Cert Guide
Advance your IT career with hands-on learning

CompTIA®
Cybersecurity Analyst (CySA+)
CS0-002

TROY McMILLAN
About the Author

Troy McMillan is a product developer and technical editor for Kaplan IT as well as a full-time trainer. He became a professional trainer 20 years ago, teaching Cisco, Microsoft, CompTIA, and wireless classes. He has written or contributed to more than a dozen projects, including the following recent ones:

- Contributing subject matter expert for *CCNA Cisco Certified Network Associate Certification Exam Preparation Guide* (Kaplan)
- Author of *CISSP Cert Guide* (Pearson)
- Author of *CompTIA Advanced Security Practitioner (CASP) Cert Guide* (Pearson)

Troy has also appeared in the following training videos for OnCourse Learning: Security+; Network+; Microsoft 70-410, 411, and 412 exam prep; ICND1; and ICND2.

He delivers CISSP training classes for CyberVista, and is an authorized online training provider for (ISC)2.

Troy also creates certification practice tests and study guides for CyberVista. He lives in Asheville, North Carolina, with his wife, Heike.
Analyzing Assessment Output

When assessments are performed there will be data that is gathered that must be analyzed. The format of the output generated by the various tools used to perform the vulnerability assessment may be intuitive, but in many cases it is not. Analysts must be able to read and correctly interpret the output to identify issues that may exist. This chapter is dedicated to analyzing vulnerability assessment output.

“Do I Know This Already?” Quiz

The “Do I Know This Already?” quiz enables you to assess whether you should read the entire chapter. If you miss no more than one of these six self-assessment questions, you might want to skip ahead to the “Exam Preparation Tasks” section. Table 4-1 lists the major headings in this chapter and the “Do I Know This Already?” quiz questions covering the material in those headings so that you can assess your knowledge of these specific areas. The answers to the “Do I Know This Already?” quiz appear in Appendix A.

<table>
<thead>
<tr>
<th>Foundation Topics Section</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Application Scanner</td>
<td>1</td>
</tr>
<tr>
<td>Infrastructure Vulnerability Scanner</td>
<td>2</td>
</tr>
<tr>
<td>Software Assessment Tools and Techniques</td>
<td>3</td>
</tr>
<tr>
<td>Enumeration</td>
<td>4</td>
</tr>
<tr>
<td>Wireless Assessment Tools</td>
<td>5</td>
</tr>
<tr>
<td>Cloud Infrastructure Assessment Tools</td>
<td>6</td>
</tr>
</tbody>
</table>

1. Which of the following is a type of proactive monitoring and uses external agents to run scripted transactions against an application?
   a. RUM
   b. Synthetic transaction monitoring
2. Which of the following is an example of a cloud-based vulnerability scanner?
   a. OpenVAS
   b. Qualys
   c. Nikto
   d. NESSUS

3. Which step in the software development life cycle (SDLC) follows the design step?
   a. Gather requirements
   b. Certify/accredit
   c. Develop
   d. Test/validate

4. Which of the following is the process of discovering and listing information?
   a. Escalation
   b. Discovery
   c. Enumeration
   d. Penetration

5. Which of the following is a set of command-line tools you can use to sniff WLAN traffic?
   a. hping3
   b. Aircrack-ng
   c. Qualys
   d. Reaver

6. Which of the following is a data collection tool that allows you to use longitudinal survey panels to track and monitor the cloud environment?
   a. Prowler
   b. ScoutSuite
   c. Pacu
   d. Mikto
Foundation Topics

Web Application Scanner

Web vulnerability scanners focus on discovering vulnerabilities in web applications. These tools can operate in two ways: synthetic transaction monitoring or real user monitoring. In synthetic transaction monitoring, preformed (synthetic) transactions are performed against the web application in an automated fashion, and the behavior of the application is recorded. In real user monitoring, real user transactions are monitored while the web application is live.

Synthetic transaction monitoring, which is a type of proactive monitoring, uses external agents to run scripted transactions against a web application. This type of monitoring is often preferred for websites and applications. It provides insight into the application’s availability and performance and warns of any potential issue before users experience any degradation in application behavior. For example, Microsoft’s System Center Operations Manager (SCOM) uses synthetic transactions to monitor databases, websites, and TCP port usage.

In contrast, real user monitoring (RUM), which is a type of passive monitoring, captures and analyses every transaction of every web application or website user. Unlike synthetic transaction monitoring, which attempts to gain performance insights by regularly testing synthetic interactions, RUM cuts through the guesswork, seeing exactly how users are interacting with the application.

Many web application scanners are available. These tools scan an application for common security issues with cookie management, PHP scripts, SQL injections, and other problems. Some examples of these tools are covered in this section.

Burp Suite

The Burp Suite is a suite of tools, one of which can be used for testing web applications. It can scan an application for vulnerabilities and can also be used to crawl an application (to discover content). This commercial software is available for Windows, Linux, and macOS. It can also be used for exploiting vulnerabilities. For more information, see https://portswigger.net/burp.

OWASP Zed Attack Proxy (ZAP)

The Open Web Application Security Project (OWASP) produces an interception proxy called OWASP Zed Attack Proxy (ZAP). It performs many of the same functions as Burp, and so it also falls into the exploit category. It can monitor the traffic
between a client and a server, crawl the application for content, and perform vulnerability scans. For more information, see https://owasp.org/www-project-zap/.

Nikto

*Nikto* is a vulnerability scanner that is dedicated to web servers. It is designed for Linux but can be run in Windows through a Perl interpreter. This tool is not stealthy, but it is a fast scanner. Everything it does is recorded in your logs. It generates a lot of information, much of it normal or informational. It is a command-line tool that is often run from within a Kali Linux server and preinstalled with more than 300 penetration-testing programs. For more information, see https://tools.kali.org/information-gathering/nikto.

Arachni

*Arachni* is a Ruby framework for assessing the security of a web application. It is often used by penetration testers. It is open source, works with all major operating systems (Windows, macOS, and Linux), and is distributed via portable packages that allow for instant deployment. Arachni can be used either at the command line or via the web interface, shown in Figure 4-1.
Infrastructure Vulnerability Scanner

An infrastructure vulnerability scanner probes for a variety of security weaknesses, including misconfigurations, out-of-date software, missing patches, and open ports. These solutions can be on premises or cloud based.

Infrastructure vulnerability scanners were covered in detail in Chapter 18.

Nessus

One of the most widely used vulnerability scanners is Nessus Professional, a proprietary tool developed by Tenable Network Security. It is free of charge for personal use in a non-enterprise environment. By default, Nessus Professional starts by listing at the top of the output the issues found on a host that are rated with the highest severity, as shown in Figure 4-2.

![Example Nessus Output](image)

For the computer scanned in Figure 4-2, you can see that there is one high-severity issue (the default password for a Firebird database located on the host), and there are five medium-level issues, including two SSL certificates that cannot be trusted and a remote desktop man-in-the-middle attack vulnerability. For more information, see https://www.tenable.com/products/nessus.

OpenVAS

As you might suspect from the name, the OpenVAS tool is open source. It was developed from the Nessus code base and is available as a package for many Linux distributions. The scanner is accompanied with a regularly updated feed of network vulnerability tests (NVT). It uses the Greenbone console, shown in Figure 4-3. For more information, see https://www.openvas.org/.
Software Assessment Tools and Techniques

Many organizations create software either for customers or for their own internal use. When software is developed, the earlier in the process security is considered, the less it will cost to secure the software. It is best for software to be secure by design. Secure coding standards are practices that, if followed throughout the software development life cycle (SDLC), help to reduce the attack surface of an application.

In Chapter 9, “Software Assurance Best Practices,” you will learn about the SDLC, a set of ordered steps to help ensure that software is developed to enhance both security and functionality. As a quick preview, the SDLC steps are listed here:

**Step 1.** Plan/initiate project

**Step 2.** Gather requirements

**Step 3.** Design

**Step 4.** Develop

**Step 5.** Test/validate
Step 6. Release/maintain

Step 7. Certify/accredit

Step 8. Perform change management and configuration management/replacement

This section concentrates on Steps 5 and 7, which is where testing of the software occurs. This testing is covered in this chapter because it is a part of vulnerability management. This testing or validation can take many forms.

Static Analysis

Static code analysis is performed without the code executing. Code review and testing must occur throughout the entire SDLC. Code review and testing must identify bad programming patterns, security misconfigurations, functional bugs, and logic flaws.

Code review and testing in the planning and design phases include architecture security reviews and threat modeling. Code review and testing in the development phase include static source code analysis and manual code review and static binary code analysis and manual binary review. Once an application is deployed, code review and testing involve penetration testing, vulnerability scanning, and fuzz testing.

Static code review can be done with scanning tools that look for common issues. These tools can use a variety of approaches to find bugs, including the following:

- **Data flow analysis:** This analysis looks at runtime information while the software is in a static state.
- **Control flow graph:** A graph of the components and their relationships can be developed and used for testing by focusing on the entry and exit points of each component or module.
- **Taint analysis:** This analysis attempts to identify variables that are tainted with user-controllable input.
- **Lexical analysis:** This analysis converts source code into tokens of information to abstract the code and make it easier to manipulate for testing purposes.

Code review is the systematic investigation of the code for security and functional problems. It can take many forms, from simple peer review to formal code review. There are two main types of reviews:

- **Formal review:** This is an extremely thorough, line-by-line inspection, usually performed by multiple participants using multiple phases. This is the most time-consuming type of code review but the most effective at finding defects.
Lightweight: This type of code review is much more cursory than a formal review. It is usually done as a normal part of the development process. It can happen in several forms:

- **Pair programming:** Two coders work side by side, checking one another’s work as they go.
- **Email:** Code is emailed around to colleagues for them to review when time permits.
- **Over the shoulder:** Coworkers review the code while the author explains his or her reasoning.
- **Tool-assisted:** Perhaps the most efficient method, this method uses automated testing tools.

While code review is most typically performed on in-house applications, it may be warranted in other scenarios as well. For example, say that you are contracting with a third party to develop a web application to process credit cards. Considering the sensitive nature of the application, it would not be unusual for you to request your own code review to assess the security of the product.

In many cases, more than one tool should be used in testing an application. For example, an online banking application that has had its source code updated should undergo both penetration testing with accounts of varying privilege levels and a code review of the critical models to ensure that defects there do not exist.

**Dynamic Analysis**

*Dynamic analysis* is testing performed while the software is running. This testing can be performed manually or by using automated testing tools. There are two general approaches to dynamic testing:

- **Synthetic transaction monitoring:** A type of proactive monitoring, often preferred for websites and applications. It provides insight into the application’s availability and performance, warning of any potential issue before users experience any degradation in application behavior. It uses external agents to run scripted transactions against an application. For example, Microsoft’s System Center Operations Manager (SCOM) uses synthetic transactions to monitor databases, websites, and TCP port usage.

- **Real user monitoring (RUM):** A type of passive monitoring that captures and analyzes every transaction of every application or website user. Unlike synthetic monitoring, which attempts to gain performance insights by regularly testing synthetic interactions, RUM cuts through the guesswork by analyzing exactly how your users are interacting with the application.
Reverse Engineering

In 1990, the Institute of Electrical and Electronics Engineers (IEEE) defined reverse engineering as “the process of analyzing a subject system to identify the system’s components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction,” where the “subject system” is the end product of software development.

Reverse engineering techniques can be applied in several areas, including the study of the security of in-house software. In Chapter 16, “Applying the Appropriate Incident Response Procedure,” you’ll learn how reverse engineering is applied to the incident response procedure. In Chapter 12, “Implementing Configuration Changes to Existing Controls to Improve Security,” you’ll learn how reverse engineering applies to the malware analysis process. The techniques you will learn about in those chapters can also be used to locate security issues with in-house software.

Fuzzing

Fuzz testing, or fuzzing, involves injecting invalid or unexpected input (sometimes called faults) into an application to test how the application reacts. It is usually done with a software tool that automates the process. Inputs can include environment variables, keyboard and mouse events, and sequences of API calls. Figure 4-4 shows the logic of the fuzzing process.

Two types of fuzzing can be used to identify susceptibility to a fault injection attack:

- **Mutation fuzzing**: Involves changing the existing input values (blindly)
- **Generation-based fuzzing**: Involves generating the inputs from scratch, based on the specification/format
The following measures can help prevent fault injection attacks:

- Implement fuzz testing to help identify problems.
- Adhere to safe coding and project management practices.
- Deploy application-level firewalls.

**Enumeration**

*Enumeration* is the process of discovering and listing information. Network enumeration is the process of discovering pieces of information that might be helpful in a network attack or compromise. There are several techniques used to perform enumeration and several tools that make the process easier for both testers and attackers. Let’s take a look at these techniques and tools.

**Nmap**

While network scanning can be done with more blunt tools, like ping, *Nmap* is stealthier and may be able to perform its activities without setting off firewalls and IDSs. It is valuable to note that while we are discussing Nmap in the context of network scanning, this tool can be used for many other operations, including performing certain attacks. When used for scanning, it typically locates the devices, locates the open ports on the devices, and determines the OS on each host.

After performing Nmap scans with certain flags set in the scan packets, security analysts (and hackers) can make certain assumptions based on the responses received. These flags are used to control the TCP connection process and so are present only in those packets. Figure 4-5 show a TCP header with the important flags circled. Normally flags are “turned on” as a result of the normal TCP process, but a hacker can craft packets to check the flags he wants to check.

![TCP Header](image)

**FIGURE 4-5** TCP Header

Figure 4-5 shows these flags, among others:

- **URG**: Urgent pointer field significant
- **ACK**: Acknowledgment field significant
After performing Nmap scans with certain flags set in the scan packets, security analysts (and hackers) can make certain assumptions based on the responses received.

Nmap exploits weaknesses with three scan types:

- **Null scan**: A Null scan is a series of TCP packets that contain a sequence number of 0 and no set flags. Because the Null scan does not contain any set flags, it can sometimes penetrate firewalls and edge routers that filter incoming packets with particular flags. When such a packet is sent, two responses are possible:
  - **No response**: The port is open on the target.
  - **RST**: The port is closed on the target.

Figure 4-6 shows the result of a Null scan using the command `nmap -sN`. In this case, `nmap` received no response but was unable to determine whether that was because a firewall was blocking the port or the port was closed on the target. Therefore, it is listed as open|filtered.

![图4-6 Null Scan](image)
**FIN scan:** This type of scan sets the FIN bit. When this packet is sent, two responses are possible:

- **No response:** The port is open on the target.
- **RST/ACK:** The port is closed on the target.

Example 4-1 shows sample output of a FIN scan using the command `nmap -sF`, with the `-v` included for verbose output. Again, nmap received no response but was unable to determine whether that was because a firewall was blocking the port or the port was closed on the target. Therefore, it is listed as open|filtered.

---

Example 4-1  
**FIN Scan Using nmap -sF**

```
# nmap -sF -v 192.168.0.7

Starting nmap 3.81 at 2016-01-23 21:17 EDT
Initiating FIN Scan against 192.168.0.7 [1663 ports] at 21:17
The FIN Scan took 1.51s to scan 1663 total ports.
Host 192.168.0.7 appears to be up ... good.
Interesting ports on 192.168.0.7:
(The 1654 ports scanned but not shown below are in state: closed)

<table>
<thead>
<tr>
<th>PORT</th>
<th>STATE</th>
<th>SERVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>21/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>22/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>23/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>79/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>110/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>111/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>514/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>886/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
<tr>
<td>2049/tcp</td>
<td>open</td>
<td>filtered</td>
</tr>
</tbody>
</table>

MAC Address: 00:03:47:6D:28:D7 (Intel)

Nmap finished: 1 IP address (1 host up) scanned in 2.276 seconds
Raw packets sent: 1674 (66.9KB) | Rcvd: 1655 (76.1KB)
```

---

**XMAS scan:** This type of scan sets the FIN, PSH, and URG flags. When this packet is sent, two responses are possible:

- **No response:** The port is open on the target.
- **RST:** The port is closed on the target.
Figure 4-7 shows the result of this scan, using the command `nmap -sX`. In this case `nmap` received no response but was unable to determine whether that was because a firewall was blocking the port or the port was closed on the target. Therefore, it is listed as open|filtered.

![FIGURE 4-7 XMAS Scan](image)

Null, FIN, and XMAS scans all serve the same purpose, to discover open ports and ports blocked by a firewall, and differ only in the switch used. While there are many more scan types and attacks that can be launched with Nmap, these scan types are commonly used during environmental reconnaissance testing to discover what the hacker might discover and take steps to close any gaps in security before the hacker gets there. For more information on Nmap, see [https://nmap.org/](https://nmap.org/).

**Host Scanning**

*Host scanning* involves identifying the live hosts on a network or in a domain namespace. Nmap and other scanning tools (such as ScanLine and SuperScan) can be used for this. Sometimes called a ping scan, a host scan records responses to pings sent to every address in the network. You can also combine a host scan with a port scan by using the proper arguments to the command. During environmental reconnaissance testing, you can make use of these scanners to identify all live hosts. You may discover hosts that shouldn’t be there. To execute this scan from `nmap`, the command is `nmap -SP 192.168.0.0-100`, where 0-100 is the range of IP addresses to be scanned in the 192.168.0.0 network. Figure 4-8 shows an example of the output from this command. This command's output lists all devices that are on. For each one, the MAC address is also listed.
**hping**

hping (and the newer version, hping3) is a command-line-oriented TCP/IP packet assembler/analyzer that goes beyond simple ICMP echo requests. It supports TCP, UDP, ICMP, and RAW-IP protocols and also has a traceroute mode. The following is a subset of the operations possible with **hping**:

- Firewall testing
- Advanced port scanning
- Network testing, using different protocols, TOS, fragmentation
- Manual path MTU discovery
- Advanced traceroute, under all the supported protocols
- Remote OS fingerprinting
- Remote uptime guessing
- TCP/IP stacks auditing

What is significant about hping is that it can be used to create or assemble packets. Attackers use packet assembly tools to create packets that allow them to mount attacks. Testers can also use hping to create malicious packets to assess the response of the network defenses or to identify vulnerabilities that may exist.

A common attack is a DoS attack using what is called a **SYN flood**. In this attack, the target is overwhelmed with unanswered SYN/ACK packets. The device answers each SYN packet with a SYN-ACK. Since devices reserve memory for the expected response to the SYN-ACK packet, and since the attacker never answers, the target system eventually runs out of memory, making it essentially a dead device. This scenario is shown in Figure 4-9.
Example 4-2 demonstrates how to deploy a SYN flood by executing the `hping` command at the terminal.

**Example 4-2**  Deploying a SYN Flood with `hping`

```
$ sudo hping3 -i u1 -S -p 80 -c 10  192.168.1.1
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 0 data bytes
--- 192.168.1.1 hping statistic ---
10 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The command in Example 4-2 would send TCP SYN packets to 192.168.1.1. Including `sudo` is necessary because `hping3` creates raw packets for the task. For raw sockets/packets, root privilege is necessary on Linux. The parts of the command and the meaning of each are described as follows:

- **i u1** means wait for 1 microsecond between each packet
- **S** indicates SYN flag
- **p 80** means target port 80
- **c 10** means send 10 packets
Were this a true attack, you would expect to see many more packets sent; however, you can see how this tool can be used to assess the likelihood that such an attack would succeed. For more information, see https://tools.kali.org/information-gathering/hping3.

**Active vs. Passive**

Chapter 3, “Vulnerability Management Activities,” covered active and passive scanning. The concept of active and passive enumeration is similar. **Active enumeration** is when you send packets of some sort to the network and then assess responses. An example of this would be using nmap to send crafted packets that interrogate the accessibility of various ports (port scan). **Passive enumeration** does not send packets of any type but captures traffic and makes educated assumptions from the traffic. An example is using a packet capture utility (sniffer) to look for malicious traffic on the network.

**Responder**

Link-Local Multicast Name Resolution (LLMNR) and NetBIOS Name Service (NBT-NS) are Microsoft Windows components that serve as alternate methods of host identification. **Responder** is a tool that can be used for a number of things, among them answering NBT and LLMNR name requests. Doing this poisons the service so that the victims communicate with the adversary-controlled system. Once the name system is compromised, Responder captures hashes and credentials that are sent to the system after the name services have been poisoned.

Figure 4-10 shows that after the target was convinced to talk to Responder, it was able to capture the hash sent for authentication, which could then be used to attempt to crack the password.

**Wireless Assessment Tools**

To assess wireless networks for vulnerabilities, you need tools that can use wireless antennas and sensors to capture and examine the wireless traffic. As a security
professional tasked with identifying wireless vulnerabilities, you must also be familiar with the tools used to compromise wireless networks. Let’s discuss some of these tools.

**Aircrack-ng**

*Aircrack-ng* is a set of command-line tools you can use to sniff wireless networks, among other things. Installers for this tool are available for both Linux and Windows. It is important to ensure that your device’s wireless chipset and driver support this tool.

Aircrack-ng focuses on these areas of Wi-Fi security:

- **Monitoring**: Packet capture and export of data to text files for further processing by third-party tools
- **Attacking**: Replay attacks, deauthentication, fake access points, and others via packet injection
- **Testing**: Checking Wi-Fi cards and driver capabilities (capture and injection)
- **Cracking**: WEP and WPA PSK (WPA1 and 2)

As you can see, capturing wireless traffic is a small part of what this tool can do. The command for capturing is `airodump-ng`.

Figure 4-11 shows Aircrack-ng being used to attempt to crack an encryption key. It attempted 1514 keys before locating the correct one. For more information on Aircrack-ng, see https://www.aircrack-ng.org/.

![Aircrack-ng screenshot](image-url)
Reaver

**Reaver** is both a package of tools and a command-line tool within the package called **reaver** that is used to attack Wi-Fi Protected Setup (WPS). Example 4-3 shows the **reaver** command and its arguments.

**Example 4-3  Reaver: Wi-Fi Protected Setup Attack Tool**

```
root@kali:~# reaver -h
Reaver v1.6.5 WiFi Protected Setup Attack Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner
<cheffner@tacnetsol.com

Required Arguments:
- **-i**, --interface=<wlan>  Name of the monitor-mode interface to use
- **-b**, --bssid=<mac>  BSSID of the target AP

Optional Arguments:
- **-m**, --mac=<mac>  MAC of the host system
- **-e**, --essid=<ssid>  ESSID of the target AP
- **-c**, --channel=<channel>  Set the 802.11 channel for the interface (implies -f)
- **-s**, --session=<file>  Restore a previous session file
- **-C**, --exec=<command>  Execute the supplied command upon successful pin recovery
- **-f**, --fixed  Disable channel hopping
- **-5**, --5ghz  Use 5GHz 802.11 channels
- **-v**, --verbose  Display non-critical warnings (-vv or -vvv for more)
- **-q**, --quiet  Only display critical messages
- **-h**, --help  Show help

Advanced Options:
- **-p**, --pin=<wps pin>  Use the specified pin (may be arbitrary string or 4/8 digit WPS pin)
- **-d**, --delay=<seconds>  Set the delay between pin attempts [1]
- **-l**, --lock-delay=<seconds>  Set the time to wait if the AP locks WPS pin attempts [60]
- **-g**, --max-attempts=<num>  Quit after num pin attempts
- **-x**, --fail-wait=<seconds>  Set the time to sleep after 10 unexpected failures [0]
- **-r**, --recurring-delay=<x:y>  Sleep for y seconds every x pin attempts
```
-t, --timeout=<seconds>  Set the receive timeout period [10]
-T, --m57-timeout=<seconds>  Set the M5/M7 timeout period [0.40]
-A, --no-associate  Do not associate with the AP  
  (association must be done by another application)
-N, --no-nacks  Do not send NACK messages when out of order packets are received
-S, --dh-small  Use small DH keys to improve crack speed
-L, --ignore-locks  Ignore locked state reported by the target AP
-E, --eap-terminate  Terminate each WPS session with an EAP FAIL packet
-J, --timeout-is-nack  Treat timeout as NACK (DIR-300/320)
-F, --ignore-fcs  Ignore frame checksum errors
-w, --win7  Mimic a Windows 7 registrar [False]
-K, --pixie-dust  Run pixiedust attack
-Z  Run pixiedust attack

Example:
    reaver -i wlan0mon -b 00:90:4C:C1:AC:21 -vv

The Reaver package contains other tools as well. Example 4-4 shows the arguments for the wash command of the Wi-Fi Protected Setup Scan Tool. For more information on Reaver, see https://tools.kali.org/wireless-attacks/reaver.

### Example 4-4  wash: Wi-Fi Protected Setup Scan Tool

```
root@kali:~# wash -h

Wash v1.6.5 WiFi Protected Setup Scan Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner

Required Arguments:
    -i, --interface=<iface>  Interface to capture packets on
    -f, --file [FILE1 FILE2 FILE3 ...]  Read packets from capture files

Optional Arguments:
    -c, --channel=<num>  Channel to listen on [auto]
    -n, --probes=<num>  Maximum number of probes to send to each AP in scan mode [15]
```
oclHashcat

oclHashcat is a general-purpose computing on graphics processing units (GPGPU)-based multi-hash cracker using a brute-force attack. All versions have now been updated and are simply called hashcat. In GPGPU, the graphics processor is tasked with running the algorithms that crack the hashes. The cracking of a hash is shown in Figure 4-12.

Cloud Infrastructure Assessment Tools

As moving assets to the cloud becomes the rule and not the exception, identifying and mitigating vulnerabilities in cloud environments steadily increase in importance. There are several monitoring and attack tools with which you should be familiar.
ScoutSuite

ScoutSuite is a data collection tool that allows you to use what are called longitudinal survey panels to track and monitor the cloud environment. It is open source and utilizes APIs made available by the cloud provider. The following cloud providers are currently supported/planned:

- Amazon Web Services (AWS)
- Microsoft Azure
- Google Cloud Platform
- Alibaba Cloud (alpha)
- Oracle Cloud Infrastructure (alpha)

Prowler

AWS Security Best Practices Assessment, Auditing, Hardening and Forensics Readiness Tool, also called Prowler, allows you to run reports of various types. These reports list gaps found between your practices and best practices of AWS as stated in CIS Amazon Web Services Foundations Benchmark 1.1.

Figure 4-13 shows partial sample report results. Notice that the results are color coded to categorize any gaps found.

Pacu

Exploit frameworks are packages of tools that provide a bed for creating and launching attacks of various types. One of the more famous of these is Metasploit. Pacu is an exploit framework used to assess and attack AWS cloud environments. Using plug-in modules, it assists an attacker in
- Enumeration
- Privilege escalation
- Data exfiltration
- Service exploitation
- Log manipulation

### Exam Preparation Tasks

As mentioned in the section “How to Use This Book” in the Introduction, you have several choices for exam preparation: the exercises here, Chapter 22, “Final Preparation,” and the exam simulation questions in the Pearson Test Prep Software Online.

### Review All Key Topics

Review the most important topics in this chapter, noted with the Key Topics icon in the outer margin of the page. Table 4-2 lists a reference of these key topics and the page numbers on which each is found.

<table>
<thead>
<tr>
<th>Key Topic Element</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulleted list</td>
<td>Software development life cycle (SDLC)</td>
<td>72</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Approaches to static code review</td>
<td>73</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Code review types</td>
<td>73</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Approaches to dynamic testing</td>
<td>74</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Fuzz testing types</td>
<td>75</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>TCP flags</td>
<td>76</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Nmap scan types</td>
<td>77</td>
</tr>
<tr>
<td>Example 4-1</td>
<td>FIN scan using nmap</td>
<td>78</td>
</tr>
<tr>
<td>Figure 4-8</td>
<td>Host scan with nmap</td>
<td>80</td>
</tr>
<tr>
<td>Bulleted list</td>
<td>Operations possible with hping</td>
<td>80</td>
</tr>
<tr>
<td>Figure 4-9</td>
<td>SYN flood</td>
<td>81</td>
</tr>
<tr>
<td>Figure 4-10</td>
<td>Capturing authentication hashes with Responder</td>
<td>82</td>
</tr>
</tbody>
</table>
Define Key Terms

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

- web vulnerability scanners, synthetic transaction monitoring, real user monitoring (RUM), Burp Suite, OWASP Zed Attack Proxy (ZAP), Nikto, Arachni, Nessus Professional, OpenVAS, Qualys, software development life cycle (SDLC), static code analysis, dynamic analysis, reverse engineering, fuzzing, enumeration, Nmap, Null scan, FIN scan, XMAS scan, host scanning, SYN flood, active enumeration, passive enumeration, Responder, Aircrack-ng, Reaver, oclHashcat, ScoutSuite, Prowler, Pacu

Review Questions

1. The _________________________________ produces an interception proxy called ZAP.

2. Match the tool on the left with its definition on the right.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burp</td>
<td>An interception proxy produced by OWASP</td>
</tr>
<tr>
<td>Nikto</td>
<td>A Ruby framework for assessing the security of a web application</td>
</tr>
<tr>
<td>ZAP</td>
<td>Vulnerability scanner that is dedicated to web servers</td>
</tr>
<tr>
<td>Arachni</td>
<td>Can scan an application for vulnerabilities and can also be used to crawl an application (to discover content)</td>
</tr>
</tbody>
</table>

3. List at least one of the advantages of the cloud-based approach to vulnerability scanning.

4. Arrange the following steps of the SDLC in the proper order.
   - Gather requirements
   - Certify/accredit
   - Release/maintain
   - Design
Test/validate
Perform change management and configuration management/replacement
Develop
Plan/initiate project

5. ________________ analysis is done without the code executing.

6. List at least one form of static code review.

7. Match the type of code review on the left with its definition on the right.

<table>
<thead>
<tr>
<th>Review Types</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reverse engineering</td>
<td>Injecting invalid or unexpected input</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>Analyzing a subject system to identify the system’s components and their interrelationships</td>
</tr>
<tr>
<td>Real user monitoring</td>
<td>Running scripted transactions against an application</td>
</tr>
<tr>
<td>Synthetic transaction</td>
<td>Monitoring method that captures and analyzes every transaction</td>
</tr>
<tr>
<td>monitoring</td>
<td></td>
</tr>
</tbody>
</table>

8. List at least one measure that can help prevent fault injection attacks.

9. Match the following tools with their definitions.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmap</td>
<td>Used to attack Wi-Fi Protected Setup (WPS)</td>
</tr>
<tr>
<td>hping</td>
<td>Tool that can be used for answering NBT and LLNR name requests</td>
</tr>
<tr>
<td>Responder</td>
<td>Command-line-oriented TCP/IP packet assembler/analyzer</td>
</tr>
<tr>
<td>Reaver</td>
<td>When used for scanning, it typically locates the devices, locates the open ports on the devices, and determines the OS on each host</td>
</tr>
</tbody>
</table>

10. List at least one of the cloud platforms supported by ScoutSuite.