- **cache:** This location is used for data caching. For example, the WebView cache is found in this directory.
- **code_cache:** This is the location of the file system's application-specific cache directory that is designed for storing cached code. On devices running Lollipop or later Android versions, the system deletes any files stored in this location when the app or the entire platform is upgraded.
- **databases:** This folder stores SQLite database files generated by the app at runtime (for example, user data files).

- **files:** This folder stores regular files created by the app.
- **lib:** This folder stores native libraries written in C/C++. These libraries can have one of several file extensions, including .so and .dll (x86 support). This folder contains subfolders for the platforms for which the app has native libraries, including the following:
 - **armeabi:** Compiled code for all ARM-based processors
 - **armeabi-v7a:** Compiled code for all ARM-based processors, version 7 and above only
 - **arm64-v8a:** Compiled code for all 64-bit ARM-based processors, version 8 and above only
 - **x86:** Compiled code for x86 processors only
 - **x86_64:** Compiled code for x86_64 processors only
 - **mips:** Compiled code for MIPS processors
- **shared_prefs:** This folder contains an XML file that stores values saved via the SharedPreferences APIs.

Android leverages Linux user management to isolate apps. This approach is different from user management in traditional Linux environments, where multiple apps are often run by the same user. Android creates a unique UID for each Android app and runs the app in a separate process. Consequently, each app can access its own resources only. This protection is enforced by the Linux kernel. Typically, apps are assigned UIDs in the range 10000 and 19999. An Android app receives a user name based on its UID. For example, the app with UID 10188 receives the username u0_a188. If the permissions an app requested are granted, the corresponding group ID is added to the app's process. For example, the user ID of the app in this example is 10188. It belongs to the group ID 3003 (inet). That group is related to the android.permission.INTERNET permission in the application manifest.

Apps are executed in the Android Application Sandbox, which separates the app data and code execution from other apps on the device. This separation adds a layer of security. Installation of a new app creates a new directory named after the app package (for example, /data/data/[package-name]). This directory holds the app's data. Linux directory permissions are set such that the directory can be read from and written to only with the app's unique UID.

The process Zygote starts up during Android initialization. Zygote is a system service for launching apps. The Zygote process is a base process that contains all the core libraries the app needs. Upon launch, Zygote opens the socket /dev/socket/

zygote and listens for connections from local clients. When it receives a connection, it forks a new process, which then loads and executes the app-specific code.

In Android, the lifetime of an app process is controlled by the operating system. A new Linux process is created when an app component is started and the same app doesn't yet have any other components running. Android may kill this process when the process is no longer necessary or when it needs to reclaim memory to run more important apps. The decision to kill a process is primarily related to the state of the user's interaction with the process.

Android apps are made of several high-level components, including the following:

- Activities
- Fragments
- Intents
- Broadcast receivers
- Content providers and services

All these elements are provided by the Android operating system, in the form of pre-defined classes available through APIs.

TIP During development, an app is signed with an automatically generated certificate. This certificate is inherently insecure and is for debugging only. Most stores don't accept this kind of certificate for publishing; therefore, a certificate with more secure features must be created. When an application is installed on the Android device, PackageManager ensures that it has been signed with the certificate included in the corresponding APK. If the certificate's public key matches the key used to sign any other APK on the device, the new APK may share a UID with the preexisting APK. This facilitates interactions between applications from a single vendor. Alternatively, specifying security permissions for the Signature protection level is possible; this restricts access to applications that have been signed with the same key.

To perform detailed analysis of Android applications, you can download Android Studio. It comes with the Android SDK, an emulator, and an app to manage the various SDK versions and framework components. Android Studio also comes with the Android Virtual Device (AVD) Manager application for creating emulator images. You can download Android Studio from <https://developer.android.com/studio>.

Figure 7-13 shows a screenshot of an application called OmarsApplication being developed using Android Studio.

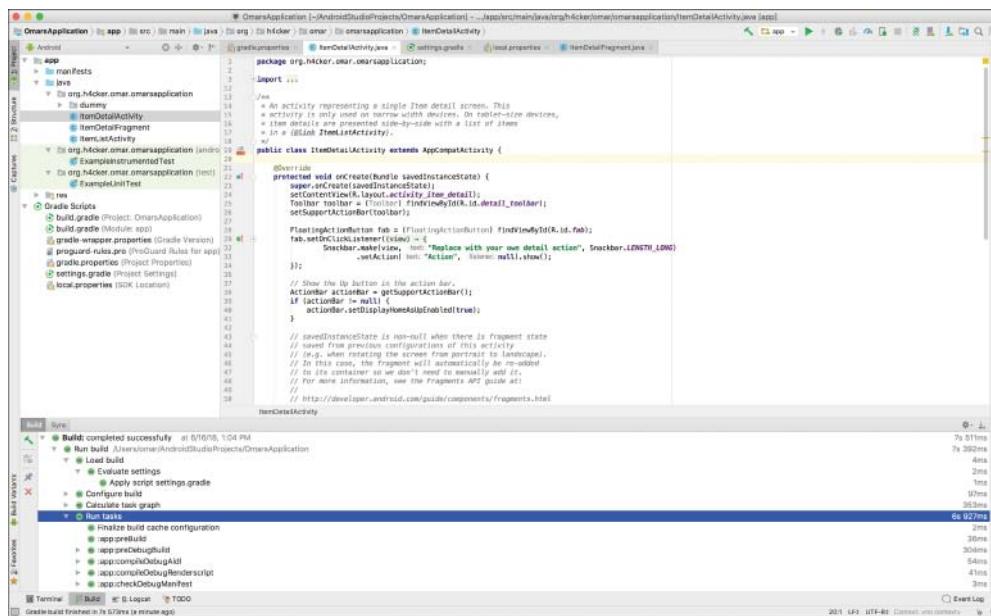


FIGURE 7-13 Android Studio

For dynamic analysis, you need an Android device to run the target app. In principle, however, you can do without a real Android device and test on the emulator. Figure 7-14 shows the Android emulator that comes with Android Studio.

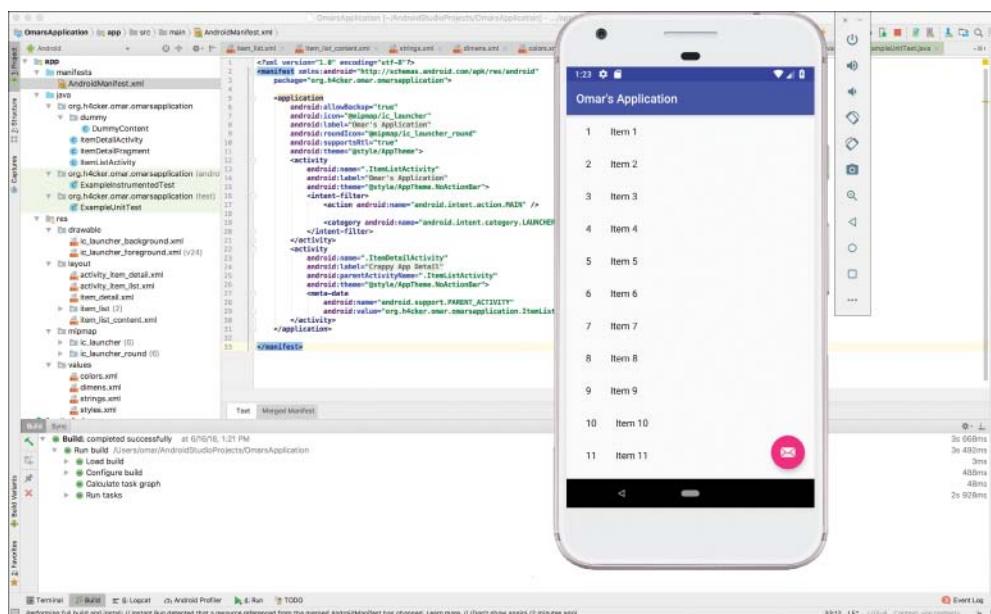


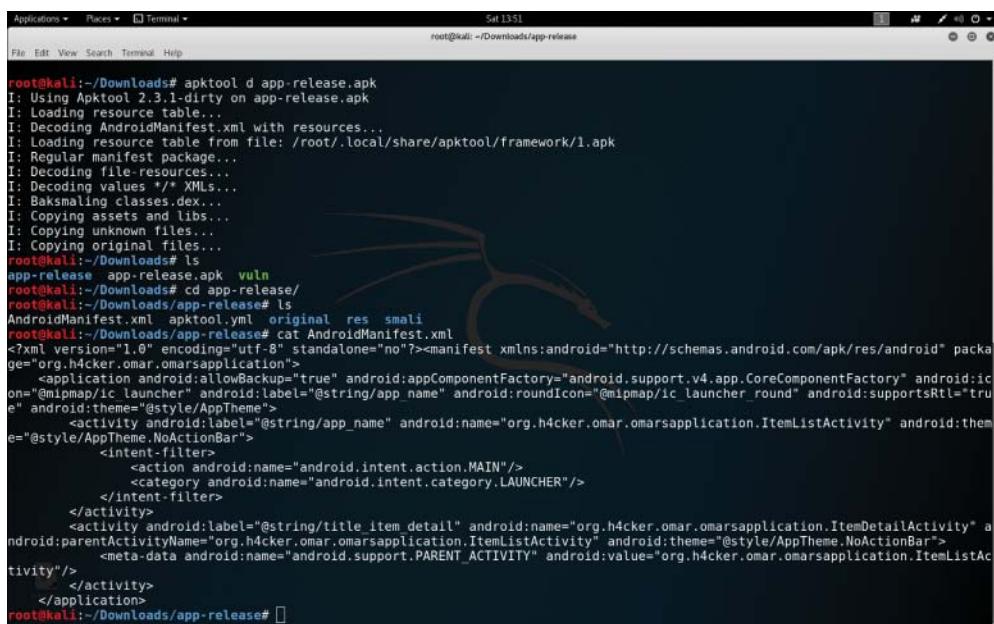
FIGURE 7-14 Android Emulator

Testing on a real device makes for a smoother process and a more realistic environment. However, emulators provide a lot of convenience and flexibility.

Developers and users often root their real devices to get full control over the operating system and to bypass restrictions such as app sandboxing. These privileges in turn allow individuals to use techniques like code injection and function hooking more easily. Rooting is risky and can void the device warranty. You might end up “bricking” a device (rendering it inoperable and unusable) if you run into problems when rooting the device. More importantly, rooting a device creates additional security risks because built-in exploit mitigations are often removed.

TIP You should not root a personal device on which you store your private information. It is recommended to use a cheap, dedicated test device instead.

Figure 7-15 demonstrates how to use Apktool to decode and analyze the Android application OmarsApplication.



The screenshot shows a terminal window titled 'root@kali: /Downloads/app-release'. The terminal is running the command 'apktool d app-release.apk'. The output shows the decoding process of the APK file, including resource tables, AndroidManifest.xml, and various resources like XMLs, dex files, and assets. The terminal then lists the contents of the 'app-release' directory, which includes the decoded APK ('app-release.apk'), a 'vuln' folder, and the original manifest ('AndroidManifest.ym'). The final part of the output shows the original manifest XML being displayed.

```

root@kali:~/Downloads# apktool d app-release.apk
I: Using Apktool 2.3.1-dirty on app-release.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /root/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
root@kali:~/Downloads# ls
app-release app-release.apk vuln
root@kali:~/Downloads# cd app-release/
root@kali:~/Downloads/app-release# ls
AndroidManifest.ym original res smali
root@kali:~/Downloads/app-release# cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" packa
ge="org.h4cker.omar.omarsapplication">
<application android:allowBackup="true" android:appComponentFactory="android.support.v4.app.CoreComponentFactory" android:ic
on="@mipmap/ic_launcher" android:label="@string/app_name" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true"
 android:theme="@style/AppTheme">
    <activity android:label="@string/app_name" android:name="org.h4cker.omar.omarsapplication.ItemListActivity" android:them
e="@style/AppTheme.NoActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <activity android:label="@string/title_item_detail" android:name="org.h4cker.omar.omarsapplication.ItemDetailActivity" a
ndroid:parentActivityName="org.h4cker.omar.omarsapplication.ItemListActivity" android:theme="@style/AppTheme.NoActionBar">
        <meta-data android:name="android.support.PARENT_ACTIVITY" android:value="org.h4cker.omar.omarsapplication.ItemListAc
tivity"/>
    </activity>
</application>
root@kali:~/Downloads/app-release# 
```

FIGURE 7-15 Using Apktool

NOTE The source code for this sample application can be accessed at <https://github.com/The-Art-of-Hacking/art-of-hacking>.

A few tools and frameworks are designed to test Android-based systems and related applications:

- **Androick:** This collaborative research project allows any user to analyze an Android application. You can download Androick from <https://github.com/Flo354/Androick>.
- **NowSecure App Testing:** This is a mobile app security testing suite for Android and iOS mobile devices. There are two versions: a commercial edition and a community (free) edition. You can obtain more information about NowSecure from <https://www.nowsecure.com/solutions/mobile-app-security-testing>.
- **OWASP SeraphimDroid:** This privacy and device protection application for Android devices helps users learn about risks and threats coming from other Android applications. SeraphimDroid is also an application firewall for Android devices that blocks malicious SMS or MMS from being sent, Unstructured Supplementary Service Data (USSD) codes from being executed, or calls from being called without user permission and knowledge. You can obtain more information about SeraphimDroid from https://www.owasp.org/index.php/OWASP_SeraphimDroid_Project.

Understanding Apple iOS Security

Key Topic

The iOS operating system runs only in Apple mobile devices, including the iPhone, iPad, and iPods. Apple tvOS has inherited many architectural components and features from iOS. iOS apps run in a restricted environment and are isolated from each other at the file system level. iOS apps are also significantly limited in terms of system API access compared to macOS and other operating systems. Apple restricts and controls access to the apps that are allowed to run on iOS devices. The Apple App Store is the only official application distribution platform.

iOS apps are isolated from each other via the Apple sandbox and mandatory access controls defining the resources an app is allowed to access. iOS offers very few IPC options compared to Android, which significantly reduces the attack surface. Uniform hardware and tight hardware/software integration create another security advantage.

The iOS security architecture consists of six core features:

- Hardware security
- Secure boot
- Code signing

- Sandbox
- Encryption and data protection
- General exploit mitigations

Every iOS device has two built-in Advanced Encryption Standard (AES) 256-bit keys (GID and UID). These keys are included in the application processor and secure enclave during manufacturing. There's no direct way to read these keys with software or debugging interfaces such as JTAG. The GID is a value shared by all processors in a class of devices that is used to prevent tampering with firmware files. The UID is unique to each device and is used to protect the key hierarchy that's used for device-level file system encryption. UIDs are not created during manufacturing, and not even Apple can restore the file encryption keys for a particular device.

The Apple secure boot chain consists of the kernel, the bootloader, the kernel extensions, and the baseband firmware. Apple has also implemented an elaborate DRM system to make sure that only Apple-approved code runs on Apple devices. FairPlay Code Encryption is applied to apps downloaded from the App Store. FairPlay was developed as a DRM for multimedia content purchased through iTunes.

The App Sandbox is an iOS sandboxing technology. It is enforced at the kernel level and has been a core security feature since the first release of iOS. All third-party apps run under the same user (mobile), and only a few system applications and services run as root. Regular iOS apps are confined to a container that restricts access to the app's own files and a very limited number of system APIs. Access to all resources (such as files, network sockets, IPCs, and shared memory) is controlled by the sandbox. In addition, iOS implements address space layout randomization (ASLR) and the eXecute Never (XN) bit to mitigate code execution attacks.

iOS developers cannot set device permissions directly; they do so by using APIs. The following are a few examples of APIs and resources that require user permission:

- Contacts
- Microphone
- Calendars
- Camera
- Reminders
- HomeKit
- Photos
- HealthKit
- Motion activity and fitness

- Speech recognition
- Location Services
- Bluetooth
- Media library
- Social media accounts

There are a few tools you can use to practice security testing on mobile devices. One of the most popular is the Damn Vulnerable iOS application, a project that provides an iOS application to practice mobile attacks and security defenses. It has a set of challenges that can be completed by an individual. Each challenge area corresponds to an in-depth article designed to teach the fundamentals of mobile security on the iOS platform. The following are examples of the challenges in the Damn Vulnerable iOS application:

- Insecure Data Storage
- Jailbreak Detection
- Runtime Manipulation
- Transport Layer Security
- Client-Side Injection
- Broken Cryptography
- Binary Patching
- Side Channel Data Leakage
- Security Decisions via Untrusted Input

A learning tool for iOS security that is very popular and maintained by OWASP is iGoat. iGoat was inspired by the OWASP WebGoat project and has a similar conceptual flow. iGoat is free software, released under the GPLv3 license. iGoat can be downloaded from https://www.owasp.org/index.php/OWASP_iGoat_Tool_Project.

Another tool is the MobiSec Live Environment Mobile Testing Framework. MobiSec is a live environment for testing mobile environments, including devices, applications, and supporting infrastructure. The purpose is to provide attackers and defenders the ability to test their mobile environments to identify design weaknesses and vulnerabilities. MobiSec can be downloaded from <https://sourceforge.net/projects/mobisec>.

MITRE started a collaborative research project focused on open source iOS security controls called iMAS. iMAS was created to protect iOS applications and data beyond the Apple-provided security model and reduce the attack surface of iOS mobile devices and applications. The source code for iMAS is available on GitHub at <https://github.com/project-imas>.

Understanding Physical Security Attacks

Physical security is a very important element when defending an organization against any security risk. The following sections provide an overview of physical device security and facilities/building security concepts.



Understanding Physical Device Security

Attackers with physical access to a device can perform a large number of attacks. Of course, device theft is one of the most common risks and the main reason it is important to encrypt workstations, laptops, and mobile devices as well as to enable remote wipe and remote recovery features. On the other hand, a few more sophisticated attacks and techniques can be carried out, including the following:

- **Cold boot attacks:** Cold boot is a type of side channel attack in which the attacker tries to retrieve encryption keys from a running operating system after using a cold reboot (system reload). Cold boot attacks attempt to compromise the data remanence property of DRAM and SRAM to retrieve memory contents that could remain readable in the seconds to minutes after power has been removed from the targeted system. Typically, this type of attack by using removable media to boot a different operating system used to dump the contents of pre-boot physical memory to a file.
- **Serial console debugging, reconnaissance, and tampering:** Many organizations use terminal servers (serial console servers) to allow remote access to the serial port of another device over a network. These devices provide remote access to infrastructure devices (for example, routers, switches), servers, and industrial control systems. They are also used to provide out-of-band access to network and power equipment for the purpose of recovery in the case of an outage. Many serial devices do not require authentication and instead assume that if you are physically connected to a serial port, you probably are assumed to be allowed to configure and connect to the system. Clearly, this can be abused by any attacker to gain access to a victim system. Even if terminal servers may allow you to connect using a non-privileged account, attackers can use unprotected serial consoles for reconnaissance and debugging to then perform further attacks on the targeted system.
- **JTAG debugging, reconnaissance, and tampering:** JTAG is a hardware access interface that allows a penetration tester to perform debugging of hardware implementations. Debuggers can use JTAG access registers, memory contents, and interrupts, and they can even pause or redirect software instruction flows. JTAG can be an effective attack research tool because it allows debugging software (such as OpenOCD) control over a JTAG interface. OpenOCD can be used to manipulate the JTAG's TAP controller and to send

bits to a state machine with the goal of the chip being able to interpret them as valid commands. These types of tools allow you to debug firmware and software in devices via the GNU Project Debugger (GDB) or even interact with other tools like IDA Pro and other disassemblers and debuggers.

Clearly, an attacker with physical access to the targeted system has an advantage. Physical security to protect buildings and facilities is therefore crucial. In the next section, you will learn details about different physical security threats and attacks against buildings and facilities.

Protecting Your Facilities Against Physical Security Attacks

Key Topic

Numerous types of attacks can be carried to infiltrate facilities and to steal sensitive information from an organization. The following are some of the most common of them:

- **Piggybacking/tailgating:** An unauthorized individual may follow an authorized individual to enter a restricted building or facility.
- **Fence jumping:** An unauthorized individual may jump a fence or a gate to enter a restricted building or facility.
- **Dumpster diving:** An unauthorized individual may search for and attempt to collect sensitive information from the trash.
- **Lockpicking:** An unauthorized individual may manipulate or tamper with a lock to enter a building or obtain access to anything that is protected by a lock. Lock bypass is a technique used in lockpicking. Locks may be bypassed in many ways, including by using techniques such as simple loiding attempts (using a “credit card” or similar items against self-closing “latch” locks) and bypassing padlocks by shimming.
- **Egress sensors:** Attackers may tamper with egress sensors to open doors.
- **Badge cloning:** Attackers may clone the badges of employees and authorized individuals to enter a restricted facility or a specific area in a building. One of the most common techniques is to clone radio-frequency identification (RFID) tags (refer to Chapter 5).

Exam Preparation Tasks

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, Chapter 11, “Final Preparation,” and the exam simulation questions in the Pearson Test Prep software online.

Review All Key Topics

Review the most important topics in this chapter, noted with the Key Topics icon in the outer margin of the page. Table 7-3 lists these key topics and the page number on which each is found.

Table 7-3 Key Topics for Chapter 7



Key Topic Element	Description	Page Number
Summary	Understanding insecure service and protocol configurations	281
Summary	Understanding local privilege escalation	285
Summary	Understanding Linux permissions	286
Summary	Changing Linux permissions and understanding sticky bits	288
Summary	Understanding SUID or SGID and Unix programs	291
Summary	Identifying insecure Sudo implementations	294
Summary	Understanding ret2libc attacks	298
Summary	Defining CPassword	299
Summary	Abusing and obtaining clear-text LDAP credentials	300
Summary	Understanding Kerberoasting	301
Summary	Compromising credentials in Local Security Authority Subsystem Service (LSASS) implementations	301
Summary	Understanding and attacking the Windows SAM database	302
Summary	Understanding dynamic link library (DLL) hijacking	303
Summary	Abusing exploitable services	304
Summary	Exploiting insecure file and folder permissions	305
Summary	Defining and understanding keyloggers	306
Summary	Defining and understanding scheduled tasks	307
Summary	Understanding sandbox escape attacks	308
Summary	Understanding virtual machine (VM) escape attacks	310
Summary	Identifying container security challenges	313
Summary	Understanding the top mobile security threats and vulnerabilities	314
Summary	Understanding Android security	316
Summary	Understanding Apple iOS security	323
Summary	Understanding cold boot attacks, serial console, and JTAG debugging reconnaissance and tampering	326
Summary	Understanding physical security attacks	327

Define Key Terms

Define the following key terms from this chapter and check your answers in the glossary:

piggybacking, tailgating, fence jumping, dumpster diving, lockpicking, lock bypass, JTAG, sandbox, keylogger, Group Policy Object (GPO), Kerberoast, CPassword, Ret2libc

Q&A

The answers to these questions appear in Appendix A. For more practice with exam format questions, use the Pearson Test Prep software online.

1. Which of the following involves an unauthorized individual searching and attempting to collect sensitive information from the trash?
 - a. Piggybacking
 - b. Fence jumping
 - c. Dumpster diving
 - d. Lockpicking
2. Which of the following is a technique that is executed using disassemblers and decompilers to translate an app's binary code or bytecode back into a more or less understandable format?
 - a. Static and dynamic binary analysis
 - b. Static and dynamic source code analysis
 - c. Binary patching, or "modding"
 - d. Binary code injection
3. Which of the following is a sandbox built in the Linux kernel to only allow the `write()`, `read()`, `exit()`, and `sigreturn()` system calls?
 - a. SUDI
 - b. Seccomp
 - c. SELinux
 - d. Linux-jail

4. Which of the following statements is not true?
 - a. Modern web browsers provide sandboxing capabilities to isolate extensions and plugins.
 - b. HTML5 has a sandbox attribute for use with iframes.
 - c. Java virtual machines include a sandbox to restrict the actions of untrusted code, such as a Java applet.
 - d. Microsoft's .NET Common Language Runtime cannot enforce restrictions on untrusted code.
5. Which of the following can attackers use to capture every keystroke of a user in a system and steal sensitive data (including credentials)?
 - a. RATs
 - b. Keybinders
 - c. Keyloggers
 - d. Ransomware
6. Which of the following functionalities can an attacker abuse to try to elevate privileges if the service is running under SYSTEM privileges?
 - a. Unquoted service paths
 - b. Unquoted PowerShell scripts
 - c. Writable SYSTEM services using the **GetSystemDirectory** function
 - d. Cross-site scripting (XSS)
7. Which of the following is not a place where Windows stores password hashes?
 - a. SAM database
 - b. LSASS
 - c. PowerShell hash store
 - d. AD database
8. Which of the following is an open source tool that allows an attacker to retrieve user credential information from the targeted system and potentially perform pass-the-hash and pass-the-ticket attacks?
 - a. SAM Stealer
 - b. Mimikatz
 - c. Kerberoast
 - d. Hashcrack

This page intentionally left blank

Index

A

AC (Attack Complexity) metrics, 37
acceptance of risk, 48
Access Vector (AV) metrics, 37
account data, elements of, 55
acquirers, 54
Activate Exam button (Pearson Test Prep software), 507
Activate New Product button (Pearson Test Prep software), 506
active reconnaissance
defined, 70–71
Enum4linux, 395–400
enumeration
defined, 71
group, 81–82
host, 78–79
network share, 82
packet crafting, 85–87
service, 85
user, 80–81
web page/web application, 83–84
Nikto scans, 84
Nmap port scans, 391–393
half-open, 71–72
ping, 77–78
SYN, 71–72
TCP connect, 73–74
TCP FIN, 76–77
UDP, 74–75
Recon-ng, 90–102
 hackbar target module, 96
 help menu, 92

key list command, 96–97
launching, 91
main menu and splash page, 91
searches, 95
Shodan API, 96–102
 show modules command, 92–95
Zenmap, 393–395
adapters, wireless, 189
Add-Persistence script, 351
Address Resolution Protocol. *See ARP*
(Address Resolution Protocol)
address space layout randomization
(ASLR), 298, 324
ADIA (Appliance for Digital Investigation and Analysis), 457
Adobe Reader, 309
Advanced Encryption Standard (AES),
 324
advertisements, malvertising, 127–128
AES (Advanced Encryption Standard),
 324
AFL (American Fuzzy Lop), 459
Aircrack-ng suite, 186–189, 191–196
 Aireplay-ng, 188, 191, 194–195
 Airmon-ng, 186–187, 191, 194
 Airodump-ng, 188, 191, 194–195
algorithms
 cryptographic, 243–244
 Luhn, 55
ALTER DATABASE statement,
 228
ALTER TABLE statement, 228
American Fuzzy Lop (AFL), 459

- amplification DDoS (distributed denial-of-service) attacks, 178–179
- Anchore, 313
- Andersson, Bjorn, 441
- Androick, 323
- Android
 - android_filesystem_config.h file, 317
 - AndroidManifest.xml file, 317
 - APK (Android Package Kit), 317
 - AVD (Android Virtual Device)
 - Manager, 320
 - security, 316–323
- APIs (application programming interfaces), 40
 - API-based keyloggers, 307
 - documentation, 40
 - ESAPI (Enterprise Security API), 240
 - RESTful (REST), 269
 - Shodan, 96–102, 378–380
 - unprotected, 267–270
- APK (Android Package Kit), 317
- Apktool, 322
 - apktool.yml file, 318
- appetite for risk, 49–50
- Apple
 - iOS security, 323–325
 - Remote Desktop, 348, 433
- Appliance for Digital Investigation and Analysis (ADIA), 457
- application-based vulnerabilities, 207
- approved scanning vendors (ASVs), 54
- APs (access points), rogue, 185
- Aqua Security, 313
 - arm64-v8a folder, 319
 - armeabi folder, 319
 - armeabi-v7a folder, 319
- armorizing, ASCII, 298
- ARP (Address Resolution Protocol)
 - cache poisoning, 173–175
 - spoofing, 173–175
- Art of Hacking GitHub repository, 225, 346, 390, 395, 410, 454
- ASCII armor region, 298
- ASCII armoring, 298
- ASLR (address space layout randomization), 298, 324
- assets directory, 318
- ASVs (approved scanning vendors), 54
- Attack Complexity (AC) metrics, 37
- attacks, 7–8. *See also* evasion techniques; privilege escalation
- authentication-based vulnerabilities
 - credential brute forcing, 243–245
 - default credential exploits, 249–250
 - Kerberos exploits, 250
 - redirect attacks, 249
 - session hijacking, 245–249
- authorization-based vulnerabilities
 - Insecure Direct Object Reference vulnerabilities, 251–252
 - parameter pollution, 250–251
- clickjacking, 261
- command injection, 241–242
- credential, 420
 - Cain and Abel, 424–425
 - CeWL, 431–432
 - Hashcat, 425–427
 - Hydra, 428–429
 - John the Ripper, 420–425
 - Johnny, 425
 - Medusa, 430–431
 - Mimikatz, 432
 - Ncrack, 430–431
 - Patator, 432–433
 - RainbowCrack, 429–430
- CSRF (cross-site request forgery), 260–261
- current threat landscape, 7–9
- DDoS (distributed denial-of-service), 8–9
- exploitation frameworks
 - BeEF, 449–450
 - Metasploit, 442–449
- file inclusion vulnerabilities

- LFI (local file inclusion), 264
- RFI (remote file inclusion), 264–265
- HTML injection, 241
- insecure code practices
 - code signing, lack of, 270
 - error-handling errors, 266
 - hard-coded credentials, 266
 - hidden elements, 270
 - race conditions, 266–267
 - source code comments, 265–266
 - unprotected APIs, 267–270
- insecure service and protocol configurations, 281–284
- to IoT devices, 8–9
- mobile device security, 314–316
 - Android, 316–323
 - Apple iOS, 323–325
 - OWASP Mobile Security Project, 314
- network-based vulnerabilities
 - DDoS (distributed denial-of-service), 178–179
 - DHCP (Dynamic Host Control Protocol), 183–185
 - DNS cache poisoning, 155–157
 - DoS (denial-of-service), 176–177
 - FTP (File Transfer Protocol), 166–168
 - Kerberos, 169–172
 - LDAP (Lightweight Directory Access Protocol), 169–172
 - man-in-the-browser attacks, 249
 - MITM (man-in-the-middle) attacks, 173–175, 193
 - NAC (network access control)
 - bypass, 179–180
 - name resolution and SMB attacks, 148–155
 - pass-the-hash attacks, 168–169, 302–303
 - route manipulation attacks, 175–176
- SMB (Server Message Block), 151–155, 157–159
- SMTP (Simple Mail Transfer Protocol), 159–166
- VLAN hopping, 181–183
- physical, 326–327
- ransomware
 - Nyeta, 354
 - WannaCry, 8
- ret2libc (“return-to-libc”), 298
- security misconfigurations, 262
 - cookie manipulation attacks, 263–264
- directory traversal vulnerabilities, 262–263
- social engineering, 11–12
 - characteristics of, 125–126
 - elicitation, 135
 - interrogation, 136
 - malvertising, 127–128
 - motivation techniques, 137
 - pharming, 126–127
 - phishing, 126
 - pretexting, 136
 - SET (Social-Engineer Toolkit), 129–134
 - shoulder surfing, 137
 - SMS phishing, 134–135
 - spear phishing, 128–134
 - USB key drop, 138
 - voice phishing, 135
 - whaling, 135
- SQL injection, 228
 - blind, 237
 - Boolean technique, 233, 237
 - categories of, 232–234
 - database fingerprinting, 234–235
 - error-based technique, 233
 - examples of, 228–232
 - mitigations, 240
 - out-of-band technique, 233, 237–238
 - stored procedures, 239–240

- time-delay technique, 233, 239
 - UNION operator technique, 233, 235–236
 - threat actors, 9–10
 - wireless and RF-based
 - credential harvesting, 199–200
 - deauthentication attacks, 186–189
 - evil twin attacks, 185–186
 - fragmentation attacks, 197–198
 - IV (initialization vector) attacks, 190
 - KARMA attacks, 197
 - KRACK (key reinstallation attack), 196–197
 - PNL (preferred network list) attacks, 189
 - RFID (radio-frequency identification) attacks, 200
 - rogue access points, 185
 - signal jamming, 189
 - war driving, 190
 - WEP (Wired Equivalent Privacy) attacks, 190–192
 - WPA (Wi-Fi Protected Access) attacks, 192–196
 - WPS (Wi-Fi Protected Setup), 197
 - XSS (cross-site scripting)
 - vulnerabilities, 252–253
 - DOM-based XSS attacks, 256–257
 - evasion techniques, 257–259
 - reflected XSS attacks, 253–254
 - stored XSS attacks, 255–256
 - zero-day, 8
 - audience for reports, identifying, 476–477
 - AUTH command (SMTP), 161
 - Authd, 284
 - authenticated scans, 105
 - authentication
 - Authentication Cheat Sheet (OWASP), 222, 246
 - vulnerabilities
 - credential brute forcing, 243–245
 - default credential exploits, 249–250
 - Kerberos, 169–172
 - Kerberos exploits, 250
 - redirect attacks, 249
 - session hijacking, 245–249
 - authority, in social engineering, 137
 - authorization-based vulnerabilities
 - Insecure Direct Object Reference vulnerabilities, 251–252
 - parameter pollution, 250–251
 - AV (Access Vector) metrics, 37
 - Availability Impact (A) metrics, 37
 - AVD (Android Virtual Device) Manager, 320
 - avoidance of risk, 49
- B**
- backdoors, 238, 346
 - backtracking, 262–263
 - badge cloning, 327
 - bandwidth limitations, vulnerability scanning and, 111
 - Bane, 313
 - base groups, 34–37
 - bash, 460–461
 - basic service set identifiers (BSSIDs), 188
 - BeEF, 449–450
 - BGP hijacking attacks, 175
 - Bienaimé, Pierre, 441
 - bilateral NDAs (nondisclosure agreements), 43
 - binary patching, 315
 - Black Hills Information Security, 90–91
 - black lists, 46
 - BlackArch Linux, 224, 367–368
 - black-box tests, 12, 47
 - blind shells, creating, 338–344
 - blind SQL injection, 237
 - Blowfish, 166
 - blue teams, 46
 - Bluejacking, 199–200

Bluesnarfing, 199–200
Booleans, SQL injection with, 233, 237
Born, Kenton, 441
botnets, 177
Bourne-Again shell (bash), 460–461
Breach Notification Rule, 52
BSSIDs (basic service set identifiers), 188
budget, planning, 32–33
Burp Suite, 41, 214
Bursztein, Elie, 138
business associates, 53
bWAPP, 225

C

C (Confidentiality Impact) metrics, 37
C2 (command and control) utilities, 344–345
cache file, 318
cache poisoning
 ARP (Address Resolution Protocol), 173–175
 DNS (Domain Name System), 155–157
Cain and Abel, 424–425
CAINE (Computer Aided Investigative Environment), 369, 457
CAPEC (Common Attack Pattern Enumeration and Classification), 114
card security codes, 56
cardholder data environment, 55
cat command, 343
CAV2 codes, 56
cd command, 343, 350
Censys, 389–390
CERT.RSA file, 317
CERT.SF file, 317
CeWL, 431–432
CFTC (Commodity Futures Trading Commission), 52
chaining analysis, 37–38
chmod command, 287–289

Choose a Tool dropdown (Dradis), 484–485
CID codes, 56
CIFS (Common Internet File System), 172
CIS Docker Benchmark, 313
Cisco ETA (Encrypted Traffic Analytics), 440
Cisco Smart Install, 281
Clair, 313
classes.dex directory, 318
classification, report, 499
cleanup process, 356
clearev command, 343
clear-test credentials in LDAP, 300–301
clickjacking, 261
Clickjacking Defense Cheat Sheet (OWASP), 261
clients, HTTP (Hypertext Transfer Protocol), 213
code injection, 315
code practices, insecurity in
 code signing, lack of, 270
 error-handling errors, 266
 hard-coded credentials, 266
 hidden elements, 270
 race conditions, 266–267
 source code comments, 265–266
 unprotected APIs, 267–270
code_cache file, 318
cold boot attacks, 326
Collignon, Nicolas, 441
command and control. *See* C2 (command and control) utilities
command injection vulnerabilities, 241–242
commands. *See also* scripts; tools
 aireplay-ng, 188, 191, 194–195
 airmon-ng, 186–187, 191, 194
 airodump-ng, 191, 194
 cat, 343
 cd, 343, 350

chmod, 287–289
 clearev, 343
 continue, 451
 Copy-Item, 349
 dig, 90, 371–372
 download, 343
 edit, 343
 execute, 343
 find, 292–293
 Get-ChildItem, 349
 Get-Command, 350
 Get-Content, 350
 Get-HotFix, 350
 Get-Location, 350
 Get-NetFirewallRule, 350
 Get-Process, 350
 Get-Service, 350
 getsystem, 449
 getuid, 343
 groups, 294
 hashdump, 343, 448
 host, 371–372
 id, 294
 idletime, 343
 ipconfig, 343
 john --list=formats, 420–421
 john -show hashes, 423
 key list, 96–97
 keys add, 386
 keys add shodan_api, 98
 keys list, 386–387
 lcd, 343
 list audit, 417–419
 locate, 423
 lpwd, 343
 ls, 343, 350
 migrate, 343
 Move-Item, 349
 msfconsole, 90–91, 442
 msfdb init, 443
 New-NetFirewallRule, 350
 nmap
 -sF option, 76–77
 -sn option, 77–78
 -sS option, 71–72
 -sT option, 73–74
 -sU option, 74–75
 nslookup, 90, 156–157, 371–372
 ps, 343
 pwd, 343
 resource, 343
 run, 451
 scapy, 86
 screenshot, 448
 search, 343
 Select-String, 349
 service dradis start, 479
 set LHOST, 155
 set RHOST, 155
 setoolkit, 129
 shell, 343
 show info, 387–388
 show modules, 383–386
 show options, 155
 SMTP (Simple Mail Transfer
 Protocol), 160–163
 sudo, 286–287, 294–298
 sysinfo, 449
 upload, 343
 use, 387–389
 use exploit/windows/smb/ms17_010_
 eternalblue, 155
 useradd, 295
 usermod, 294–295
 visudo, 296
 webcam_list, 344
 webcam_snap, 344
 whois, 372–373
 comments, exploiting, 265–266
 Commodity Futures Trading Commission
 (CFTC), 52
 Common Attack Pattern Enumeration
 and Classification (CAPEC),
 114

- Common Internet File System (CIFS), 172
- Common Vulnerabilities and Exposures (CVE), 114–115
- Common Vulnerability Scoring System (CVSS), 34–37, 494–495
- Common Weakness Enumeration (CWE), 115
- communication escalation path, 31–32
- communications, reporting and, 500–501
- Community Edition (Dradis), 479
- compliance scans, 109–110
- compliance-based assessment, 45, 50
- financial sector regulations, 50–52
 - healthcare sector regulations, 52–53
 - key technical elements, 56–57
 - limitations of, 57–58
- PCI DSS (Payment Card Industry Data Security Standard), 53–56
- Computer Aided Investigative Environment (CAINE), 369, 457
- Confidentiality Impact (C) metrics, 37
- confidentiality of findings, 32
- CONNECT method (HTTP), 217
- container security, 310–314
- continue command, 451
- contracts, 41–42
- cookie manipulation attacks, 263–264
- Copy-Item command, 349
- corporate policies, 43–44
- covering tracks, 356
- CPassword attacks, 299
- cracking passwords. *See password crackers*
- CREATE DATABASE statement, 228
- CREATE INDEX statement, 228
- CREATE TABLE statement, 228
- credentials
- attacks, 420
 - brute forcing, 243–245
 - Cain and Abel, 424–425
 - CeWL, 431–432
- credential harvesting, 199–200
- Hashcat, 425–427
- Hydra, 428–429
- John the Ripper, 420–425
- Medusa, 430–431
- Mimikatz, 432
- Ncrack, 430–431
- Patator, 432–433
- RainbowCrack, 429–430
- RFID (radio-frequency identification) attacks, 200
- hard-coded, 266
- Cri-o, 311
- cross-site scripting. *See XSS (cross-site scripting) vulnerabilities*
- cryptographic algorithms, 243–244
- Crysis, 8
- CSRF (cross-site request forgery), 260–261
- curl, 221
- custom daemons and processes, creating, 346
- CVC2 codes, 56
- CVE (Common Vulnerabilities and Exposures), 114–115
- CVSS (Common Vulnerability Scoring System), 34–37, 494–495
- CVV2 codes, 56
- CWE (Common Weakness Enumeration), 115
- cyber ranges, 227
- Cydia Substrate, 315
- D**
- daemons, creating, 346
- Dagda, 313
- Dalvik, 316
- Damn Vulnerable ARM Router (DVAR), 225
- Damn Vulnerable iOS Application (DVIA), 225
- Damn Vulnerable Web App (DVWA), 225

- DATA command (SMTP), 160
- data isolation, 56
- database fingerprinting, 234–235
- databases, SAM (Security Account Manager), 302–303
- databases folder, 318
- DDoS (distributed denial-of-service)
 - attacks, 8–9
 - amplification, 178–179
 - reflected, 178
- deauthentication attacks, 186–189
- debugging tools
 - edb debugger, 452–454
 - GDB (GNU Project Debugger), 450–452
 - Immunity, 454
 - OllyDbg, 452–453
 - Windows Debugger, 452
- default credential exploits, 249–250
- defensive controls, 49
- DEFT (Digital Evidence & Forensics Toolkit), 457
- DELETE method (HTTP), 217
- DELETE statement (SQL), 228
- Dembour, Olivier, 441
- denial-of-service (DoS) attacks, 176–177
- DeNiSe, 441
- Department of Health and Human Services (HHS), 52
- DES, 166
- Dev-Sec.io, 313
- DHCP (Dynamic Host Control Protocol)
 - spoofing, 183–185
 - starvation attacks, 183–185
- dig command, 90
- Dig tool, 371–372
- Digital Evidence & Forensics Toolkit (DEFT), 457
- Dirbuster, 419
- directory climbing, 262–263
- Directory Information Tree (DIT), 170
- directory traversal vulnerabilities, 262–263
- disassemblers
 - IDA, 454–455
 - Objdump, 455–457
- disclaimers, documentation of, 38–39
- discovery scans, 106
- Distinguished Name (DN), 170
- distributed denial-of-service. *See* DDoS
 - (distributed denial-of-service) attacks
- distribution of reports, 499–500
- DIT (Directory Information Tree), 170
- DLL (dynamic link library) hijacking, 303–304
- DN (Distinguished Name), 170
- DNS (Domain Name System)
 - cache poisoning, 155–157
 - DNSSEC (Domain Name System Security Extension), 157
 - exfiltration, 440–442
 - tunneling, 440–442
- dns2tcp, 441
- DNSScapy, 441
- DNSCat, 441
- DNSCat2, 345, 441
- DNSdumpster, 88
- DNSRecon, 67–69
- DNSSEC (Domain Name System Security Extension), 157
- Docker, 310–313
- docker-bench-security, 313
- docker-explorer, 314
- Document Object Model. *See* DOM (Document Object Model)
- documentation, 41
 - API (application programming interface), 268
- legal
 - contracts, 41–42
 - MSAs (master service agreements), 42

- NDAs (nondisclosure agreements), 43
SOWs (statements of work), 42
pre-engagement
 budget, 32–33
 communication escalation path, 31–32
 confidentiality of findings, 32
 disclaimers, 38–39
 impact analysis and remediation
 timelines, 34–38
 point-in-time assessment, 33
 rules of engagement, 30
 target audience, 29–30
 technical constraints, 39
rules of engagement, 30
support resources, 40–41
DOM (Document Object Model)
 DOM-based XSS (cross-site scripting)
 attacks, 256–257
 stored DOM-based attacks, 263–264
domain enumeration, 88–91
Domain Name System. *See* DNS
 (Domain Name System)
DoS (denial-of-service) attacks, 176–177
dot-dot-slash, 262–263
 downgrade attacks, 175
download command, 343
Dradis Framework
 Choose a Tool dropdown, 484–485
 data import, 483
 imported scans, viewing, 488–490
 launching, 479
 login screen, 480
 node organization, 486–488
 node/subnode creation, 481–483
 output file processing, 485–486
 password creation, 479–480
 plugin.output node, 486
 Project Summary screen, 481
Upload Manager screen, 484
Upload Output from Tool option, 483–484
versions, 478–479
DROP INDEX statement, 228
DROP TABLE statement, 228
DropboxC2 (DBC2), 345
Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG), 439
dumpster diving, 327
DVAR (Damn Vulnerable ARM Router), 225
DVIA (Damn Vulnerable iOS Application), 225
DVWA (Damn Vulnerable Web App), 225
dynamic binary analysis, 316
Dynamic Host Control Protocol. *See* DHCP (Dynamic Host Control Protocol)
dynamic link library (DLL) hijacking, 303–304
DynDNS service, 8–9
- ## E
- E (Exploit Code Maturity), 36
eavesdropping, packet, 90
edb debugger, 452–454
edit command, 343
egress sensors, 327
EHLO command (SMTP), 160
Ekman, Erik, 441
electronic protected health information (ePHI), 53
elicitation, 135
email threats
 pharming, 126–127
 phishing, 126
 SMS phishing, 134–135
 spear phishing, 128–134
 voice phishing, 135
 whaling, 135

- Empire, 171, 353–354, 434
- Encrypted Traffic Analytics (ETA), 440
- Encryption, 439–440
- encryption
 - AES (Advanced Encryption Standard), 324
 - cryptographic algorithms, 243–244
 - engagement, rules of, 30
 - Enterprise Security API (ESAPI), 240
 - Enum4linux, 155, 395–400
 - enumeration
 - defined, 71
 - domain, 88–91
 - group, 81–82
 - host, 78–79
 - network share, 82
 - packet crafting, 85–87
 - service, 85
 - user, 80–81
 - web page/web application, 83–84
 - environmental groups, 34–37
 - ePHI (electronic protected health information), 53
 - error handling
 - lack of, 266
 - verbose, 266
 - error-based SQL injection, 233
 - ESAPI (Enterprise Security API), 240
 - escalation of privileges. *See privilege escalation*
 - escalation path, 31–32
 - ESSIDs (extended basic service set identifiers), 188
 - ETA (Encrypted Traffic Analytics), 440
 - EternalBlue exploit, 8, 154–155
 - ethical hacking
 - defined, 6–7
 - importance of, 7
 - nonethical hacking compared to, 6–7
 - Ettercap, 199
 - Evan’s debugger, 452–454
 - evasion techniques
 - DNS tunneling, 440–442
 - Encryption, 439–440
 - Proxychains, 439
 - Tor, 438–439
 - Veil, 434–437
 - XSS (cross-site scripting)
 - vulnerabilities, 257–258
 - evil twin attacks, 185–186
 - exam preparation
 - chapter-ending review tools, 509
 - Pearson Test Prep software, 505
 - exam customization, 507–508
 - exam updates, 508
 - offline access, 506–507
 - online access, 505
 - Premium Edition, 508–509
 - study plans, 509
 - execute command, 343
 - eXecute Never (XN), 324
 - executive summary section (reports), 493
 - exfiltration, DNS (Domain Name System), 440–442
 - ExifTool, 374–375
 - Exploit Code Maturity (E), 36
 - The Exploit Database, 151
 - Exploitability metrics, 35
 - exploitable services, 304–305
 - exploitation frameworks
 - BeEF, 449–450
 - Metasploit, 90–91, 115, 154–155, 442–449
 - Metasploit Unleashed course, 344
 - Meterpreter, 299, 343–344, 446–449
 - RDP connections, creating, 348–349
 - exploits. *See attacks*
 - EXPN command (SMTP), 161
 - export restrictions, 43
 - extended basic service set identifier (ESSID), 188

F

false negatives, 475–476, 477
false positives, 475–476, 477, 495–496
FBI (Federal Bureau of Investigation), 439
FDIC (Federal Deposit Insurance Corporation) Safeguards Act, 51
fear, social engineering and, 137
Federal Bureau of Investigation (FBI), 439
Federal Deposit Insurance Corporation (FDIC) Safeguards Act, 51
Federal Financial Institutions Examination Council (FFIEC), 51
Federal Trade Commission (FTC), 52
Feederbot and Moto, 442
fence jumping, 327
FFIEC (Federal Financial Institutions Examination Council), 51
file inclusion vulnerabilities
 LFI (local file inclusion), 264
 RFI (remote file inclusion), 264–265
File Transfer Protocol. *See* FTP (File Transfer Protocol)
files
 permissions, insecurity in, 305
 SOAP (Simple Object Access Protocol), 40
files folder, 319
FILS (Financial Institutions Letters), 51
financial sector regulations, 50–52
find command, 292–293
Find-AVSignture script, 351
Findbugs, 458
findings, reporting, 495–497
Findsecbugs, 458
Finger, 284
fingerprinting databases, 234–235
Fingerprinting Organizations with Collected Archives (FOCA), 374
FIRST (Forum of Incident Response and Security Teams), 34, 37
FIRST.org, 494

FOCA (Fingerprinting Organizations with Collected Archives), 374

folder permissions, 305
forensics tools, 457–458
forgery, CSRF (cross-site request forgery), 260–261
Forum of Incident Response and Security Teams (FIRST), 34, 37
fragile systems, vulnerability scans for, 111–112
fragmentation attacks, 197–198
Frida, 315
FTC (Federal Trade Commission), 52
FTP (File Transfer Protocol)
 exploits, 166–168
 FTPS (File Transfer Protocol Secure), 166
full scans, 106–108
functions. *See also* commands
 GetSystemDirectory, 304
 GetWindowsDirectory, 304
fuzz testing, 458
fuzzers, 458
 AFL (American Fuzzy Lop), 459
 Mutiny Fuzzing Framework, 459
 Peach, 459

G

Game of Hacks, 225
GDB (GNU Project Debugger), 450–452
GET method (HTTP), 215, 217
Get-ChildItem command, 349
Get-Command command, 350
Get-Content command, 350
Get-GPPAutologon script, 352
Get-GPPPPassword script, 352
Get-HotFix command, 350
Get-HttpStatus script, 352
Get-Keystrokes script, 352
Get-Location command, 350
Get-MicrophoneAudio script, 352
Get-NetFirewallRule command, 350

- Get-Process command, 350
 - Get-SecurityPackages script, 351
 - Get-Service command, 350
 - getsystem command, 449
 - GetSystemDirectory function, 304
 - Get-TimedScreenshot script, 352
 - getuid command, 343
 - Get-VaultCredential script, 352
 - Get-VolumeShadowCopy script, 352
 - GetWindowsDirectory function, 304
 - GLBA (Gramm-Leach-Bliley Act), 51–52
 - GNU Project Debugger (GDB),
 - 450–452
 - goals-based assessment, 45
 - golden ticket attacks (Kerberos), 170–172
 - GPOs (Group Policy Objects), 305–306
 - GPP attacks. *See* CPassword attacks
 - Gramm-Leach-Bliley Act (GLBA), 51–52
 - GraphQL, 40, 267
 - gray-box tests, 13, 47
 - groups
 - CVSS (Common Vulnerability Scoring System), 34–37
 - enumeration, 81–82
 - Group Policy Objects (GPOs), 305–306
 - groups command, 294
- H**
- Hack This, 225
 - Hack This Site, 225
 - Hackazon, 225
 - hackertarget module, 96
 - hacktivists, 10
 - HAL (Hardware Abstraction Layer),
 - 316
 - half-open scanning, 71–72
 - handling reports, 499–500
 - hard-coded credentials, 266
 - Hardware Abstraction Layer (HAL), 316
 - Hashcat, 425–427
 - hashdump command, 343, 448
- HEAD method (HTTP), 217
 - Health Information Technology for Economic and Clinical Health Act, 52
 - health plans, 53
 - healthcare clearinghouses, 53
 - healthcare providers, 53
 - healthcare sector regulations, 52–53
 - HellBound Hackers, 225
 - HELO command (SMTP), 160
 - HELP command (SMTP), 161
 - help menu (Recon-*ng*), 92
 - Heyoka, 441
 - hidden elements, 270
 - hijacking
 - DLL (dynamic link library), 303–304
 - session, 245–249
 - HIPAA Security Enforcement Final Rule, 52
 - HIPAA Security Rule, 52–53
 - HITECH (Health Information Technology for Economic and Clinical Health) Act, 52
 - hopping, VLAN, 181–183
 - Host, 371–372
 - host enumeration, 78–79
 - hosts, local. *See* local host vulnerabilities
 - HPP (HTTP parameter pollution),
 - 250–251
 - HTML (Hypertext Markup Language)
 - HTML5, 309
 - injection, 241
 - HTTP (Hypertext Transfer Protocol),
 - 213–221
 - clients, 213
 - HPP (HTTP parameter pollution),
 - 250–251
 - proxies
 - defined, 214
 - ZAP, 214
 - request/response model, 215–218

- servers, 213
sessions, 213
URLs (uniform resource locators), 219–220
http-enum script, 83–84
Hydra, 428–429
Hypertext Markup Language.
See HTML (Hypertext Markup Language)
Hypertext Transfer Protocol. *See* HTTP (Hypertext Transfer Protocol)
hypervisor-based keyloggers, 307
- I**
- I (Integrity Impact) metrics, 37
id command, 294
IDA, 454–455
Identd, 284
idletime command, 343
IDs
 SGID (set-group-ID)
 Linux, 289
 Unix, 291–293
 SUID (set-user-ID)
 Linux, 289
 Unix, 291–293
IETF (Internet Engineering Task Force), 157
iGoat, 325
IIHI (individually identifiable health information), 53
IMAP (Internet Message Address Protocol), 159
iMAS, 325
Immunity, 454
impact analysis, 34–38
Impact metrics, 36–44
impersonation, 136
individually identifiable health information (IIHI), 53
information gathering.
See reconnaissance
- Information Systems Security Assessment Framework (ISSAF), 16
initialization vector (IV) attacks, 190
injection-based vulnerabilities
 command injection, 241–242
 HTML injection, 241
 SQL injection, 228
 blind, 237
 Boolean technique, 233, 237
 categories of, 232–234
 database fingerprinting, 234–235
 error-based technique, 233
 examples of, 228–232
 mitigations, 240
 out-of-band technique, 233, 237–238
 stored procedures, 239–240
 time-delay technique, 233, 239
 UNION operator technique, 233, 235–236
insecure code practices
 code signing, lack of, 270
 error-handling errors, 266
 hard-coded credentials, 266
 hidden elements, 270
 race conditions, 266–267
 source code comments, 265–266
 unprotected APIs, 267–270
Insecure Direct Object Reference vulnerabilities, 251–252
INSERT INTO statement, 228
insider threats, 10
inspection, packet, 90
Install-SSP script, 351
Integrity Impact (I) metrics, 37
intentionally vulnerable systems, 224–227
Internet Engineering Task Force. *See* IETF (Internet Engineering Task Force)
Internet Message Address Protocol (IMAP), 159
interrogation, 136
intrusion prevention systems (IPSSs), 46

Invoke-CredentialInjection script, 351
 Invoke-DllInjection script, 351
 Invoke-Mimikatz script, 352
 Invoke-NinjaCopy script, 352
 Invoke-Portscan script, 352
 Invoke-ReflectivePEInjection script, 351
 Invoke-ReverseDnsLookup script, 352
 Invoke-Shellcode script, 351
 Invoke-TokenManipulation script, 351
 Invoke-WmiCommand script, 351
 Iodine, 441
 iOS security, 323–325
 IoT (Internet of Things), threats to, 8–9
 ipconfig command, 343
 IPSs (intrusion prevention systems), 46
 ISSAF (Information Systems Security Assessment Framework), 16
 IV (initialization vector) attacks, 190

J

jail, 309
 Japan Computer Emergency Response Team (JPCERT), 113
 JavaScript-based keyloggers, 307
 john --list=formats command, 420–421
 john -show hashes command, 423
 John the Ripper, 420–425
 Johnny, 425
 JPCERT (Japan Computer Emergency Response Team), 113
 JTAG debugging, 326–327
 JWT (JSON Web Token), 223

K

Kali Linux, 224, 366
 Kaminsky, Dan, 441
 KARMA attacks, 197
 Kennedy, Dave, 11–12
 Kerberoast, 301
 Kerberos
 exploits, 250
 Kerberos Delegation, 172
 KRBTGT(Kerberos TGT) password hash, 170
 vulnerabilities, 169–172
 kernel-based keyloggers, 307
 key list command, 96–97
 key management, 57
 key reinstallation attack (KRACK), 196–197
 keyloggers, 306–307
 keys add command, 386
 keys add shodan_api command, 98
 keys list command, 386–387
 KRACK (key reinstallation attack), 196–197
 KRBTGT (Kerberos TGT) password hash, 170

L

labs, 16–17
 recovery, 19
 requirements and guidelines, 18
 tools, 18–19
 web application, 224–227
 languages
 Python, 461
 Ruby, 461–462
 Lanman, 80–81
 lateral movement, 347
 post-exploitation scanning, 347–348
 remote access protocols, 348–349
 lcd command, 343
 LDAP (Lightweight Directory Access Protocol)
 clear-test credentials in, 300–301
 vulnerabilities, 169–172
 legal concepts, 41
 contracts, 41–42
 MSAs (master service agreements), 42
 NDAs (nondisclosure agreements), 43
 SOWs (statements of work), 42

- legislation. *See* regulations
- LFI (local file inclusion) vulnerabilities, 264
- lib directory, 318
- lib folder, 319
- Lightweight Directory Access Protocol.
See LDAP (Lightweight Directory Access Protocol)
- likeness, social engineering and, 137
- Link-Local Multicast Name Resolution.
See LLMNR (Link-Local Multicast Name Resolution)
- Linux
- distributions, 224, 365
 - BlackArch Linux, 224, 367–368
 - CAINE (Computer Aided Investigative Environment), 369
 - Kali Linux, 224, 366
 - Parrot, 224, 367
 - Security Onion, 369–370
 - SELinux (Security Enhanced Linux), 293
 - LXC (Linux Containers), 310
 - permissions, 286–291
- list audit command, 417–419
- LLMNR (Link-Local Multicast Name Resolution), 148–150
- local file inclusion (LFI) vulnerabilities, 264
- local host vulnerabilities, 281. *See also* privilege escalation
- insecure service and protocol configurations, 281–284
 - mobile device security, 314–316
 - Android, 316–323
 - Apple iOS, 323–325
 - physical security attacks, 326–327
- local privilege escalation. *See* privilege escalation
- Local Security Authority Subsystem Service (LSASS) credentials, 301–302
- locate command, 423
- lockpicking, 327
- Locky, 8
- lpwd command, 343
- ls command, 343, 350
- LSASS (Local Security Authority Subsystem Service) credentials, 301–302
- Luhn, Hans Peter, 55
- Luhn algorithm, 55
- LXC (Linux Containers), 310
- ## M
- MAC authentication (auth) bypass, 179–180
- magnetic credit card stripes, 56
- MAIL command (SMTP), 161
- Maltego, 381–382
- malvertising, 127–128
- Management Frame Protection (MFP), 189
- Management Information Base (MIB), 158
- MANIFEST.MF file, 317
- man-in-the-browser attacks, 249
- man-in-the-middle attacks. *See* MITM (man-in-the-middle) attacks
- Masscan, 78–79
- master service agreements (MSAs), 42
- MASVS (Mobile AppSec Verification Standard) Anti-Reversing Controls, 315
- MD5 algorithm, 166
- measurements, 494–495
- Medusa, 430–431
- memory-injection-based keyloggers, 307
- merchants, 54
- messages (SMB)
 - SMB_COM_NEGOTIATE, 80
 - SMB_COM_SESSION_SETUP_ANDX, 80–81
- META-INF file, 317

- Metasploit, 90–91, 115, 154–155, 442–449
 - Metasploit Unleashed course, 344
 - Meterpreter, 299, 343–344, 446–449
 - RDP connections, creating, 348–349
 - Metasploitable2, 225
 - Metasploitable3, 225
 - Meterpreter, 299, 343–344, 446–449
 - methodologies, penetration testing, 13–16
 - methodology section (reports), 494
 - metrics, 34–37, 494–495
 - MFA (multifactor authentication), 243
 - MFP (Management Frame Protection), 189
 - MIB (Management Information Base), 158
 - Microsoft
 - MOM (Microsoft Operations Manager), 354
 - MS17-010 security bulletin, 8
 - MSRPC (Microsoft Remote Procedure Call), 82
 - Office, 309
 - migrate command, 343
 - Mimikatz, 169–172, 302, 432
 - mips file, 319
 - mitigation
 - risk, 48–49
 - SQL injection, 240
 - XSS (cross-site scripting) vulnerabilities, 258–259
 - MITM (man-in-the-middle) attacks, 249
 - ARP cache poisoning, 173–175
 - ARP spoofing, 173–175
 - downgrade attacks, 175
 - KARMA, 197
 - session hijacking, 193
 - MITRE iMAS, 325
 - Mobile AppSec Verification Standard (MASVS) Anti-Reversing Controls, 315
 - mobile device security, 314–316
 - Android, 316–323
 - Apple iOS, 323–325
 - OWASP Mobile Security Project, 314
 - MobiSec Live Environment Mobile Testing Framework, 325
 - modding, 315
 - Modified Base Metrics, 36
 - modules
 - PowerSploit, 351–352
 - Recon-*ng*
 - hackertarget*, 96
 - show modules* command, 92–95
 - MOM (Microsoft Operations Manager), 354
 - Moore, H. D., 442
 - motivation techniques, social engineering, 137
 - Mount-VolumeShadowCopy script, 352
 - Move-Item command, 349
 - MS17-010 security bulletin, 8
 - MSAs (master service agreements), 42
 - msfconsole command, 90–91, 442
 - msfdb init command, 443
 - MSRPC (Microsoft Remote Procedure Call), 82
 - multifactor authentication (MFA), 243
 - multilateral NDAs (nondisclosure agreements), 43
 - Mutiny Fuzzing Framework, 459
- N**
- NAC (network access control), 46, 179–180
 - name resolution, 148
 - LLMNR (Link-Local Multicast Name Resolution), 148–150
 - NetBIOS, 148–150
 - National Cybersecurity and Communications Integration Center (NCCIC), 113
 - National Institute of Standards and Technology (NIST), 15, 57, 113

- National Security Agency (NSA), 439
NCCIC (National Cybersecurity and Communications Integration Center), 113
Ncrack, 430–431
NDAs (nondisclosure agreements), 43
need-to-know, 499
Nessus scanner, 106–108, 403–404
.NET Common Language Runtime, 309
NetBIOS, 148–150
Netcat, 338–342
Netdump, 284
Netdump-server, 284
netstat command, 105
network access control. *See NAC*
(network access control)
network diagrams, 41
network infrastructure tests, 11
network share enumeration, 82
Network Time Protocol (NTP), 178
network-based vulnerabilities, 148
DDoS (distributed denial-of-service)
amplification, 178–179
reflected, 178
DHCP (Dynamic Host Control Protocol)
spoofing, 183–185
starvation attacks, 183–185
DNS cache poisoning, 155–157
DoS (denial-of-service), 176–177
FTP (File Transfer Protocol), 166–168
Kerberos, 169–172
LDAP (Lightweight Directory Access Protocol), 169–172
man-in-the-browser attacks, 249
MITM (man-in-the-middle) attacks, 249
ARP cache poisoning, 173–175
ARP spoofing, 173–175
 downgrade attacks, 175
KARMA, 197
session hijacking, 193
NAC (network access control) bypass, 179–180
name resolution and SMB attacks, 148
LLMNR (Link-Local Multicast Name Resolution), 148–150
NetBIOS, 148–150
SMB (Server Message Block), 151–155
network topology, 110–111
pass-the-hash attacks, 168–169, 302–303
route manipulation attacks, 175–176
SMTP (Simple Mail Transfer Protocol) commands, 160–163
known SMTP server exploits, 163–166
open relay, 160
TCP port numbers, 159
SNMP (Simple Network Management Protocol), 157–159
VLAN hopping, 181–183
wireless and RF-based
 credential harvesting, 199–200
deauthentication attacks, 186–189
evil twin attacks, 185–186
fragmentation attacks, 197–198
IV (initialization vector) attacks, 190
KARMA attacks, 197
KRACK (key reinstallation attack), 196–197
PNL (preferred network list) attacks, 189
RFID (radio-frequency identification) attacks, 200
rogue access points, 185
signal jamming, 189
war driving, 190
WEP (Wired Equivalent Privacy) attacks, 190–192
WPA (Wi-Fi Protected Access) attacks, 192–196
WPS (Wi-Fi Protected Setup), 197

- New York Department of Financial Services Cybersecurity Regulation, 51, 52
- New-ElevatedPersistenceOption script, 351
- New-NetFirewallRule command, 350
- New-UserPersistenceOption script, 351
- New-VolumeShadowCopy script, 352
- Nfs, 284
- Nikto, 84, 410–413, 488–489
- NIST (National Institute of Standards and Technology), 15, 57, 113
- Nmap, 155, 391–393
- enumeration
 - group, 81–82
 - host, 78–79
 - network share, 82
 - service, 85
 - user, 80–81
 - web page/web application, 83–84
 - scans
 - ping, 77–78
 - TCP connect, 73–74
 - TCP FIN, 76–77
 - UDP, 74–75
 - scripts
 - http-enum, 83–84
 - smb-enum-groups, 81–82
 - smb-enum-processes, 85
 - smb-enum-shares, 82
 - smb-enum-users.nse, 80–81
 - smtp-open-relay, 160
 - SNMP-related, 158–159
 - Zenmap, 393–395
- nmap command
- sF option, 76–77
 - sS option, 71–72
 - sT option, 73–74
 - sU option, 74–75
- Nmap Scripting Engine (NSE), 69
- nodes (Dradis)
- creating, 481–483
- organizing, 486–488
- plugin.output, 486
- no-execute (NX) bit feature, 298
- nondisclosure agreements (NDAs), 43
- nonethical hacking, 6–7
- nontraditional assets, vulnerability scans for, 111–112
- Notary, 314
- NotPetya, 8
- NowSecure App Testing:323
- NSA (National Security Agency), 439
- NSE (Nmap Scripting Engine), 69
- http-enum script, 83–84
 - smb-enum-groups script, 81–82
 - smb-enum-processes script, 85
 - smb-enum-shares script, 82
 - smb-enum-users.nse script, 80–81
 - smtp-open-relay script, 160
 - SNMP-related scripts, 158–159
- Nslookup, 90, 156–157, 371–372
- NTLM (NT LAN Manager), 80–81, 168, 302–303
- NTP (Network Time Protocol), 178
- NX (no-execute) bit feature, 298
- Nyeta ransomware, 354

O

- OASP Mobile Security Testing Guidelines, 16
- OBEX (Object Exchange), 199
- Objdump, 455–457
- Object Exchange (OBEX), 199
- offensive controls, 49
- Offensive Security Example penetration test report, 497
- offline brute-force attacks, 243
- OllyDbg, 452–453
- one-click attacks, 260–261
- online brute-force attacks, 243
- open relay (SMTP), 160

- Open Source Intelligence. *See* OSINT (Open Source Intelligence) gathering
- Open Source Security Testing Methodology Manual (OSSTMM), 15–16
- Open Web Application Security Project. *See* OWASP (Open Web Application Security Project)
- OpenAPI, 40, 268
- OpenSCAP, 314
- OpenSSL, POODLE (Padding Oracle on Downgraded Legacy Encryption) vulnerability, 175
- OpenVAS, 401–403
- OpenVz, 311
- operators, UNION, 233, 235–236
- OPTIONS method (HTTP), 217
- organized crime, 9–10
- original folder, 318
- oscap-docker, 314
- OSINT (Open Source Intelligence) gathering, 90 defined, 90 tools, 370 Censys, 389–390 Dig, 371–372 ExifTool, 374–375 FOCA (Fingerprinting Organizations with Collected Archives), 374 Host, 371–372 Maltego, 381–382 Nslookup, 371–372 Recon-*ng*, 382–389 Shodan API, 378–380 Theharvester, 376–378 Whois, 372–373
- OSSTMM (Open Source Security Testing Methodology Manual), 15–16
- Out-CompressedDll script, 351
- Out-EncodedCommand script, 351
- Out-EncryptedScript script, 351
- Out-Minidump script, 352
- out-of-band SQL injection, 233, 237–238
- output file processing (Dradis), 485–486
- OverTheWire Wargames, 225
- OWASP (Open Web Application Security Project), 11, 226 Authentication Cheat Sheet, 222, 246 Clickjacking Defense Cheat Sheet, 261 Enterprise Security API (ESAPI), 240 iGoat, 325 Mobile Security Project, 314 Mutillidae II, 225 REST Security Cheat Sheet, 269 Risk Rating Methodology, 495 SeraphimDroid, 323 ZAP (Zed Attack Proxy), 41, 251, 413–414
- OzymanDNS and sods, 441
- P**
- Packetforge-*ng*, 197–198
- packets capture, 215 crafting, 85–87 inspection and eavesdropping, 90
- Padding Oracle on Downgraded Legacy Encryption (POODLE) vulnerability, 175
- PALADIN, 457
- PANs (primary account numbers), 54
- parameter pollution, 250–251
- Parrot, 224, 367
- passive reconnaissance, 87–88, 370 Censys, 389–390 defined, 70–71 Dig, 371–372 domain enumeration, 88–91 ExifTool, 374–375 FOCA (Fingerprinting Organizations with Collected Archives), 374

- Host, 371–372
- Maltego, 381–382
- Nslookup, 90, 156–157, 371–372
- OSINT (Open Source Intelligence)
 - gathering, 90
 - packet inspection and eavesdropping, 90
- Recon-*ng*, 90–102, 382–389
 - hackertarget module, 96
 - help menu, 92
 - key list command, 96–97
 - keys add command, 386
 - keys list command, 96–97, 386–387
 - launching, 91
 - main menu and splash page, 91
 - searches, 95
 - Shodan API, 96–102
 - show info command, 387–388
 - show modules command, 92–95, 383–386
 - support resources, 389
 - use command, 387–389
- Shodan API, 96–102, 378–380
- Theharvester, 376–378
- vulnerability scans, 103
 - authenticated, 105
 - challenges of, 109–112
 - compliance, 109–110
 - discovery, 106
 - full, 106–108
 - how it works, 103–104
 - results analysis, 112–113
 - stealth, 108–109
 - support resources, 113–115
 - unauthenticated, 104–105
 - vulnerability management, 115–116
- Whois, 372–373
- passive vulnerability scanners, 108. *See also* scans
- pass-the-hash attacks, 168–169, 302–303
- password crackers
 - Cain and Abel, 424–425
 - CeWL, 431–432
 - Hashcat, 425–427
 - Hydra, 428–429
 - John the Ripper, 420–425
 - Johnny, 425
 - Medusa, 430–431
 - Mimikatz, 432
 - Ncrack, 430–431
 - Patator, 432–433
 - RainbowCrack, 429–430
- passwords
 - Dradis Framework, 479–480
 - management, 56
 - Patator, 432–433
 - path traversals, 262–263
 - payment brands, 54
 - Payment Card Industry Data Security Standard. *See* PCI DSS (Payment Card Industry Data Security Standard)
 - PCI DSS (Payment Card Industry Data Security Standard), 13–14, 53–56, 491–493
 - PCI forensic investigators (PFIs), 54
 - PCI SSC (Payment Card Industry Security Standards Council), 53
- Peach, 459
- Pearson Test Prep software, 505
 - exam customization, 507–508
 - exam updates, 508
 - offline access, 506–507
 - online access, 505
 - Premium Edition, 508–509
- PearsonITCertification.com, 506
- penetration testing, defined, 6–7
- Penetration Testing Execution Standard (PTES), 13, 16
- Penetration Testing Framework, 14
- penetration testing labs. *See* labs
- penetration testing methodologies, 10, 13–16

- reasons for following, 10
- web application tests, 11
- penetration testing planning. *See* planning and preparation
- penetration testing reports. *See* reports
- penetration testing tools. *See* tools
- permission escalation. *See* privilege escalation
- Permissions Calculator website, 290
- persistence, 337, 433
 - blind shells, creating, 338–344
 - C2 (command and control) utilities, 344–345
 - custom daemons and processes, creating, 346
 - reverse shells, creating, 338–344
 - scheduled tasks, creating, 346
 - tools, 433–434
 - users, creating, 346
- Peruggia, 225
- PFIIs (PCI forensic investigators), 54
- pharming, 126–127
- phishing, 126
 - SMS, 134–135
 - spear, 128–134
 - voice, 135
 - whaling, 135
- physical facility tests, 11
- physical security attacks, 326–327
- Piessens, Frank, 196
- Pietraszek, Tadeusz, 441
- piggybacking, 327
- ping scans, 77–78
- ping sweeps, 77
- pivoting, 347
 - post-exploitation scanning, 347–348
 - remote access protocols, 348–349
- planning and preparation
 - compliance-based assessment, 50
 - financial sector regulations, 50–52
 - healthcare sector regulations, 52–53
 - key technical elements, 56–57
- limitations of, 57–58
- PCI DSS (Payment Card Industry Data Security Standard), 53–56
- corporate policies, 43–44
- export restrictions, 43
- importance of, 29
- legal concepts, 41
 - contracts, 41–42
 - MSAs (master service agreements), 42
 - NDAs (nondisclosure agreements), 43
 - SOWs (statements of work), 42
- pre-engagement documentation
 - budget, 32–33
 - communication escalation path, 31–32
 - confidentiality of findings, 32
 - disclaimers, 38–39
 - impact analysis and remediation timelines, 34–38
 - point-in-time assessment, 33
 - rules of engagement, 30
 - target audience, 29–30
 - technical constraints, 39
- risk management, 47–50
- scoping
 - assessment types, 45
 - importance of, 44
 - scope creep, 44
 - special considerations, 45–46
 - target selection, 46–47
 - strategy, 47
 - support resources, 40–41
- plugin.output node (Dradis), 486
- PNL (preferred network list) attacks, 189
- point-in-time assessments, 33
- policies
 - corporate, 43–44
 - Windows Group Policy, 305–306
- POODLE (Padding Oracle on Downgraded Legacy Encryption) vulnerability, 175

- POP3 (Post Office Protocol v3) port numbers, 159
- port numbers, 159
- port scans, Nmap, 391–393
 - half-open, 71–72
 - ping, 77–78
 - SYN, 71–72
 - TCP connect, 73–74
 - TCP FIN, 76–77
 - UDP, 74–75
 - Zenmap, 393–395
- POST method (HTTP), 217
- Post Office Protocol v3 (POP3) port numbers, 159
- post-engagement activities, 474–475.
 - See also* reports
- post-exploitation techniques
 - blind shells, creating, 338–344
 - C2 (command and control) utilities, 344–345
 - cleanup process, 356
 - custom daemons and processes, creating, 346
 - lateral movement, 347
 - post-exploitation scanning, 347–348
 - remote access protocols, 348–349
 - persistence, 337
 - reverse shells, creating, 338–344
 - scheduled tasks, creating, 346
 - users, creating, 346
- Windows legitimate utilities, 349
 - Empire, 353–354
 - PowerShell, 349–350
 - PowerSploit, 351–353
 - PSEexec, 355–356
 - Sysinternals, 355–356
 - WMI (Windows Management Instrumentation), 354–355
- PowerShell, 349–350, 433, 462
- PowerSploit, 351–353, 434
- PowerUp script, 352
- PowerView script, 352
- PR (Privilege Required) metrics, 37
- pre-engagement documentation
 - budget, 32–33
 - communication escalation path, 31–32
 - confidentiality of findings, 32
 - disclaimers, 38–39
 - impact analysis and remediation timelines, 34–38
 - point-in-time assessment, 33
 - rules of engagement, 30
 - target audience, 29–30
 - technical constraints, 39
- preferred network list (PNL) attacks, 189
- pretexting, 136
- primary account numbers (PANs), 54
- privilege escalation, 285–286
 - insecure SUDO implementations, 294–298
 - Linux permissions, 286–291
 - ret2libc (“return-to-libc”) attacks, 298
 - Unix programs, 291–293
- Windows privileges
 - clear-test credentials in LDAP, 300–301
 - container security, 310–314
 - CPassword, 299
 - DLL (dynamic link library) hijacking, 303–304
 - exploitable services, 304–305
 - Group Policy, 305–306
 - insecure file/folder permissions, 305
 - Kerberoast, 301
 - keyloggers, 306–307
 - LSASS (Local Security Authority Subsystem Service) credentials, 301–302
 - SAM (Security Account Manager) database, 302–303
 - sandbox escape, 308–310
 - scheduled tasks, 307–308
 - VM (virtual machine) escape, 310

Privilege Required (PR) metrics, 37
 PRNGs (pseudorandom number generators), 247
 ProcDump, 301–302
 procedures, stored, 239–240
 processes, creating, 346
 Professional Edition (Dradis), 479
 Project Summary screen (Dradis), 481
 protocol configurations, insecurity in, 281–284
 proxies (HTTP)
 defined, 214
 ZAP, 214
 Proxychains, 439
 ps command, 343
 pseudorandom number generators (PRNGs), 247
 PSEexec, 355–356
 PsExec tool (Sysinternals), 355
 PsFile tool (Sysinternals), 355
 PsGetSid tool (Sysinternals), 355
 PsInfo tool (Sysinternals), 355
 PsKill tool (Sysinternals), 355
 PsList tool (Sysinternals), 355
 PsLoggedOn tool (Sysinternals), 355
 PsLogList tool (Sysinternals), 355
 PsPassword tool (Sysinternals), 355
 PsPing tool (Sysinternals), 355
 PsService tool (Sysinternals), 355
 PsShutdownPsSuspend tool (Sysinternals), 355
 psudp, 441
 PTES (Penetration Testing Execution Standard), 13, 16
 PUT method (HTTP), 217
 pwd command, 343
 Python, 461

Q

QSAs (qualified security assessors), 54
 qualified security assessors (QSAs), 54
 Qualys scanner, 404

query throttling, 111
 QUIT command (SMTP), 161

R

race conditions, 266–267
 Radamsa, 459
 radio-frequency identification (RFID)
 attacks, 200
 rainbow tables, 244, 429
 RainbowCrack, 429–430
 ransomware
 Nyeta, 354
 WannaCry, 8
 Rapid7, 404
 RC (Report Confidence), 36
 RCPT command (SMTP), 160
 rcrack, 429–430
 RDP (Remote Desktop Protocol), 348, 433
 Reader (Adobe), 309
 Reaver, 197
 recommendations for remediation, reporting, 495–497
 reconnaissance. *See* active reconnaissance; passive reconnaissance
 Recon-*ng*, 90–102, 382–389
 hackertarget module, 96
 help menu, 92
 key list command, 96–97
 keys add command, 386
 keys list command, 386–387
 launching, 91
 main menu and splash page, 91
 searches, 95
 Shodan API, 96–102
 show info command, 387–388
 show modules command, 92–95, 383–386
 support resources, 389
 use command, 387–389
 red teams, 46

- redirect attacks, 249
- reflected DDoS (distributed denial-of-service) attacks, 178
- reflected XSS (cross-site scripting) attacks, 253–254
- regulations
 - financial sector, 50–52
 - healthcare sector, 52–53
 - PCI DSS (Payment Card Industry Data Security Standard), 53–56
- Remediation Level (RL), 36
- remediation timelines, 34–38
- remote access protocols, 348–349
- Remote Desktop Protocol (RDP), 348, 433
- remote file inclusion (RFI) vulnerabilities, 264–265
- Remove-Comments script, 351
- Remove-VolumeShadowCopy script, 352
- Report Confidence (RC), 36
- reporting/html module, 102
- reports
 - classifying, 499
 - common elements of, 490
 - executive summary, 493
 - findings and recommendations, 495–497
 - methodology, 494
 - metrics and measurements, 494–495
 - communications, 500–501
 - distribution, 499–500
- Dradis Framework
 - Choose a Tool dropdown, 484–485
 - data import, 483
 - imported scans, viewing, 488–490
 - launching, 479
 - login screen, 480
 - node organization, 486–488
 - node/subnode creation, 481–483
 - output file processing, 485–486
 - password creation, 479–480
 - plugin.output node, 486
- Project Summary screen, 481
- Upload Manager screen, 484
- Upload Output from Tool option, 483–484
- versions, 478–479
- handling, 499–500
- Offensive Security Example penetration test report, 497
- PCI DSS reporting guidelines, 491–493
- writing
 - best practices, 475, 476–478
 - importance of, 475–476
- Representational State Transfer (REST), 267
- request for proposal (RFP), 44
- requests (HTTP), 215–218
- res directory, 318
- resource command, 343
- resources, support, 40–41
 - CAPEC (Common Attack Pattern Enumeration and Classification), 114
 - CVE (Common Vulnerabilities and Exposures), 114–115
 - CWE (Common Weakness Enumeration), 115
 - JPCERT (Japan Computer Emergency Response Team), 113
 - NIST (National Institute of Standards and Technology), 113
 - US-CERT (U.S. Computer Emergency Readiness Team), 113
- resources.arsc directory, 318
- responses
 - HTTP (Hypertext Transfer Protocol), 215–218
- port scans
 - SYN scans, 71
 - TCP connect scans, 73
 - TCP FIN scans, 76
 - UDP scans, 75

- REST (Representational State Transfer), 267
RESTful (REST) APIs, 269
ret2libc (“return-to-libc”) attacks, 298
reverse shells, 238, 338–344
RF-based attacks. *See* wireless network vulnerabilities
RFI (remote file inclusion) vulnerabilities, 264–265
RFP (request for proposal), 44
risk, 47–50
 acceptance, 48
 appetite for risk, 49–50
 avoidance, 49
 mitigation, 48–49
 Risk Rating Methodology (OWASP), 495
 sharing, 49
 tolerance, 47–48
 transfer, 49
RL (Remediation Level), 36
Rlogin, 284
Rocket, 311
rockyou wordlist, 424
rogue access points, 185
rogue DHCP servers, 183–185
Root Me, 225
route manipulation attacks, 175–176
RSET command (SMTP), 161
Rsh, 284
Ruby, 461–462
rules of engagement, 30
run command, 451
Rwhod, 284
- S**
- S (Scope) metrics, 37
SAM (Security Account Manager)
 database, 302–303
Samba, 284
Samurai Web Testing Framework, 225
sandboxes, escaping, 308–310
SANS Institute InfoSec Reading Room, 493
SANS Investigative Forensic Toolkit (SIFT) Workstation, 458
Saved State Analysis, 302
scans, 391–393
 Nmap port scans
 half-open, 71–72
 ping, 77–78
 SYN, 71–72
 TCP connect, 73–74
 TCP FIN, 76–77
 UDP, 74–75
 post-exploitation, 347–348
 tools, 18–19
vulnerability, 103
 authenticated, 105
 challenges of, 109–112
 compliance, 109–110
 Dirbuster, 419
 discovery, 106
 full, 106–108
 how it works, 103–104
 management, 115–116
 Nessus, 403
 Nexpose, 403–404
 Nikto, 84, 410–413, 488–489
 OpenVAS, 401–403
 Qualys, 404
 results analysis, 112–113
 SQLmap, 404–410
 stealth, 108–109
 support resources, 113–115
 unauthenticated, 104–105
 W3AF, 415–419
ZAP (Zed Attack Proxy), 41, 214, 251, 413–414
vulnerability scans, 400
Scapy, 85–87
scapy command, 86
scarcity, social engineering and, 137

scheduled tasks
 creating, 346
 privilege escalation and, 307–308

Scope (S) metrics, 37

scoping. *See also* planning and preparation
 assessment types, 45
 importance of, 44
 scope creep, 44
 special considerations, 45–46
 target selection, 46–47

screenshot command, 448

scripts
Add-Persistence, 351
docker-bench-security, 313
Find-AVSignture, 351
Get-GPPAutologon, 352
Get-GPPPPassword, 352
Get-HttpStatus, 352
Get-Keystrokes, 352
Get-MicrophoneAudio, 352
Get-SecurityPackages, 351
Get-TimedScreenshot, 352
Get-VaultCredential, 352
Get-VolumeShadowCopy, 352
http-enum, 83–84
Install-SSP, 351
Invoke-CredentialInjection, 351
Invoke-DllInjection, 351
Invoke-Mimikatz, 352
Invoke-NinjaCopy, 352
Invoke-Portscan, 352
Invoke-ReflectivePEInjection, 351
Invoke-ReverseDnsLookup, 352
Invoke-Shellcode, 351
Invoke-TokenManipulation, 351
Invoke-WmiCommand, 351
Mount-VolumeShadowCopy, 352
New-ElevatedPersistenceOption, 351
New-UserPersistenceOption, 351
New-VolumeShadowCopy, 352
Out-CompressedDll, 351
Out-EncodedCommand, 351
Out-EncryptedScript, 351
Out-Minidump, 352
PowerUp, 352
PowerView, 352
Remove-Comments, 351
Remove-VolumeShadowCopy, 352
Set-CriticalProcess, 352
Set-MasterBootRecord, 352
smb-enum-groups, 81–82
smb-enum-processes, 85
smb-enum-shares, 82
smb-enum-users.nse, 80–81
smtp-open-relay, 160
SNMP-related, 158–159

SDKs (software development kits), 40

search command, 343

searches, *Recon-ng*, 95

SearchSploit, 151–154, 163–166

SEC (Securities and Exchange Commission), 52

Secure Computing Mode (*seccomp*), 309

Secure File Transfer Protocol (SFTP), 166

Secure SMTP (SSMTP) port number, 159

Securities and Exchange Commission (SEC), 52

Security Account Manager (SAM) database, 302–303

Security Enhanced Linux (SELinux), 293

security misconfigurations, 262
 cookie manipulation attacks, 263–264
 directory traversal vulnerabilities, 262–263

Security Onion, 369–370, 457

Security Requirements metrics, 36

Security Standards for the Protection of Electronic Protected Health Information. *See* HIPAA Security Rule

SELECT statement, 228

Select-String command, 349
SELinux (Security Enhanced Linux), 293
Sendmail, 284
sensors, egress, 327
SeraphimDroid, 323
serial console debugging, 326
Server Message Block (SMB), 8, 76, 151–155
servers, HTTP (Hypertext Transfer Protocol), 213
service dradis start command, 479
service providers, 54–55
service set identifiers (SSIDs), 46, 186
services
 enumeration, 85
 insecure configurations of, 281–284
session hijacking, 245–249
session riding, 260–261
session sniffing, 249
sessions
 HTTP (Hypertext Transfer Protocol), 213
 web, 221–224
SET (Social-Engineer Toolkit), 11–12, 129–134
set LHOST command, 155
set RHOST command, 155
Set-CriticalProcess script, 352
Set-MasterBootRecord script, 352
setoolkit command, 129
-sF option (nmap command), 76–77
SFI (software fault isolation), 309
SFTP (Secure File Transfer Protocol), 166
SGID (set-group-ID)
 Linux, 289
 Unix, 291–293
SHA-1 algorithm, 166
SHA-2 algorithm, 166
SHA-512 algorithm, 166
The Shadow Brokers, 8
shared_prefs folder, 319
sharing risk, 49
shell command, 343
shells
 bash, 460–461
 blind, 338–344
 reverse, 238, 338–344
Shodan API, 96–102, 378–380
shodan_hostname module, 100
shoulder surfing, 137
show info command, 387–388
show modules command, 383–386
show options command, 155
SIFT (SANS Investigative Forensic Toolkit) Workstation, 458
signal jamming, 189
silver ticket attacks (Kerberos), 172
Simple Network Management Protocol.
 See SNMP (Simple Network Management Protocol)
Simple Object Access Protocol (SOAP), 40, 267
Skadi, 457
Smali, 318
smalidea, 318
Smart Install, 281
SMB (Server Message Block), 8, 76, 151–155
SMB_COM_NEGOTIATE message, 80
SMB_COM_SESSION_SETUP_ANDX message, 80–81
smb-enum-groups script, 81–82
smb-enum-processes script, 85
smb-enum-shares script, 82
smb-enum-users.nse script, 80–81
SMS phishing, 134–135
SMTP (Simple Mail Transfer Protocol)
 commands, 160–163
 known SMTP server exploits, 163–166
 open relay, 160
 SMTPS (SMTP over SSL), 159
 TCP port numbers, 159
smtp-open-relay script, 160

- SMTSPS (SMTP over SSL), 159
- `smtp-user-enum` tool, 161–163
- `-sn` option (`nmap` command), 77–78
- sniffing, session, 249
- SNMP (Simple Network Management Protocol), 157–159
- SOAP (Simple Object Access Protocol), 40, 267
- `socat`, 345
- social engineering attacks, 11–12
 - characteristics of, 125–126
 - elicitation, 135
 - interrogation, 136
 - malvertising, 127–128
 - motivation techniques, 137
 - pharming, 126–127
 - phishing, 126
 - SMS, 134–135
 - spear, 128–134
 - voice phishing, 135
 - whaling, 135
 - pretexting, 136
- SET (Social-Engineer Toolkit), 129–134
 - shoulder surfing, 137
 - USB key drop, 138
- social engineering tests, 11–12, 129–134
- social proof, in social engineering, 137
- Social-Engineer Toolkit. *See SET (Social-Engineer Toolkit)*
- Social-Engineer Toolkit (SET), 11–12
- software. *See tools*
- software assurance tools, 458–459
- software development kits (SDKs), 40
- software fault isolation (SFI), 309
- SonarQube, 458
- source code comments, exploits in, 265–266
- SOWs (statements of work), 42
- spear phishing, 128–134
- Special Publication 800–57 (NIST), 57
- Special Publication 800–115 (NIST), 15
- spoofing
 - ARP, 173–175
 - DHCP (Dynamic Host Control Protocol), 183–185
- SQL injection vulnerabilities, 228
 - blind SQL injection, 237
 - Boolean technique, 233, 237
 - categories of, 232–234
 - database fingerprinting, 234–235
 - error-based technique, 233
 - examples of, 228–232
 - mitigations, 240
 - out-of-band technique, 233, 237–238
 - SQL statements, 228–232
 - stored procedures, 239–240
 - time-delay technique, 233, 239
 - UNION operator technique, 233, 235–236
- SQLi. *See SQL injection vulnerabilities*
- SQLmap, 404–410
- `-sS` option (`Nmap`), 71–72
- SSIDs (service set identifiers), 46, 186
- SSLStrip, 174
- SSMTP (Secure SMTP) port number, 159
- `-sT` option (`nmap` command), 73–74
- stack-smashing protection, 298
- STARTTLS, 159, 160
- starvation attacks (DHCP), 183–185
- statements (SQL), 228–232
- statements of work (SOWs), 42
- state-sponsored attackers, 10
- static binary analysis, 316
- stealth scans, 108–109
- sticky bits, 288
- stored DOM-based attacks, 263–264
- stored procedures, 239–240
- stored XSS (cross-site scripting) attacks, 255–256
- study plans (exam prep), 509
- `-sU` option (`nmap` command), 74–75
- `sudo` command, 286–287, 294–298

- SUID (set-user-ID)
 - Linux, 289
 - Unix, 291–293
- Sun Tzu, 9
- support resources, 40–41
 - CAPEC (Common Attack Pattern Enumeration and Classification), 114
 - CVE (Common Vulnerabilities and Exposures), 114–115
 - CWE (Common Weakness Enumeration), 115
 - intentionally vulnerable systems, 224–227
 - JPCERT (Japan Computer Emergency Response Team), 113
 - NIST (National Institute of Standards and Technology), 113
 - US-CERT (U.S. Computer Emergency Readiness Team), 113
- Swagger, 40, 268
- SYN flood attacks, 176
- SYN scans, 71–72
- sysinfo command, 449
- Sysinternals, 302, 355–356
- system cleanup, 356
- system diagrams, 41
- SYSTEM privileges, 304

- T**
- tables, rainbow, 244, 429
- tailgating, 327
- target audience, identification of, 29–30
- target selection, 46–47
- tasks, scheduled
 - creating, 346
 - privilege escalation, 307–308
- TCP (Transmission Control Protocol)
 - Nmap scans
 - TCP connect, 73–74
 - TCP FIN, 76–77
 - port numbers, 159
- TCPDUMP, 90, 215, 282–284
- teams
 - blue, 46
 - red, 46
- technical constraints, 39
- Telnet, 281
- temporal groups, 34–37
- tests
 - gray-box, 13
 - network infrastructure, 11
 - physical facility, 11
 - social engineering, 11–12
 - white-box, 12–13
 - wireless network, 11
- Theharvester, 376–378
- theoretical vulnerabilities, 38
- threat actors, 9–10
- threats. *See* attacks
- time of check to time of use (TOCTOU)
 - attacks, 266–267
- time-delay SQL injection, 233, 239
- timeline, remediation, 34–38
- Times, Tim, 90–91
- TOCTOU (time of check to time of use), 266–267
- tolerance, risk, 47–48
- tools, 18–19, 313. *See also* commands; scripts
 - ADIA (Appliance for Digital Investigation and Analysis), 457
 - AFL (American Fuzzy Lop), 459
 - Aircrack-ng suite, 186–189, 191–196
 - Aireplay-ng, 188, 191, 194–195
 - Airmon-ng, 186–187, 191, 194
 - Airodump-ng, 188, 191, 194–195
 - Anchore, 313
 - Androick, 323
 - Apktool, 322
 - Apple Remote Desktop, 433
 - Aqua Security, 313
 - Bane, 313
 - bash, 460–461

- BeEF, 449–450
- Burp, 214
- Cain and Abel, 424–425
- CAINE (Computer Aided Investigative Environment), 457
- Censys, 389–390
- CeWL, 431–432
- Clair, 313
- Cydia Substrate, 315
- Dagda, 313
- DEFT (Digital Evidence & Forensics Toolkit), 457
- DeNiSe, 441
- Dev-Sec.io, 313
- Dig, 371–372
- Dirbuster, 419
- dns2tcp, 441
- DNSScapy, 441
- DNScat, 441
- DNScat2, 345, 441
- DNSdumpster, 88
- DNSRecon, 67–69
- docker-bench-security, 313
- docker-explorer, 314
- Dradis Framework
 - Choose a Tool dropdown, 484–485
 - data import, 483
 - imported scans, viewing, 488–490
 - launching, 479
 - login screen, 480
 - node organization, 486–488
 - node/subnode creation, 481–483
 - output file processing, 485–486
 - password creation, 479–480
 - plugin.output node, 486
 - Project Summary screen, 481
 - Upload Manager screen, 484
 - Upload Output from Tool option, 483–484
 - versions, 478–479
- DropboxC2 (DBC2), 345
- edb debugger, 452–454
- Empire, 171, 353–354, 434
- Encryption, 439–440
- Enum4linux, 395–400
- Ettercap, 199
- ExifTool, 374–375
- Feederbot and Moto, 442
- Findbugs, 458
- Findsecbugs, 458
- FOCA (Fingerprinting Organizations with Collected Archives), 374
- Frida, 315
- GDB (GNU Project Debugger), 450–452
- Hashcat, 425–427
- Heyoka, 441
- Host, 371–372
- Hydra, 428–429
- IDA, 454–455
- iGoat, 325
- iMAS, 325
- Immunity, 454
- Iodine, 441
- John the Ripper, 420–425
- Johnny, 425
- Kerberoast, 301
- keyloggers, 306–307
- Lanman, 80–81
- Linux distributions, 224, 365
 - BlackArch Linux, 224, 367–368
 - CAINE (Computer Aided Investigative Environment), 369
 - Kali Linux, 224, 366
 - Parrot, 224, 367
 - Security Onion, 369–370
- Maltego, 381–382
- Medusa, 430–431
- Metasploit, 90–91, 115, 442–449
 - Metasploit Unleashed course, 344
 - Meterpreter, 299, 343–344, 446–449
 - RDP connections, creating, 348–349
- Mimikatz, 169–172, 432

MobiSec Live Environment Mobile Testing Framework, 325

Mutiny Fuzzing Framework, 459

Ncrack, 430–431

Nessus, 106–108, 403–404

Netcat, 338–342

Nikto, 84, 410–413, 488–489

Nmap. *See* Nmap

Notary, 314

Nslookup, 90, 156–157, 371–372

NTLM, 80–81

Objdump, 455–457

OllyDbg, 452–453

OpenVAS, 401–403

oscap-docker, 314

OzymanDNS and sods, 441

Packetforge-ng, 197–198

PALADIN, 457

Patator, 432–433

Peach, 459

Pearson Test Prep software, 505

- exam customization, 507–508
- exam updates, 508
- offline access, 506–507
- online access, 505
- Premium Edition, 508–509

PowerShell, 349–350, 433, 462

PowerSploit, 351–353, 434

ProcDump, 301

Proxychains, 439

PSEExec, 355–356

psudp, 441

Qualys scanner, 404

Radamsa, 459

RainbowCrack, 429–430

RDP (Remote Desktop Protocol), 433

Reaver, 197

Recon-ng, 90–102, 382–389

- hackertarget module, 96
- help menu, 92
- key list command, 96–97
- keys add command, 386

keys list command, 386–387

launching, 91

main menu and splash page, 91

searches, 95

Shodan API, 96–102

show info command, 387–388

show modules command, 92–95, 383–386

support resources, 389

use command, 387–389

Security Onion, 457

Shodan API, 96–102, 378–380

SIFT (SANS Investigative Forensic Toolkit) Workstation, 458

Skadi, 457

smtp-user-enum, 161–163

socat, 345

SonarQube, 458

SQLmap, 404–410

SSLStrip, 174

Sysinternals, 355–356

tcpdump, 90, 215, 282–284

Theharvester, 376–378

Tor, 438–439

TrevorC2, 345

Try-SQL Editor, 229

Tshark, 284

Twittor, 345

use cases for, 365

Veil, 434–437

vmss2core, 301

VNC, 433

W3AF scanner, 415–419

W3AFusage, 419

WebGoat, 225, 231, 254

Whois, 372–373

Windows Debugger, 452

Wireshark, 90, 216

WMI (Windows Management Instrumentation), 354–355

WMIImplant, 345

wsc2, 345

- X server forwarding, 433
- Xposed, 315
- ZAP (Zed Attack Proxy), 41, 214, 251, 413–414
- Zenmap, 393–395
- Tor, 438–439
- TRACE method (HTTP), 217
- transfer of risk, 49
- TrevorC2, 345
- Try2Hack, 225
- Try-SQL Editor, 229
- Tshark, 284
- tunneling, DNS (Domain Name System), 440–442
- Twittor, 345
- U**
- UDP (User Datagram Protocol), Nmap
 - UDP scans, 74–75
- UI (User Interaction) metrics, 37
- unauthenticated scans, 104–105
- uniform resource locators (URLs), 219–220
- unilateral NDAs (nondisclosure agreements), 43
- UNION operator, 233, 235–236
- Unix, privilege escalation in, 291–293
- unprotected APIs, 267–270
- Unstructured Supplementary Service Data (USSD), 323
- Update Products button (Pearson Test Prep software), 508
- UPDATE statement, 228
- updating Pearson Test Prep software, 508
- upload command, 343
- Upload Manager screen (Dradis), 484
- Upload Output from Tool option (Dradis), 483–484
- urgency, in social engineering, 137
- URLs (uniform resource locators), 219–220
- USB key drops, 138
- US-CERT (U.S. Computer Emergency Readiness Team), 113
- use cases, 365
- use command, 387–389
- use exploit/windows/smb/ms17_010_ eternalblue command, 155
- user enumeration, 80–81
- User Interaction (UI) metrics, 37
- useradd command, 295
- usermod command, 294–295
- users, creating, 346
- USSD (Unstructured Supplementary Service Data), 323
- utilities. *See* tools

V

- Vanhoef, Mathy, 196
- Veil, 434–437
- verbose error handling, 266
- Vicnum, 225
- visudo command, 296
- VLANs (virtual LANs), hopping, 181–183
- VMs (virtual machines)
 - containers compared to, 311–312
 - dumping memory from, 301
 - escaping, 310
 - .vmsn file extension, 301
 - .vmss file extension, 301
 - vmss2core, 301–302
- VMware Snapshot, 302
- VNC, 348, 433
- voice phishing, 135
- Volatility Foundation, 302
- Volatility Framework, 301
- VRFY command (SMTP), 161
- vulnerability management, 115–116
 - chaining analysis, 37–38
 - impact analysis, 34–37
 - theoretical vulnerabilities, 38
- vulnerability scans, 103, 400

authenticated, 105
challenges of, 109–112
compliance, 109–110
Dirbuster, 419
discovery, 106
full, 106–108
how it works, 103–104
Nessus, 403–404
Nikto, 84, 410–413, 488–489
OpenVAS, 401–403
Qualys, 404
results analysis, 112–113
SQLmap, 404–410
stealth, 108–109
support resources, 113–115
tools, 18–19
unauthenticated, 104–105
vulnerability management, 115–116
W3AF, 415–419
ZAP (Zed Attack Proxy), 41, 214, 251, 413–414

W

W3AF scanner, 415–419
W3AFusage, 419
W3Schools, 218, 229
WADL (Web Application Description Language), 40, 268
WAFs (web application firewalls), 46
WannaCry, 8
war driving, 190
Wassenaar Arrangement, 439
weak cryptographic algorithms, 243–244
Web Application Description Language (WADL), 40, 268
web applications
authentication-based vulnerabilities
 credential brute forcing, 243–245
 default credential exploits, 249–250
 Kerberos exploits, 250
 redirect attacks, 249
 session hijacking, 245–249

authorization-based vulnerabilities
 Insecure Direct Object Reference vulnerabilities, 251–252
 parameter pollution, 250–251
clickjacking, 261
command injection vulnerabilities, 241–242
CSRF (cross-site request forgery), 260–261
enumeration, 83–84
file inclusion vulnerabilities
 LFI (local file inclusion), 264
 RFI (remote file inclusion), 264–265
HTML injection vulnerabilities, 241
HTTP (Hypertext Transfer Protocol), 213–221
 clients, 213
 proxies, 214
 request/response model, 215–218
 servers, 213
 sessions, 213
URLs (uniform resource locators), 219–220
insecure code practices
 code signing, lack of, 270
 error-handling errors, 266
 hard-coded credentials, 266
 hidden elements, 270
 race conditions, 266–267
 source code comments, 265–266
 unprotected APIs, 267–270
labs for, 224–227
security misconfigurations, 262
 cookie manipulation attacks, 263–264
 directory traversal vulnerabilities, 262–263
SQL injection vulnerabilities, 228
 blind SQL injection, 237
 Boolean technique, 233, 237
 categories of, 232–234
 database fingerprinting, 234–235

- error-based technique, 233
- examples of, 228–232
- mitigations, 240
- out-of-band technique, 233, 237–238
- stored procedures, 239–240
- time-delay technique, 233, 239
- UNION operator technique, 233, 235–236
- tests, 11
- WAFs (web application firewalls), 46
- web sessions, 221–224
- XSS (cross-site scripting) vulnerabilities, 252–253
- DOM-based XSS attacks, 256–257
- evasion techniques, 257–258
- mitigations, 258–259
- reflected XSS attacks, 253–254
- stored XSS attacks, 255–256
- web browsers, 309
- web form-grabbing keyloggers, 307
- web page enumeration, 83–84
- Web Security Dojo, 225, 227
- Web Services Description Language (WSDL), 40, 268
- web sessions, 221–224
- webcam_list command, 344
- webcam_snap command, 344
- WebGoat, 225, 231, 254
- WEP (Wired Equivalent Privacy) attacks, 190–192
- whaling, 135
- white lists, 46
- white-box tests, 12–13, 47
- Whois, 372–373
- Wi-Fi Protected Access (WPA) attacks, 192–196
- Wi-Fi Protected Setup (WPS), 197
- WiGLE, 190
- Windows
 - Debugger, 452
 - legitimate utilities for post-exploitation tasks, 349
 - Empire, 353–354
 - PowerShell, 349–350
 - PowerSploit, 351–353
 - PSEExec, 355–356
 - Sysinternals, 355–356
 - WMI (Windows Management Instrumentation), 354–355
- privilege escalation
 - clear-test credentials in LDAP, 300–301
 - container security, 310–314
 - CPassword, 299
 - DLL (dynamic link library)
 - hijacking, 303–304
 - exploitable services, 304–305
 - Group Policy, 305–306
 - insecure file/folder permissions, 305
 - Kerberoast, 301
 - keyloggers, 306–307
 - LSASS (Local Security Authority Subsystem Service) credentials, 301–302
 - SAM (Security Account Manager) database, 302–303
 - sandbox escape, 308–310
 - scheduled tasks, 307–308
 - VM (virtual machine) escape, 310
- Sysinternals, 302
- WinRM (Windows Remote Management), 354
- WMI (Windows Management Instrumentation), 172, 354–355
- WinRM (Windows Remote Management), 354
- Wired Equivalent Privacy (WEP), 190–192, 243
- wireless adapters, 189
- wireless network tests, 459
- wireless network vulnerabilities
 - credential harvesting, 199–200
 - deauthentication attacks, 186–189
 - evil twin attacks, 185–186

- fragmentation attacks, 197–198
 - IV (initialization vector) attacks, 190
 - KARMA attacks, 197
 - KRACK (key reinstallation attack), 196–197
 - network tests, 11
 - PNL (preferred network list) attacks, 189
 - RFID (radio-frequency identification) attacks, 200
 - rogue access points, 185
 - signal jamming, 189
 - war driving, 190
 - WEP (Wired Equivalent Privacy) attacks, 190–192
 - WPA (Wi-Fi Protected Access) attacks, 192–196
 - WPS (Wi-Fi Protected Setup), 197
 - Wireshark, 90, 216
 - WMI (Windows Management Instrumentation), 172, 354–355
 - WMIImplant, 345
 - wordlists
 - creating with CeWL, 431–432
 - defined, 423
 - rockyou, 424
 - workgroups, 150
 - WPA (Wi-Fi Protected Access) attacks, 192–196
 - WPS (Wi-Fi Protected Setup), 197
 - “Writing a Penetration Testing Report” whitepaper, 493
 - writing reports
 - best practices, 475, 476–478
 - importance of, 475–476
 - wsc2, 345
 - WSDL (Web Services Description Language), 40, 268
- X**
- X server forwarding, 348, 433
 - x86 file, 319
 - x86_64 file, 319
 - XN (eXecute Never), 324
 - XPosed, 315
 - XSS (cross-site scripting) vulnerabilities, 252–253
 - DOM-based XSS attacks, 256–257
 - evasion techniques, 257–259
 - reflected XSS attacks, 253–254
 - stored XSS attacks, 255–256
- Y**
- Yppasswdd, 284
 - Ypserv, 284
 - Ypxfrd, 284
- Z**
- ZAP (Zed Attack Proxy), 41, 214, 251, 413–414
 - Zenmap, 393–395
 - zero-day attacks, 8
 - Zygote, 319–320