Chapter 7

Protecting Against Advanced Attacks

CompTIA Security+ objectives covered in this chapter:

1.2 Compare and contrast types of attacks.

- Application/service attacks (DoS, DDoS, Man-in-the-middle, Buffer overflow, Injection, Cross-site scripting, Cross-site request forgery, Privilege escalation, ARP poisoning, Amplification, DNS poisoning, Domain hijacking, Man-in-the-browser, Zero day, Replay, Pass the hash, Hijacking and related attacks [Clickjacking, Session hijacking, URL hijacking, Typo squatting], Driver manipulation [Shimming, Refactoring], MAC spoofing, IP spoofing), Cryptographic attacks (Birthday, Known plain text/cipher text, Rainbow tables, Dictionary, Brute force [Online vs. offline], Collision, Replay)

1.6 Explain the impact associated with types of vulnerabilities.

- Race conditions, Improper input handling, Improper error handling, Memory/buffer vulnerability (Memory leak, Integer overflow, Buffer overflow, Pointer dereference, DLL injection), New threats/zero day

3.1 Explain use cases and purpose for frameworks, best practices and secure configuration guides.

- Industry-standard frameworks and reference architectures (Regulatory, Non-regulatory, National vs. international, Industry-specific frameworks), Benchmarks/secure configuration guides (Platform/vendor-specific guides [Web server, Operating system, Application server, Network infrastructure devices], General purpose guides)

3.6 Summarize secure application development and deployment concepts.

- Development life-cycle models (Waterfall vs. Agile), Secure

DevOps (Security automation, Continuous integration, Baselining, Immutable systems, Infrastructure as code), Version control and change management, Provisioning and deprovisioning, Secure coding techniques (Proper error handling, Proper input validation, Normalization, Stored procedures, Code signing, Encryption, Obfuscation/camouflage, Code reuse/dead code, Server-side vs. client-side execution and validation, Memory management, Use of third- party libraries and SDKs, Data exposure), Code quality and testing (Static code analyzers, Dynamic analysis (e.g., fuzzing), Stress testing, Sandboxing, Model verification), Compiled vs. runtime code

6.1 Compare and contrast basic concepts of cryptography.

- Collision

**

If there's one thing that's abundant in the IT world, it is attacks and attackers. Attackers lurk almost everywhere. If you have computer systems, you can't escape them. However, you can be proactive in identifying the different types of attacks and take steps to prevent them, or at least prevent their effectiveness. This chapter covers a wide assortment of attacks from different sources and provides some insight into preventing many of them.

Comparing Common Attacks

This section summarizes many common and advanced types of attacks launched against systems and networks. It's important to realize that effective countermeasures exist for all of the attacks listed in this book. However, attackers are actively working on beating the countermeasures. As they do, security professionals create additional countermeasures and the attackers try to beat them. The battle continues daily.

The goal in this section is to become aware of many of the well-known attacks. By understanding these, you'll be better prepared to comprehend the improved attacks as they emerge and the improved countermeasures.

DoS Versus DDoS

A denial-of-service (***DoS***) attack is an attack from one attacker against one target. A distributed denial-of-servic*e* (***DDoS***) attack is an attack from two or more computers against a single target. DDoS attacks often include

sustained, abnormally high network traffic on the network interface card of the attacked computer. Other system resource usage (such as the processor and memory usage) will also be abnormally high. The goal of both is to perform a service attack and prevent legitimate users from accessing services on the target computer.

Remember this

A denial-of-service (DoS) attack is an attack from a single source that attempts to disrupt the services provided by another system. A distributed denial-of-service (DDoS) attack includes multiple computers attacking a single target. DDoS attacks typically include sustained, abnormally high network traffic.

Privilege Escalation

Chapter 6, "Comparing Threats, Vulnerabilities, and Common Attacks," discusses ***privilege escalation*** tactics that attackers often use. For example, attackers often use remote access Trojans (RATs) to gain access to a single system. They typically have limited privileges (a combination of rights and permissions) when they first exploit a system. However, they use various privilege escalation techniques to gain more and more privileges.

Most of the attacks in this chapter also use privilege escalation techniques for the same reason—to gain more and more access to a system or a network.

Spoofing

Spoofing occurs when one person or entity impersonates or masquerades as someone or something else. Two common spoofing attacks mentioned specifically in the CompTIA objectives are media access control (MAC) address spoofing and Internet Protocol (IP) address spoofing.

Host systems on a network have a media access control (MAC) address assigned to the network interface card (NIC). These are hard-coded into the NIC. However, it's possible to use software methods to associate a different MAC address to the NIC in a ***MAC spoofing*** attack. For example, Chapter 3, "Exploring Network Technologies and Tools," discusses a MAC flood attack where an attacker overwhelms a switch with spoofed MAC addresses. Flood

guards prevent these types of attacks. Chapter 4, "Securing Your Network," discusses how wireless attackers can bypass MAC address filtering by spoofing the MAC address of authorized systems.

In an ***IP spoofing*** attack, the attacker changes the source address so that it looks like the IP packet originated from a different source. This can allow an attacker to launch an attack from a single system, while it appears that the attack is coming from different IP addresses.

Remember this

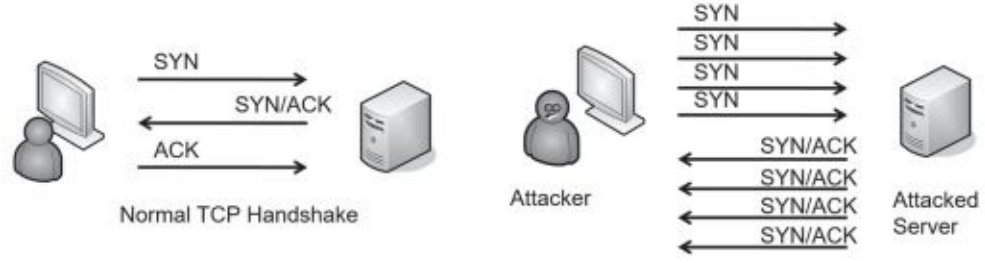
Spoofing attacks typically change data to impersonate another system or person. MAC spoofing attacks change the source MAC address and IP spoofing attacks change the source IP address.

SYN Flood Attacks

The SYN flood attack is a common attack used against servers on the Internet. They are easy for attackers to launch, difficult to stop, and can cause significant problems. The SYN flood attack disrupts the TCP handshake process and can prevent legitimate clients from connecting.

Chapter 3 explains how TCP sessions use a three-way handshake when establishing a session. As a reminder, two systems normally start a TCP session by exchanging three packets in a TCP handshake. For example, when a client establishes a session with a server, it takes the following steps:

1. The client sends a SYN (synchronize) packet to the server.
2. The server responds with a SYN/ACK (synchronize/acknowledge) packet.
3. The client completes the handshake by sending an ACK (acknowledge) packet. After establishing the session, the two systems exchange data.



However, in a SYN flood attack, the attacker never completes the handshake by sending the ACK packet. Additionally, the attacker sends a barrage of

SYN packets, leaving the server with multiple half-open connections. Figure 7.1 compares a normal TCP handshake with the start of a SYN flood attack.

Figure 7.1: TCP handshake and SYN flood attack

In some cases, these half-open connections can consume a server's resources while it is waiting for the third packet, and it can actually crash. More often, though, the server limits the number of these half-open connections. Once the limit is reached, the server won't accept any new connections, blocking connections from legitimate users. For example, Linux systems support an iptables command that can set a threshold for SYN packets, blocking them after the threshold is set. Although this prevents the SYN flood attack from crashing the system, it also denies service to legitimate clients.

Attackers can launch SYN flood attacks from a single system in a DoS attack. They will often spoof the source IP address when doing so. Attackers can also coordinate an attack from multiple systems using a DDoS attack.

Man-in-the-Middle Attacks

A ***man-in-the-middle(MITM)*** attack is a form of active interception or active eavesdropping. It uses a separate computer that accepts traffic from each party in a conversation and forwards the traffic between the two. The two computers are unaware of the MITM computer, and it can interrupt the traffic at will or insert malicious code.

For example, imagine that Maggie and Bart are exchanging information with their two computers over a network. If Hacker Harry can launch an MITM attack from a third computer, he will be able to intercept all traffic. Maggie and Bart still receive all the information, so they are unaware of the attack. However, Hacker Harry also receives all the information. Because the

MITM computer can control the entire conversation, it is easy to insert malicious code and send it to the computers. Address Resolution Protocol (ARP) poisoning is one way that an attacker can launch an MITM attack.

Kerberos helps prevent man-in-the-middle attacks with mutual authentication. It doesn't allow a malicious system to insert itself in the middle of the conversation without the knowledge of the other two systems.

ARP Poisoning Attacks

ARP poisoning is an attack that misleads computers or switches about the actual MAC address of a system. The MAC address is the physical address, or hardware address, assigned to the NIC. ARP resolves the IP addresses of systems to their hardware address and stores the result in an area of memory known as the ARP cache.

TCP/IP uses the IP address to get a packet to a destination network. Once the packet arrives on the destination network, it uses the MAC address to get it to the correct host. ARP uses two primary messages:

- **ARP request.** The ARP request broadcasts the IP address and essentially asks, "Who has this IP address?"
- **ARP reply.** The computer with the IP address in the ARP request responds with its MAC address. The computer that sent the ARP request caches the MAC address for the IP. In many operating systems, all computers that hear the ARP reply also cache the MAC address.

A vulnerability with ARP is that it is very trusting. It will believe any ARP reply packet. Attackers can easily create ARP reply packets with spoofed or bogus MAC addresses, and poison the ARP cache on systems in the network. Two possible attacks from ARP poisoning are a man- in-the-middle attack and a DoS attack.

ARP Man-in-the-Middle Attacks

In a man-in-the-middle attack, an attacker can redirect network traffic and, in some cases, insert malicious code. Consider Figure 7.2. Normally, traffic from the user to the Internet will go through the switch directly to the router, as shown in the top of Figure 7.2. However, after poisoning the ARP cache of the victim, traffic is redirected to the attacker.

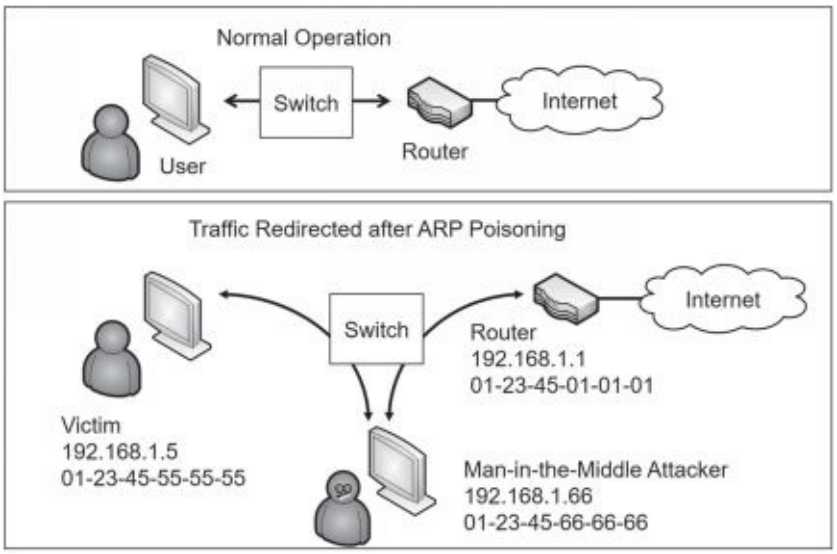


Figure 7.2: ARP poisoning used to redirect traffic

The victim's ARP cache should include this entry to send data to the router:

192.168.1.1, 01-23-45-01-01-01

However, after poisoning the ARP cache, it includes this entry:

192.168.1.1, 01-23-45-66-66-66

The victim now sends all traffic destined for the router to the attacker. The attacker captures the data for analysis later. It also uses another method such as IP forwarding to send the traffic to the router so that the victim is unaware of the attack.

Remember this

ARP poisoning attacks attempt to mislead systems about the actual MAC address of a system. ARP poisoning is sometimes used in man-in-the-middle attacks.

ARP DoS Attacks

An attacker can also use ARP poisoning in a DoS attack. For example, an attacker can send an ARP reply with a bogus MAC address for the default gateway. The default gateway is the IP address of a router connection that provides a path out of the network. If all the computers cache a bogus MAC address for the default gateway, none of them can reach it, and it stops all traffic out of the network.

DNS Attacks

Chapter 3 covers Domain Name System (DNS) in much more depth, but as a reminder, DNS resolves host names to IP addresses. This eliminates the need for you and me to have to remember the IP address for web sites. Instead, we simply type the name into the browser, and it connects. For example, if you type in *gcgapremium.com* as the Uniform Resource Locator (URL) in your web browser, your system queries a DNS server for the IP address. DNS responds with the correct IP address and your system connects to the web site using the IP address.

DNS also provides reverse lookups. In a reverse lookup, a client sends an IP address to a DNS server with a request to resolve it to a name. Some applications use this as a rudimentary security mechanism to detect spoofing. For example, an attacker may try to spoof the computer's identity by using a different name during a session. However, the Transmission Control Protocol/ Internet Protocol (TCP/IP) packets in the session include the IP address of the masquerading system and a reverse lookup shows the system's actual name. If the names are different, it shows suspicious activity. Reverse lookups are not 100 percent reliable because reverse lookup records are optional on DNS servers. However, they are useful when they're available.

Three attacks against DNS services are DNS poisoning, pharming, and DDoS.

DNS Poisoning Attacks

A ***DNS poisoning*** attack attempts to modify or corrupt DNS results. For example, a successful DNS poisoning attack can modify the IP address associated with *google.com* and replace it with the IP address of a malicious web site. Each time a user queries DNS for the IP address of *google.com*, the DNS server responds with the IP address of the malicious web site.

There have been several successful DNS poisoning attacks over the years. Many current DNS servers use Domain Name System Security Extensions (***DNSSEC***) to protect the DNS records and prevent DNS poisoning attacks. Chapter 3 covers DNSSEC in more depth.

Pharming Attacks

A pharming attack is another type of attack that manipulates the DNS

name resolution process. It either tries to corrupt the DNS server or the DNS client. Just as a DNS poisoning attack can redirect users to different web sites, a successful pharming attack redirects a user to a different web site.

Pharming attacks on the client computer modify the hosts file used on Windows systems. This file is in the *C:\Windows\System32\drivers\etc* folder and can include IP addresses along with host name mappings. By default, it doesn't have anything other than comments on current Windows computers. However, a mapping might look like this:

```
127.0.0.1
localhost
13.207.21.200     google.com
```

The first entry maps the name localhost to the loopback IP address of 127.0.0.1. The second entry maps the name google.com to the IP address of bing.com (13.207.21.200). If a user enters google.com into the address bar of a browser, the browser will instead go to bing.com. Practical jokers might do this to a friend's computer and it isn't malicious. However, if the IP address points to a malicious server, this might cause the system to download malware.

DDoS DNS Attacks

It's difficult to take down the Internet. However, a cyberattack in October 2016 effectively did so for millions of users in North America and Europe. Specifically, on October 21, attackers launched three DDoS attacks during the day at 7:00 a.m., at 11:52 a.m., and at 4:00 p.m. These attacks prevented users from accessing a multitude of sites, such as Amazon, CNN, Fox News, Netflix, PayPal, Reddit, Spotify, Twitter, Xbox Live, and more.

Attackers infected many Internet-connected devices, such as video cameras, video recorders, printers, and baby monitors, with malware called Mirai. Mirai forces individual systems to become bots within large botnets. Chapter 6 covers bots and botnets in more depth. On October 21, they sent commands to millions of infected devices directing them to repeatedly send queries to DNS servers. These queries overwhelmed the DNS servers and prevented regular users from accessing dozens of web sites.

These three attacks were launched against DNS servers maintained by Dyn, Inc., an Internet performance management company. They clearly demonstrated that it is possible to seriously disrupt DNS services, causing Internet access problems for millions of people.

Amplification Attacks

An ***amplification attack*** is a type of DDoS attack. It typically uses a method that significantly increases the amount of traffic sent to, or requested from, a victim. As an example, a smurf attack spoofs the source address of a directed broadcast ping packet to flood a victim with ping replies. It's worthwhile to break this down:

- **A ping is normally unicast—one computer to one computer.** A ping sends ICMP echo requests to one computer, and the receiving computer responds with ICMP echo responses.
- **The smurf attack sends the ping out as a broadcast.** In a broadcast, one computer sends the packet to all other computers in the subnet.
- **The smurf attack spoofs the source IP.** If the source IP address isn't changed, the computer sending out the broadcast ping will get flooded with the ICMP replies. Instead, the smurf attack substitutes the source IP with the IP address of the victim, and the victim gets flooded with these ICMP replies.

DNS amplification attacks send DNS requests to DNS servers spoofing the IP address of the victim. Instead of just asking for a single record, these attacks tell the DNS servers to send as much zone data as possible, amplifying the data sent to the victim. Repeating this process from multiple attackers can overload the victim system.

An example of a Network Time Protocol (NTP) amplification attack uses the monlist command. When used normally, it sends a list of the last 600 hosts that connected to the NTP server. In an NTP amplification attack with monlist, the attacker spoofs the source IP address when sending the command. The NTP server then floods the victim with details of the last 600 systems that requested the time from the NTP server.

Remember this

DNS poisoning attacks attempt to corrupt DNS data. Amplification attacks increase the amount of traffic sent to or requested from a victim and can be used against a wide variety of systems, including individual hosts, DNS servers, and NTP servers.

Password Attacks

Password attacks attempt to discover or bypass passwords used for authentication on systems and networks, and for different types of files. Some password attacks are sophisticated cryptographic attacks, while others are rather simple brute force attacks. The following sections cover some common password attacks.

Brute Force Attacks

A ***brute force*** attack attempts to guess all possible character combinations. The two types of brute force attacks are online and offline.

An online password attack attempts to discover a password from an online system. For example, an attacker can try to log on to an account by repeatedly guessing the username and password. Many tools are available that attackers can use to automate the process. For example, ncrack is a free tool that can be used to run online brute force password attacks.

Chapter 2, "Understanding Identity and Access Management," discusses account lockout policies used in Windows systems. They are effective against online brute force password attacks. An account lockout setting locks an account after the user enters the incorrect password a preset number of times. Individual services often have their own settings to prevent brute force attacks. For example, Secure Shell (SSH) can disconnect an attacker if he hasn't logged on within 60 seconds and limit the number of authentication attempts per connection. These settings often thwart brute force attacks against these services.

Offline password attacks attempt to discover passwords from a captured database or captured packet scan. For example, when attackers hack into a system or network causing a data breach, they can download entire databases. They then perform offline attacks to discover the passwords contained within the databases.

One of the first steps to thwart offline brute force attacks is to use complex passwords and to store the passwords in an encrypted or hashed format. Complex passwords include a mix of uppercase letters, lowercase letters, numbers, and special characters. Additionally, longer passwords are much more difficult to crack than shorter passwords.

Dictionary Attacks

A ***dictionary*** attack is one of the original password attacks. It uses a dictionary of words and attempts every word in the dictionary to see if it works. A dictionary in this context is simply a list of words and character combinations.

Dictionaries used in these attacks have evolved over time to reflect user behavior. Today, they include many of the common passwords that uneducated users configure for their accounts. For example, even though 12345 isn't a dictionary word, many people use it as a password, so character sets such as these have been added to many dictionaries used by dictionary attack tools.

These attacks are thwarted by using complex passwords. A complex password will not include words in a dictionary.

Remember this

Brute force attacks attempt to guess passwords. Online attacks guess the password of an online system. Offline attacks guess the password stored within a file, such as a database. Dictionary attacks use a file of words and common passwords to guess a password. Account lockout policies help protect against brute force attacks and complex passwords thwart dictionary attacks.

Password Hashes

Most systems don't store the actual password for an account. Instead, they store a hash of the password. Hash attacks attack the hash of a password instead of the password. Chapter 1, "Mastering Security Basics," introduces hashing and Chapter 10, "Understanding Cryptography and PKI," discusses hashing in much more depth. A ***hash*** is simply a number created with a hashing algorithm such as Message Digest 5 (***MD5***) or Secure Hash Algorithm 3 (***SHA-3***). A system can use a hashing algorithm such as MD5 to create a hash of a password.

As an example, if a user's password is IC@nP@$$S3curity+, the system calculates the hash and stores it instead. In this example, the MD5 hash is 75c8ac11c86ca966b58166187589cc15. Later, a user authenticates with a username and password. The system then calculates the hash of the

password that the user entered, and compares the calculated hash against the stored hash. If they match, it indicates the user entered the correct password.

Unfortunately, tools are available to discover many hashed passwords. For example, MD5 Online (*http://www.md5online.org/*) allows you to enter a hash and it gives you the text of the password. If the password is 12345, the hash is 827ccb0eea8a706c4c34a16891f84e7b. If you enter that hash into MD5 Online, it returns the password of 12345 in less than a second. MD5 Online uses a database of hashed words from a dictionary. If the hash matches a database entry, the site returns the password.

The password is rarely sent across the network in cleartext. Chapter 8, "Using Risk Management Tools," discusses protocol analyzers and shows how an attacker can capture and view a password if it is sent across a network in cleartext. To prevent this, a protocol can calculate the hash of the password on the user's system and then send the hash across the network instead of the password. Unfortunately, if the hash is passed across the network in an unencrypted format, the attacker may be able to capture the hash and use it to log on to a system. Instead, most authentication protocols encrypt the password or the hash before sending it across the network.

Pass the Hash Attacks

In a ***pass the hash*** attack, the attacker discovers the hash of the user's password and then uses it to log on to the system as the user. Any authentication protocol that passes the hash over the network in an unencrypted format is susceptible to this attack. However, it is most associated with Microsoft LAN Manager (LM) and NT LAN Manager (NTLM), two older security protocols used to authenticate Microsoft clients. They are both susceptible to pass the hash attacks.

Any system using LM or NTLM is susceptible to a pass the hash attack. The simple solution (and the recommended solution) is to use NTLMv2 or Kerberos instead. NTLMv2 uses a number used once (nonce) on both the client and the authenticating server. The authentication process uses both the client nonce and the server nonce in a challenge/response process. Chapter 2 discusses Kerberos.

Unfortunately, many existing applications still use NTLM, so it can still be enabled on many Windows systems for backward compatibility. However, Microsoft recommends configuring clients to only send NTLMv2 responses and configuring authenticating servers to refuse any use of LM or NTLM.

This is relatively easy to do via a Group Policy setting.

Birthday Attacks

A ***birthday*** attack is named after the birthday paradox in mathematical probability theory. The birthday paradox states that for any random group of 23 people, there is a 50 percent chance that 2 of them have the same birthday. This is not the same year, but instead one of the 365 days in any year.

In a birthday attack, an attacker is able to create a password that produces the same hash as the user's actual password. This is also known as a hash collision.

A hash ***collision*** occurs when the hashing algorithm creates the same hash from different passwords. This is not desirable. As an example, imagine a simple hashing algorithm creates three-digit hashes. The password "success" might create a hash of 123 and the password "passed" might create the same hash of 123. In this scenario, an attacker could use either "success" or "passed" as the password and both would work.

Birthday attacks on hashes are thwarted by increasing the number of bits used in the hash to increase the number of possible hashes. For example, the MD5 algorithm uses 128 bits and is susceptible to birthday attacks. SHA-3 can use as many as 512 bits and it is not susceptible to birthday attacks.

Rainbow Table Attacks

Rainbow table attacks are a type of attack that attempts to discover the password from the hash. A rainbow table is a huge database of precomputed hashes. It helps to look at the process of how some password cracker applications discover passwords without a rainbow table. Assume that an attacker has the hash of a password. The application can use the following steps to crack it:

1. The application guesses a password (or uses a password from a dictionary).
2. The application hashes the guessed password.
3. The application compares the original password hash with the guessed password hash. If they are the same, the application now knows the password.
4. If they aren't the same, the application repeats steps 1 through 3 until finding a match. From a computing perspective, the most time-consuming part

of these steps is hashing the guessed password in step 2. However, by using rainbow tables, applications eliminate this step. Rainbow tables are huge databases of passwords and their calculated hashes. Some rainbow tables are as large as 160 GB in size, and they include hashes for every possible combination of characters up to eight characters in length. Larger rainbow tables are also available using more characters.

In a rainbow table attack, the application simply compares the hash of the original password against hashes stored in the rainbow table. When the application finds a match, it identifies the password used to create the hash (or at least text that can reproduce the hash of the original password). Admittedly, this is a simplistic explanation of a rainbow table attack, but it is adequate unless you plan on writing an algorithm to create your own rainbow table attack software.

Salting passwords is a common method of preventing rainbow table attacks, along with other password attacks such as dictionary attacks. A ***salt*** is a set of random data such as two additional characters. Password salting adds these additional characters to a password before hashing it. These additional characters add complexity to the password, and also result in a different hash than the system would create using only the original password. This causes password attacks that compare hashes to fail.

Chapter 10 covers bcrypt and Password-Based Key Derivation Function 2 (PBKDF2). Both use salting techniques to increase the complexity of passwords and thwart brute force and rainbow table attacks.

Remember this

Passwords are typically stored as hashes. A pass the hash attack attempts to use an intercepted hash to access an account. Salting adds random text to passwords before hashing them and thwarts many password attacks, including rainbow table attacks. A hash collision occurs when the hashing algorithm creates the same hash from different passwords. Birthday attacks exploit collisions in hashing algorithms.

Replay Attacks

A ***replay attack*** is one where an attacker replays data that was already part of a communication session. In a replay attack, a third party attempts to impersonate a client that is involved in the original session. Replay attacks can occur on both wired and wireless networks.

As an example, Maggie and Bart may initiate a session with each other. During the communication, each client authenticates with the other by passing authentication credentials to the other system. Hacker Harry intercepts all the data, including the credentials, and later initiates a conversation with Maggie pretending to be Bart. When Maggie challenges Hacker Harry, he sends Bart's credentials.

Many protocols use timestamps and sequence numbers to thwart replay attacks. For example, Kerberos, covered in Chapter 2, helps prevent replay attacks with timestamped tickets.

Remember this

Replay attacks capture data in a session with the intent of later impersonating one of the parties in the session. Timestamps and sequence numbers are effective counter measures against replay attacks.

Known Plaintext Attacks

Many cryptographic attacks attempt to decrypt encrypted data. Chapter 10 covers encryption and decryption in more depth, but two relevant terms in this section are plaintext and ciphertext. Plaintext is human-readable data. An encryption algorithm scrambles the data, creating ciphertext.

An attacker can launch a ***known plaintext*** attack if he has samples of both the plaintext and the ciphertext. As an example, if an attacker captures an encrypted message (the ciphertext) and knows the plaintext of the message, he can use both sets of data to discover the encryption and decryption method. If successful, he can use the same decryption method on other ciphertext.

A chosen plaintext attack is similar, but the attacker doesn't have access to all the plaintext. As an example, imagine a company includes the following sentences at the end of every email:

"The information contained in this email and any accompanying attachments may contain proprietary information about the Pay & Park

& Pay parking garage. If you are not the intended recipient of this information, any use of this information is prohibited."

If the entire message is encrypted, the attacker can try various methods to decrypt the chosen plaintext (the last two sentences included in every email). When he's successful, he can use the same method to decrypt the entire message.

In a ciphertext only attack, the attacker doesn't have any information on the plaintext. Known plaintext and chosen plaintext attacks are almost always successful if an attacker has the resources and time. However, ciphertext only attacks are typically only successful on weak encryption algorithms. They can be thwarted by not using legacy and deprecated encryption algorithms.

Hijacking and Related Attacks

Typo squatting (also called ***URL hijacking***) occurs when someone buys a domain name that is close to a legitimate domain name. People often do so for malicious purposes. As an example, CompTIA hosts the comptia.org web site. If an attacker purchases the name compt**ai**.org with a slight misspelling at the end of comptia, some users might inadvertently go to the attacker's web site instead of the legitimate web site.

Attackers might buy a similar domain for a variety of reasons, including:

- **Hosting a malicious web site.** The malicious web site might try to install drive-by malware on users' systems when they visit.
- **Earning ad revenue.** The attacker can host pay-per-click ads. When visitors go to the site and click on the ads, advertisers pay revenue to the attacker.
- **Reselling the domain.** Attackers can buy domain names relatively cheaply, but resell them to the owner of the original site for a hefty profit.

Clickjacking tricks users into clicking something other than what they think they're clicking. As a simple example, imagine Bart is browsing Facebook. He sees a comment labeled Chalkboard Sayings, so he clicks it. He's taken to a page with a heading of "Human Test" and directions to "Find the blue button to continue." This looks like a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA), but it isn't. When he clicks the blue button, he is actually clicking on a Facebook share

button, causing him to share the original comment labeled Chalkboard Sayings with his Facebook friends.

While it's rarely apparent to the user, most clickjacking attacks use Hypertext Markup Language (HTML) frames. A frame allows one web page to display another web page within an area defined as a frame or iframe.

Attackers continue to find new ways to launch clickjacking attacks. As they appear, web developers implement new standards to defeat them. Most methods focus on breaking or disabling frames. This ensures that attackers cannot display your web page within a frame on their web page. For example, the Facebook share example is thwarted by Facebook web developers adding code to their web pages preventing the use of frames.

Session hijacking takes advantage of session IDs stored in cookies. When a user logs on to a web site, the web site often returns a small text file (called a cookie) with a session ID. In many cases, this cookie is stored on the user's system and remains active until the user logs off. If the user closes the session and returns to the web site, the web site reads the cookie and automatically logs the user on. This is convenient for the user, but can be exploited by an attacker.

In a session hijacking attack, the attacker utilizes the user's session ID to impersonate the user. The web server doesn't know the difference between the original user and the attacker because it is only identifying the user based on the session ID.

Attackers can read cookies installed on systems through several methods, such as through cross-site scripting attacks (described later in this chapter). Once they have the session ID, they can insert it into the HTTP header and send it to the web site. If the web server uses this session ID to log the user on automatically, it gives the attacker access to the user's account.

Domain Hijacking

In a ***domain hijacking*** attack, an attacker changes the registration of a domain name without permission from the owner. Attackers often do so with social engineering techniques to gain unauthorized access to the domain owner's email account.

As an example, imagine that Homer sets up a domain named homersimpson.com. He uses his Gmail account as the email address when he

registers it, though he rarely checks his Gmail account anymore.

Attackers watch his Facebook page and notice that he often adds simple comments like “Doh!” Later, they try to log on to his Gmail account with a brute force attempt. They try the password of Doh!Doh! and get in. They then go to the domain name registrar, and use the Forgot Password feature. It sends a link to Homer’s Gmail account to reset the password. After resetting the password at the domain name registrar site, the attackers change the domain ownership. They also delete all the emails tracking what they did. Later, Homer notices his web site is completely changed and he no longer has access to it.

Remember this

Attackers purchase similar domain names in typo squatting (also called URL hijacking) attacks. Users visit the typo squatting domain when they enter the URL incorrectly with a common typo. In a session hijacking attack, the attacker utilizes the user’s session ID to impersonate the user. In a domain hijacking attack, an attacker changes the registration of a domain name without permission from the owner.

Man-in-the-Browser

A ***man-in-the-browser*** is a type of proxy Trojan horse that infects vulnerable web browsers. Successful man-in-the-browser attacks can capture browser session data. This includes keyloggers to capture keystrokes, along with all data sent to and from the web browser.

As an example, Zeus is a Trojan horse that has used man-in-the-browser techniques after infecting systems. Zeus includes keystroke logging and form grabbing. Once the attackers collect logon information for a user’s bank, they use it to log on and transfer money to offshore accounts.

Driver Manipulation

Operating systems use drivers to interact with hardware devices or software components. For example, when you print a page using Microsoft Word, Word accesses the appropriate print driver via the Windows operating

system. Similarly, if you access encrypted data on your system, the operating system typically accesses a software driver to decrypt the data so that you can view it.

Occasionally, an application needs to support an older driver. For example, Windows 10 needed to be compatible with drivers used in Windows 8, but all the drivers weren't compatible at first. ***Shimming*** provides the solution that makes it appear that the older drivers are compatible.

A driver shim is additional code that can be run instead of the original driver. When an application attempts to call an older driver, the operating system intercepts the call and redirects it to run the shim code instead. ***Refactoring*** code is the process of rewriting the internal processing of the code, without changing its external behavior. It is usually done to correct problems related to software design.

Developers have a choice when a driver is no longer compatible. They can write a shim to provide compatibility or they can completely rewrite the driver to refactor the relevant code. If the code is clunky, it's appropriate to rewrite the driver.

Attackers with strong programming skills can use their knowledge to manipulate drivers by either creating shims, or by rewriting the internal code. If the attackers can fool the operating system into using a manipulated driver, they can cause it to run malicious code contained within the manipulated driver.

Zero-Day Attacks

A ***zero-day vulnerability*** is a weakness or bug that is unknown to trusted sources, such as operating system and antivirus vendors. A zero-day attack exploits an undocumented vulnerability. Many times, the vendor isn't aware of the issue. At some point, the vendor learns of the vulnerability and begins to write and test a patch to eliminate it. However, until the vendor releases the patch, the vulnerability is still a zero-day vulnerability.

In most cases, a zero-day vulnerability is a new threat. However, there have been zero-day vulnerabilities that have existed for years.

As an example, a bug existed in the virtual DOS machine (VDM) that shipped with every version of 32-bit Windows systems from 1993 to 2010. The bug allowed attackers to escalate their privileges to full system level, effectively allowing them to take over the system. Google researcher Tavis

Ormandy stated that he reported the bug to Microsoft in mid-2009. At this point, Microsoft (the vendor) knew about the bug, but didn't release a work-around until January 2010 and a patch until February 2010. Because the bug wasn't known publicly until January 2010, it remained a zero-day vulnerability until then.

Both attackers and security experts are constantly looking for new threats, such as zero-day vulnerabilities. Attackers want to learn about them so that they can exploit them. Most security experts want to know about them so that they can help ensure that vendors patch them before causing damage to users.

Remember this

Zero-day exploits are undocumented and unknown to the public. The vendor might know about it, but has not yet released a patch to address it.

Memory Buffer Vulnerabilities

Many application attacks take advantage of vulnerabilities in a system's memory or buffers. Because of this, it's important for developers to use secure memory management techniques within their code. For example, poor memory management techniques can result in a memory leak, or allow various overflow issues. The following sections describe some common memory issues related to applications.

Memory Leak

A ***memory leak*** is a bug in a computer application that causes the application to consume more and more memory the longer it runs. In extreme cases, the application can consume so much memory that the operating system crashes.

Memory leaks are typically caused by an application that reserves memory for short-term use, but never releases it. As an example, imagine a web application collects user profile data to personalize the browsing experience for users. However, it collects this data every time a user accesses a web page and it never releases the memory used to store the data. Over time, the web server will run slower and slower and eventually need to be rebooted.

Integer Overflow

An ***integer overflow*** attack attempts to use or create a numeric value that is too big for an application to handle. The result is that the application gives inaccurate results. As an example, if an application reserves 8 bits to store a number, it can store any value between 0 and 255. If the application attempts to multiply two values such as 95 × 59, the result is 5,605. This number cannot be stored in the 8 bits, so it causes an integer overflow error. It's a good practice to double- check the size of buffers to ensure they can handle any data generated by the applications.

In some situations, an integer overflow error occurs if an application expects a positive number, but receives a negative number instead. If the application doesn't have adequate error- and exception-handling routines, this might cause a buffer overflow error. Input handling and error handling are discussed later in this chapter.

Buffer Overflows and Buffer Overflow Attacks

A ***buffer overflow*** occurs when an application receives more input, or different input, than it expects. The result is an error that exposes system memory that would otherwise be protected and inaccessible. Normally, an application will have access only to a specific area of memory, called a buffer. The buffer overflow allows access to memory locations beyond the application's buffer, enabling an attacker to write malicious code into this area of memory.

As an example, an application may be expecting to receive a string of 15 characters for a username. If it receives more than 15 characters, it can cause a buffer overflow and expose system memory. The following HTTP GET command shows an example of sending a long string to the system to create a buffer overflow:

```
GET  /index.php?
username=ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

The buffer overflow exposes a vulnerability, but it doesn't necessarily cause damage by itself. However, once attackers discover the vulnerability, they exploit it and overwrite memory locations with their own code. If the attacker uses the buffer overflow to crash the system or disrupt its services, it is a DoS attack.

More often, the attacker's goal is to insert malicious code in a memory

location that the system will execute. It's not easy for an attacker to know the exact memory location where the malicious code is stored, making it difficult to get the computer to execute it. However, an attacker can make educated guesses to get close.

A popular method that makes guessing easier is with no operation (NOP, pronounced as "no-op") commands, written as a NOP slide or NOP sled. Many Intel processors use hexadecimal 90 (often written as x90) as a NOP command, so a string of x90 characters is a NOP sled. The attacker writes a long string of x90 instructions into memory, followed by malicious code. When a computer is executing code from memory and it comes to a NOP, it just goes to the next memory location. With a long string of NOPs, the computer simply slides through all of them until it gets to the last one and then executes the code in the next instruction. If the attacker can get the computer to execute code from a memory location anywhere in the NOP slide, the system will execute the attacker's malicious code.

The malicious code varies. In some instances, the attackers write code to spread a worm through the web server's network. In other cases, the code modifies the web application so that the web application tries to infect every user who visits the web site with other malware. The attack possibilities are almost endless.

A buffer overflow attack includes several different elements, but they happen all at once. The attacker sends a single string of data to the application. The first part of the string causes the buffer overflow. The next part of the string is a long string of NOPs followed by the attacker's malicious code, stored in the attacked system's memory. Last, the malicious code goes to work.

In some cases, an attacker writes a malicious script to discover buffer overflow vulnerabilities. For example, the attacker could use JavaScript to send random data to another service on the same system.

Although error-handling routines and input validation go a long way to prevent buffer overflows, they don't prevent them all. Attackers occasionally discover a bug allowing them to send a specific string of data to an application, causing a buffer overflow. When vendors discover buffer overflow vulnerabilities, they are usually quick to release a patch or hotfix. From an administrator's perspective, the solution is easy: Keep the systems up to date with current patches.

Remember this

Buffer overflows occur when an application receives more data than it can handle, or receives unexpected data that exposes system memory. Buffer overflow attacks often include NOP instructions (such as x90) followed by malicious code. When successful, the attack causes the system to execute the malicious code. Input validation helps prevent buffer overflow attacks.

Pointer Dereference

Programming languages such as C, C++, and Pascal commonly use pointers, which simply store a reference to something. Some languages such as Java call them references.

As an example, imagine an application has multiple modules. When a new customer starts an order, the application invokes the CustomerData module. This module needs to populate the city and state in a form after the user enters a zip code.

How does the module get this array? One way is to pass the entire array to the module when invoking it. However, this consumes a lot of memory.

The second method is to pass a reference to the data array, which is simply a pointer to it. This consumes very little memory and is the preferred method. This method uses a ***pointer dereference***.

Dereferencing is the process of using the pointer to access the data array. Imagine the pointer is named ptrZip and the name of the full data array is named arrZip. The value within ptrZip is arrZip, which references the array. What is this thing that the pointer points to? There isn't a standard name, but some developers refer to it as a pointee.

Ah, I hear your question. What's the point?

A failed dereference operation can cause an application to crash. In some programming languages, it can subtly corrupt memory, which can be even worse than a crash. The subtle, random changes result in the application using incorrect data. This can often be difficult to troubleshoot and correct.

The cause of a failed dereference operation is a pointer that references a nonexistent pointee. If we continue with the previous example, imagine that the ptrZip pointer contains the value of arrayZip. This points to a nonexistent pointee because the array is named arrZip.

Admittedly, this programming error (using arrayZip instead of arrZip) would be quickly discovered because the CustomerData module wouldn't

correctly populate the city and state. However, other pointer dereferencing problems aren't so easy to discover.

DLL Injection

Applications commonly use a Dynamic Link Library (DLL) or multiple DLLs. A DLL is a compiled set of code that an application can use without re-creating the code. As an example, most programming languages include math-based DLLs. Instead of writing the code to discover the square root of a number, a developer can include the appropriate DLL and access the square root function within it.

DLL injection is an attack that injects a DLL into a system's memory and causes it to run. For example, imagine an attacker creates a DLL named *malware.dll* that includes several malicious functions. In a successful DLL injection attack, the attacker attaches to a running process, allocates memory within the running process, connects the malicious DLL within the allocated memory, and then executes functions within the DLL.

Summarizing Secure Coding Concepts

Secure application development and deployment concepts are important for application developers to understand. Additionally, IT security managers who manage development projects should understand these concepts, too, even if they aren't writing the code.

Applications often provide an avenue for attackers to generate attacks unless developers create them using secure coding concepts. This section covers common methods used to create secure applications.

Compiled Versus Runtime Code

Compiled code has been optimized by an application (called a compiler) and converted into an executable file. The compiler checks the program for errors and provides a report of items developers might like to check. Some commonly used compiled programming languages are C, C++, Visual Basic, and Pascal.

Runtime code is code that is evaluated, interpreted, and executed when the code is run. As an example, HTML is the standard used to create web pages. It includes specific tags that are interpreted by the browser, when it renders the web page. HTML-based web pages are interpreted at runtime.

Many languages use a cross between compiled and runtime code. For example, Python is an interpreted language widely used to create sophisticated web sites. However, when it is first run, the Python interpreter compiles it. The server will then use the compiled version each time it runs. If the system detects a change in the Python source code, it will recompile it.

Proper Input Validation

One of the most important security steps that developers should take is to include ***input validation***. Input validation is the practice of checking data for validity before using it. Input validation prevents an attacker from sending malicious code that an application will use by either sanitizing the input to remove malicious code or rejecting the input.

Improper input handling (or the lack of input validation) is one of the most common security issues on web-based applications. It allows many different types of attacks, such as buffer overflow attacks, SQL injection, command injection, and cross-site scripting attacks. Each of these attacks is covered within this chapter.

Consider a web form that includes a text box for a first name. You can logically expect a valid first name to have only letters, and no more than 25 letters. The developer uses input validation techniques to ensure that the name entered by the user meets this validity check. If a user enters other data, such as numbers, semicolons, or HTML code, it fails the validity check. Instead of using the data, the application rejects it and provides an error to the user.

You've probably seen input validation checks and error-handling routines in use if you've ever filled out a form on a web page. If you didn't fill out all the required text boxes, or if you entered invalid data into one or more of the boxes, the web site didn't crash. Instead, it redisplayed the page and showed an error. Web sites often use a red asterisk next to text boxes with missing or invalid data.

Some common checks performed by input validation include:

- **Verifying proper characters.** Some fields such as a zip code use

only numbers, whereas other fields such as state names use only letters. Other fields are a hybrid. For example, a phone number uses only numbers and dashes. Developers can configure input validation code to check for specific character types, and even verify that characters are entered in the correct order. For example, a telephone number mask of ###-###-#### accepts only three numbers, a dash, three numbers, a dash, and four numbers.

- **Implementing boundary or range checking.** These checks ensure that values are within expected boundaries or ranges. For example, if the maximum purchase for a product is three, a range check verifies the quantity is three or less. The validation check identifies data outside the range as invalid and the application does not use it.
- **Blocking HTML code.** Some malicious attacks embed HTML code within the input as part of an attack. These can be blocked by preventing the user from entering the HTML code, such as the < and > characters.
- **Preventing the use of certain characters.** Some attacks, such as SQL injection attacks, use specific characters such as the dash (-), apostrophe (‘), and equal sign (=). Blocking these characters helps to prevent these attacks.

Client-Side and Server-Side Input Validation

It’s possible to perform input validation at the client and the server. Client-side execution indicates that the code runs on the client’s system, such as a user’s web browser. Server-side execution indicates that the code runs on the server, such as on a web server.

Client-side input validation is quicker, but is vulnerable to attacks. Server-side input validation takes longer, but is secure because it ensures the application doesn’t receive invalid data. Many applications use both. Imagine Homer is using a web browser to purchase the newest version of Scrabbleships through the Duff web site. Customers cannot purchase more than three at a time.

In client-side input validation, the validation code is included in the HTML page sent to Homer. If he enters a quantity of four or more, the HTML code gives him an error message, and doesn’t submit the page to the server until Homer enters the correct data.

Unfortunately, it’s possible to bypass client-side validation techniques.

Many web browsers allow users to disable JavaScript in the web browser, which bypasses client-side validation. It's also possible to use a web proxy to capture the data sent from the client in the HTTP POST command and modify it before forwarding to the server.

Server-side input validation checks the inputted values when it reaches the server. This ensures that the user hasn't bypassed the client-side checks.

Using both client-side and server-side validation provides speed and security. The client- side validation checks prevent round-trips to the server until the user has entered the correct data. The server-side validation is a final check before the server uses the data.

Remember this

The lack of input validation is one of the most common security issues on web-based applications. Input validation verifies the validity of inputted data before using it, and server-side validation is more secure than client-side validation. Input validation protects against many attacks, such as buffer overflow, SQL injection, command injection, and cross- site scripting attacks.

Other Input Validation Techniques

Other input validation techniques attempt to sanitize HTML code before sending it to a web browser. These methods are sometimes referred to as escaping the HTML code or encoding the HTML code. As a simple example, the greater than symbol (>) can be encoded with the ASCII replacement characters (>). Doing so, along with following specific guidelines related to not inserting untrusted data into web pages, helps prevent many web application attacks.

Most languages include libraries that developers can use to sanitize the HTML code. As an example, the Open Web Application Security Project (OWASP) Enterprise Security API (ESAPI) is a free, open source library available for many programming languages. It includes a rich set of security-based tools, including many used for input validation.

Avoiding Race Conditions

When two or more modules of an application, or two or more

applications, attempt to access a resource at the same time, it can cause a conflict known as a ***race condition***. Most application developers are aware of race conditions and include methods to avoid them when writing code. However, when new developers aren't aware of race conditions, or they ignore them, a race condition can cause significant problems.

As a simple example of a potential problem, imagine you are buying a plane ticket online and use the web application to pick your seat. You find a window seat and select it. However, at the same time you're selecting this window seat, someone else is, too. You both make the purchase at the same time and you both have tickets with the same seat number. You arrive after the other person and he's unwilling to move, showing his ticket with the seat number. A flight attendant ultimately helps you find a seat. Unfortunately, it's between two burly gentlemen who have been on an all-cabbage diet for the last week. You probably wouldn't be too happy.

Online ticketing applications for planes, concerts, and other events avoid this type of race condition. In some cases, they lock the selection before offering it to a customer. In other cases, they double-check for a conflict later in the process. Most database applications have internal concurrency control processes to prevent two entities from modifying a value at the same time. However, inexperienced web application developers often overlook race conditions.

Proper Error Handling

Error-handling and exception-handling routines ensure that an application can handle an error gracefully. They catch errors and provide user-friendly feedback to the user. When an application doesn't catch an error, it can cause the application to fail. In the worst-case scenario, improper error-handling techniques within an application can cause the operating system to crash. Using effective error- and exception-handling routines protects the integrity of the underlying operating system.

Improper error handling can often give attackers information about an application. When an application doesn't catch an error, it often provides debugging information that attackers can use against the application. In contrast, when an application catches the error, it can control what information it shows to the user. There are two important points about error reporting:

- **Errors to users should be general.** Detailed errors provide information that attackers can use against the system, so the errors should be general. Attackers can analyze the errors to determine details about the system. For example, if an application is unable to connect with a database, a detailed error can let the attacker know exactly what type of database the system is using. This indirectly lets the attacker know what types of commands the system will accept. Also, detailed errors confuse most users.
- **Detailed information should be logged.** Detailed information on the errors typically includes debugging information. By logging this information, it makes it easier for developers to identify what caused the error and how to resolve it.

Remember this

Error and exception handling helps protect the integrity of the operating system and controls the errors shown to users. Applications should show generic error messages to users but log detailed information.

Cryptographic Techniques

Chapter 10 discusses various cryptographic techniques used for ***encryption*** and ***authentication***. Some of these techniques can also be used when applying secure coding techniques.

In general, sensitive data is often encrypted to prevent the unauthorized disclosure of data. If an application is accessing any sensitive data, developers need to ensure that this access doesn't result in inadvertent data exposure. For example, if an application accesses encrypted data on a different server, the application needs to ensure that the data is encrypted while in transit.

Applications need to decrypt data before processing it. When done processing the data, applications need to encrypt the data before storing it. Additionally, applications need to ensure that all remnants of the data are flushed from memory.

Certificates are used for various purposes such as authenticating users and computers. They can also be used to authenticate and validate software code. As an example, developers can purchase a certificate and associate it with an application or code. This ***code signing*** process provides a digital

signature for the code and the certificate includes a hash of the code. This provides two benefits. First, the certificate identifies the author. Second, the hash verifies the code has not been modified. If malware changes the code, the hash no longer matches, alerting the user that the code has been modified.

Code Reuse and SDKs

Developers are encouraged to reuse code whenever possible. As an example, imagine a developer created code for a web application to create, modify, and authenticate users and this code has been in use for a year. The code has gone through internal testing and has survived the use within the application. Instead of creating brand-new code for a new application, it's best to use this tested code. Code reuse saves time and helps prevent the introduction of new bugs.

However, when reusing code, developers should ensure that they are using all the code that they copy into another application. As an example, imagine a developer has created a module that has three purposes: create users, modify users, and authenticate users. While working on a new application, he realizes he needs a module that will authenticate users. If he simply copies the entire module into the new application, it creates dead code. ***Dead code*** is code that is never executed or used. In this example, the copied code to create and modify users isn't used in the new application, so it is dead code.

Logic errors can also create dead code. For example, imagine a function tests the value of a variable called Donuts. If Donuts has a value (such as 12), it squares it. If Donuts is null (a value of nothing), it returns an error and exits the function.

Next, the function checks to see if Donuts is null and if so, it prints a message in an error log. Do you see the error? The code to print to an error log never executes. If Donuts is null, the previous check exited the function, so the second check never occurs. This logic error creates the dead code.

Another popular method of code reuse is the use of third-party libraries. As an example, JavaScript is a rich, interpreted language used by many web applications. Netscape originally developed it and it was ultimately standardized as an open source language.

Software development kits (SDKs) are like third-party libraries, but they

are typically tied to a single vendor. For example, if you're creating an Android app, you can use the Android SDK. It includes software tools that will help you create apps for Android-based devices.

Code Obfuscation

Developers often spend a lot of time developing code. If it is JavaScript, it is rather easy for other developers to just copy the code and use it. One way to slow this down is with an ***obfuscation***/camouflage method.

Obfuscation attempts to make something unclear or difficult to understand. Code obfuscation (or code camouflage) attempts to make the code unreadable. It does things like rename variables, replace numbers with expressions, replace strings of characters with hexadecimal codes, and remove comments. For example, a meaningful variable of strFirstName might be renamed to 94mdiwl, and the number 11 might be changed to 0xF01B – 0x73 – 0xEF9D (which still results in the decimal number 11).

It's worth noting that most security experts reject security through obscurity as a reliable method of maintaining security. Similarly, code obfuscation might make the code difficult to understand by most people. However, it's still possible for someone with skills to dissect the code.

Code Quality and Testing

Many organizations that create applications also employ testers to verify the quality of the code. Testers use a variety of different methods to put the code through its paces. Ideally, they will detect problems with the code before it goes live. Some of the common methods of testing code include:

- **Static code analyzers.** Static code analysis examines the code without executing it. Automated tools can analyze code and mark potential defects. Some tools work as the developer creates the code, similar to a spell checker. Other tools can examine the code once it is semifinalized.
- **Dynamic analysis.** Dynamic analysis checks the code as it is running. A common method is to use fuzzing. Fuzzing uses a computer program to send random data to an application. In some cases, the random data can crash the program or create unexpected results, indicating a vulnerability. Problems discovered during a

dynamic analysis can be fixed before releasing the application.

- **Stress testing.** Stress testing methods attempt to simulate a live environment and determine how effective or efficient an application operates with a load. As an example, a web application is susceptible to a DDoS attack. A stress test can simulate a DDoS attack and determine its impact on the web application.
- **Sandboxing.** A sandbox is an isolated area used for testing programs. The term comes from a sandbox in a playground. Children can play in the sandbox where they are relatively safe (and parents can easily keep their eyes on them). Similarly, application developers can test applications in a sandbox, knowing that any changes they make will not affect anything outside the sandbox. Virtual machines (VMs) are often used for sandboxing. For example, Java virtual machines include a sandbox to restrict untrusted applications.
- **Model verification.** Testing helps identify and remove bugs. However, it's also important that the software does what it's meant to do. Model verification is the process of ensuring that software meets specifications and fulfills its intended purpose.

Remember this

Static code analysis examines the code without running it and dynamic analysis checks the code while it is running. Fuzzing techniques send random strings of data to applications looking for vulnerabilities. Stress testing verifies an application can handle a load. Sandboxing runs an application within an isolated environment to test it. Model verification ensures that the application meets all specifications and fulfills its intended purpose.

Development Life-Cycle Models

Software development life cycle (SDLC) models attempt to give structure to software development projects. Two popular models are waterfall and agile.

The ***waterfall*** model includes multiple stages going from top to bottom. Each stage feeds the next stage, so when you finish one stage, you move on to the next stage. When following the waterfall model strictly, you

don’t go back to a stage after finishing it. There are multiple variations of the waterfall model, but they all use stages. However, the names of these stages vary from one model to another. Some typical stages used with the waterfall model include:

- **Requirements.** The developers work with the customer to understand the req
  The output of this stage is a requirements document, which provides clear guidance on what the application will do.
- **Design.** Developers begin to design the software architecture in this stage. This is similar to creating the blueprints for a building. The design stage doesn’t include any detailed coding, but instead focuses on the overall structure of the project.
- **Implementation.** Developers write the code at this stage, based on the requirements and design.
- **Verification.** The verification stage ensures the code meets the requirements.
- **Maintenance.** The maintenance stage implements changes and updates as desired.

A challenge with the waterfall model is that it lacks flexibility. It is difficult to revise anything from previous stages. For example, if a customer realizes a change in the requirements is needed, it isn’t possible to implement this change until the maintenance stage.

The ***agile*** model uses a set of principles shared by cross-functional teams. These principles stress interaction, creating a working application, collaborating with the customer, and responding to change.

Instead of strict phases, the agile model uses iterative cycles. Each cycle creates a working, if not complete, product. Testers verify the product works with the current features and then developers move on to the next cycle. The next cycle adds additional features, often adding small, incremental changes from the previous cycle.

A key difference of the agile model (compared with the waterfall model) is that it emphasizes interaction between customers, developers, and testers during each cycle. In contrast, the waterfall model encourages interaction with customers during the requirements stage, but not during the design and implementation stages.

The agile model can be very effective if the customer has a clear idea of the requirements. If not, the customer might ask for changes during each

cycle, extending the project's timeline.

Secure DevOps

DevOps combines the words development and operations and it is an agile-aligned software development methodology. ***Secure DevOps*** is a software development process that includes extensive communication between software developers and operations personnel. It also includes security considerations throughout the project. When applied to a software development project, it can allow developers to push out multiple updates a day in response to changing business needs.

Some of the concepts included within a secure DevOps project are summarized in the following bullets:

- **Security automation** uses automated tests to check code. When modifying code, it's important to test it and ensure that the code doesn't introduce software bugs or security flaws. It's common to include a mirror image of the production environment and run automated tests on each update to ensure it is error free.
- **Continuous integration** refers to the process of merging code changes into a central repository. Software is then built and tested from this central repository. The central repository includes a version control system, and the version control system typically supports rolling back code changes when they cause a problem.
- **Baselining** refers to applying changes to the baseline code every day and building the code from these changes. For example, imagine five developers are working on different elements of the same project. Each of them have modified and verified some code on their computers. At the end of the day, each of these five developers uploads and commits their changes. Someone then builds the code with these changes and then automation techniques check the code. The benefit is that bugs are identified and corrected quicker. In contrast, if all the developers applied their changes once a week, the bugs can multiply and be harder to correct.
- **Immutable systems** cannot be changed. Within the context of secure DevOps, it's possible to create and test systems in a controlled environment. Once they are created, they can be deployed into a production environment. As an example, it's possible to create a

secure image of a server for a specific purpose. This image can be deployed as an immutable system to ensure it stays secure.

- **Infrastructure as code** refers to managing and provisioning data centers with code that defines virtual machines (VMs). Chapter 1 introduces virtualization concepts and many VMs are created with scripts. Once the script is created, new VMs can be created just by running the script.

Remember this

SDLC models provide structure for software development projects. Waterfall uses multiple stages going from top to bottom, with each stage feeding the next stage. Agile is a flexible model that emphasizes interaction with all players in a project. Secure DevOps is an agile-aligned methodology that stresses security throughout the lifetime of the project.

Version Control and Change Management

Chapter 5, "Securing Hosts and Data," covers change management policies for IT systems. The primary purpose is to ensure that changes to systems do not cause unintended outages. Secure coding practices use version control and change management practices for the same reason—to prevent unintended outages.

Change management helps ensure that developers do not make unauthorized changes. As an example, if a customer wants a change or addition to the application, a developer doesn't just implement it, no matter how easy it might be to do so. Instead, any changes to the application go through a specific, predefined process.

The change management process allows several people to examine the change to ensure it won't cause unintended consequences. Also, any change to the application becomes an added responsibility. If the customer discovers a bug due to this change after it's delivered, the developer may be responsible for fixing it, even if it wasn't authorized.

In addition to preventing unauthorized changes and related problems, a

change management process also provides an accounting structure to document the changes. Once a change is authorized and implemented, the change is documented in a version control document.

Version control tracks the versions of software as it is updated, including who made the update and when. Many advanced software development tools include sophisticated version control systems. Developers check out the code to work on it and check it back into the system when they're done. The version control system can then document every single change made by the developer. Even better, this version control process typically allows developers to roll back changes to a previous version when necessary.

Provisioning and Deprovisioning

Provisioning and deprovisioning typically refers to user accounts. For example, when an employee starts working at an organization, administrators create the account and give the account appropriate privileges. This way, the user can use the account as authorization to access various resources. Deprovisioning an account refers to removing access to these resources and can be as simple as disabling the account.

Within the context of secure application development and deployment concepts, these terms apply to an application. Provisioning an application refers to preparing and configuring the application to launch on different devices and to use different application services.

As an example, developers who create iOS apps (running on Apple devices) provision the apps based on the devices they'll run on. Apps can run on iPhones, iPads, and Macs. Additionally, these apps can use different services, such as an accelerometer and gyroscope to detect movement. The app needs to be properly provisioned with the appropriate code on the target device to use these services.

Deprovisioning an app refers to removing it from a device. For example, if a user decides to delete the app, the app should be able to remove it completely. Leaving remnants of the app consumes resources on the device.

Identifying Application Attacks

Many attacks target server applications such as those hosted on web

servers. Web servers are highly susceptible to several types of attacks, such as buffer overflow attacks and SQL injection attacks, because they commonly accept data from users. Other servers are susceptible to some types of command injection attacks. This section covers many of the common attacks related to different types of servers.

Web Servers

Web servers most commonly host web sites accessible on the Internet, but they can also serve pages within an internal network. Organizations place web servers within a demilitarized zone (DMZ) to provide a layer of protection.

The two primary applications used for web servers are:

- **Apache.** Apache is the most popular web server used on the Internet. It's free and can run on Unix, Linux, and Windows systems.
- **Internet Information Services (IIS).** IIS is a Microsoft web server, and it's included free with any Windows Server product.

Establishing a web presence is almost a requirement for organizations today, and users expect fancy web sites with dynamic pages that are easy to use. Although many applications make it easy to create web sites, they don't always include security. This often results in many web sites being highly susceptible to attacks. The following sections identify many common attacks on web sites.

Database Concepts

Several of the secure coding techniques and attacks apply directly to databases, so they're organized in this section. SQL (pronounced as "sequel" or "es-que-el") is a Structured Query Language used to communicate with databases. SQL statements read, insert, update, and delete data to and from a database. Many web sites use SQL statements to interact with a database, providing users with dynamic content.

A database is a structured set of data. It typically includes multiple tables and each table holds multiple columns and rows. As an example, consider Figure 7.3. It shows the database schema for a database intended to hold information about books and their authors. It includes two incorrect entries, which are described in the "Normalization" section.

Figure 7.3: Database schema

The Book table (on the left) identifies the column names for the table. Each of these columns has a name and identifies the data type or attribute type allowed in the column. For example, INT represents integer, VARCHAR represents a variable number of alphanumeric characters, TEXT is used for paragraphs, and decimal can store monetary values.

The Author table holds information on authors, such as their names and addresses. The BookAuthor table creates a relationship between the Book table and the Author table. The Publisher column should not be there, but it helps describe normalization in the next section.

Column

AuthorID	FirstName	LastName	StreetAddress	City	State	
1	Lisa	Simpson	742 Evergreen Terrace	Springfield	IDK	← Row
2	Moe	Szylak	1313 Walnut Street	Springfield	IDK	← Row
3	Ned	Flanders	744 Evergreen Terrace	Springfield	IDK	← Row

Figure 7.4 shows three rows of the Author table. It also shows the difference between columns and rows. Because the column identifies the data type, columns are sometimes referred to as attributes. Also, because each row represents a record, rows are sometimes called records or tuples.

Figure 7.4: Database table

Individual elements within a database are called fields. For example, the field in the second row of the FirstName column is a field holding the value of Moe.

Normalization

Normalization of a database refers to organizing the tables and columns to reduce redundant data and improve overall database performance. Although there are several normal forms, the first three are the most important.

First Normal Form

A database is in first normal form (1NF) if it meets the following three criteria:

- **Each row within a table is unique and identified with a primary key.** For example, the Author table has a primary key of AuthorID and each row within the table has a different and unique AuthorID, or a different primary key. Primary keys are shown in Figure 7.3 as small key icons. The primary key in the Book table is BookID. The BookAuthor has a composite primary key using two values: Book_BookID and Author_AuthorID.
- **Related data is contained in a separate table.** The author information is contained in a different table. While it's possible to create one single table to hold all the information, this creates multiple problems. First, you'd have to create several extra columns such as FirstName, LastName, and so on every time you added a book. Imagine Lisa Simpson writes five books. Each of her books in the book table would then need to include all of Lisa's information. Entering the same information multiple times increases the chance for errors. If she moves to a new address, you need to change the address five times.
- **None of the columns include repeating groups.** As an example, the Author table includes FirstName for the first name and LastName for the last name. If you combine these into a single column of name, it violates this rule. It also makes it more difficult to access only one part of the repeating group, such as the first name or the last name.

Second Normal Form

Second normal form (2NF) only applies to tables that have a composite primary key, where two or more columns make up the full primary key. The BookAuthor table has a composite key that includes the Book_BookID column and the Author_AuthorID column. A database is in 2NF if it meets the following criteria:

- It is in 1NF.
- Non-primary key attributes are completely dependent on the composite primary key. If any column is dependent on only one column of the composite key, it is not in 2NF.

The BookAuthor table shown in Figure 7.3 violates this with the Publisher column. A book has a unique publisher so the publisher is related to

the Book_BookID column. However, an author can publish books through multiple publishers, so the publisher value is not dependent on the Author_AuthorID column.

Notice that the Book table correctly has the Publisher column, so the easy fix to have this database in 2NF is to delete the Publisher column in the BookAuthor table.

Third Normal Form

Third normal form (3NF) helps eliminate unnecessary redundancies within a database. A database is in 3NF if it meets the following criteria:

- It is in 2NF. This implies it is also in 1NF.
- All columns that aren't primary keys are only dependent on the primary key. In other words, none of the columns in the table are dependent on non-primary key attributes.

The Book table violates the second rule of 3NF with the PublisherCity column. The city where the publisher is located is dependent on the publisher, not the book. Imagine this table had 100 book entries from the same publisher located in Virginia Beach. When entering the data, you'd need to repeatedly enter Virginia Beach for this publisher.

There are two ways to fix this. First, ask if the city is needed. If not, delete the column and the database is now in 3NF. If the city is needed, you can create another table with publisher data. You would then relate the Publisher table with the Book table.

Remember this

Normalization is a process used to optimize databases. While there are several normal forms available, a database is considered normalized when it conforms to the first three normal forms.

SQL Queries

One of the vulnerabilities related to databases is SQL injection attacks. The following sections identify how SQL queries work, how attackers launch a SQL injection attack, and how to protect against SQL injection attacks.

As a simple example of a web site that uses SQL queries, think of Amazon.com. When you enter a search term and click Go (as shown in Figure 7.5), the web application creates a SQL query, sends

it to a database server, and formats the results into a web page that it sends back to you.

Figure 7.5: Web page querying a database with SQL

In the example, I selected the Books category and entered **Darril Gibson**. The result shows a list of books authored by Darril Gibson available for sale on Amazon. The query sent to the database from the Amazon web application might look like this:

```
SELECT * FROM Books WHERE Author ='Darril Gibson'
```

The * is a wildcard and returns all columns in a table. Notice that the query includes the search term entered into the web page form (Darril Gibson) and encloses the search term in single quotes. If the web site simply plugs the search term into the SELECT statement, surrounded by single quotes, it will work, but it's also highly susceptible to SQL injection attacks.

SQL Injection Attacks

In a SQL injection attack, the attacker enters additional data into the web page form to generate different SQL statements. SQL query languages use a semicolon (;) to indicate the end of the SQL line and use two dashes (--) as an ignored comment. With this knowledge, the attacker could enter different information into the web form like this:

Darril Gibson'; SELECT * FROM Customers;--

If the web application plugged this string of data directly into the SELECT statement surrounded by the same single quotes, it would look like this:

```
SELECT * FROM Books WHERE Author
='Darril Gibson'; SELECT * FROM
Customers;
--'
```

The first line retrieves data from the database, just as before. However, the semicolon signals the end of the line and the database will accept another command. The next line reads all the data in the Customers table, which can give the attacker access to names, credit card data, and more. The last line comments out the second single quote to prevent a SQL error.

If the application doesn't include error-handling routines, these errors provide details about the type of database the application is using, such as an

Oracle, Microsoft SQL Server, or MySQL database. Different databases format SQL statements slightly differently, but once the attacker learns the database brand, it's a simple matter to format the SQL statements required by that brand. The attacker then follows with SQL statements to access the database and may allow the attacker to read, modify, delete, and/or corrupt data.

This attack won't work against Amazon (please don't try it) because Amazon is using secure coding principles. I don't have access to its code, but I'd bet the developers are using input validation and SQL-based stored procedures (described in the next section).

Many SQL injection attacks use a phrase of **or '1' = '1'** to create a true condition. For example, if an online database allows you to search a Customers table looking for a specific record, it might expect you to enter a name. If you entered **Homer Simpson**, it would create a query like this:

SELECT * FROM Customers WHERE name ='Homer Simpson'

This query will retrieve a single record for Homer Simpson. However, if the attacker enters

' or'1'='1'-- instead of Homer Simpson, it will create a query like this: SELECT * FROM Customers WHERE name =' ' or '1'='1' --'

Although this is a single SELECT statement, the **or** clause causes it to behave as two separate SELECT statements:

SELECT * FROM Customers WHERE
name =' ' SELECT * FROM Customers
WHERE '1'='1'

The first clause will likely not return any records because the table is unlikely to have any records with the name field empty. However, because the number 1 always equals the number 1, the WHERE clause in the second statement always equates to True, so the SELECT statement retrieves all records from the Customers table.

In many cases, a SQL injection attack starts by sending improperly formatted SQL statements to the system to generate errors. Proper error handling prevents the attacker from gaining information from these errors, though. Instead of showing the errors to the user, many web sites simply present a generic error web page that doesn't provide any details.

Input validation and stored procedures reduce the risk of SQL injection attacks.

Remember this

Attackers use SQL injection attacks to pass queries to back-end databases through web servers. Many SQL injection attacks use the phrase ‘ or ‘1’=’1’ -- to trick the database server into providing information.

Protecting Against SQL Injection Attacks

As mentioned previously, input validation provides strong protection against SQL injection attacks. Before using the data entered into a web form, the web application verifies that the data is valid.

Additionally, database developers often use ***stored procedures*** with dynamic web pages. A stored procedure is a group of SQL statements that execute as a whole, similar to a mini- program. A parameterized stored procedure accepts data as an input called a parameter. Instead of copying the user’s input directly into a SELECT statement, the input is passed to the stored procedure as a parameter. The stored procedure performs data validation, but it also handles the parameter (the inputted data) differently and prevents a SQL injection attack.

Consider the previous example searching for a book by an author where an attacker entered the following text: **Darril Gibson’; SELECT * From Customers;--**. The web application passes this search string to a stored procedure. The stored procedure then uses the entire search string in a SELECT statement like this:

SELECT * From Books Where Author =“Darril Gibson’; SELECT * From Customers;-- ”

In this case, the text entered by the user is interpreted as harmless text rather than malicious SQL statements. It will look for books with an author name using all of this text: Darril Gibson’; SELECT * From Customers;--. Books don’t have names with SELECT statements embedded in them, so the query comes back empty.

Depending on how well the database server is locked down (or not), SQL injection attacks may allow the attacker to access the structure of the database, all the data, and even modify data. In some cases, attackers have modified the price of products from several hundred dollars to just a few dollars, purchased several of them, and then returned the price to normal.

Injection Attacks

There are multiple types of injection attacks beyond DLL injection and SQL injection attacks discussed previously in this chapter. Another type of ***injection attack*** is a command injection attack.

In some cases, attackers can inject operating system commands into an application using web page forms or text boxes. Any web page that accepts input from users is a potential threat. Directory traversal is a specific type of command injection attack that attempts to access a file by including the full directory path, or traversing the directory structure.

For example, in Unix systems, the passwd file includes user logon information, and it is stored in the /etc directory with a full directory path of /etc/passwd. Attackers can use commands such as**../../etc/passwd** or **/etc/passwd** to read the file. Similarly, they could use a remove directory command (such as **rm -rf**) to delete a directory, including all files and subdirectories. Input validation can prevent these types of attacks.

Cross-Site Scripting

Cross-site scripting (XSS) is another web application vulnerability that can be prevented with input validation techniques. Attackers embed malicious HTML or JavaScript code into a web site's code. The code executes when the user visits the site.

You may be wondering why the acronym isn't CSS instead of XSS. The reason is that web sites use Cascading Style Sheets identified as CSS and CSS files are not malicious.

The primary protection against XSS attacks is at the web application with sophisticated input validation techniques. Developers should avoid any methods that allow the web page to display untrusted data. Additionally, OWASP strongly recommends the use of a security encoding library. When implemented, an encoding library will sanitize HTML code and prevent XSS attacks. OWASP includes more than 10 rules that developers can follow to prevent XSS attacks.

It's also important to educate users about the dangers of clicking links. Some XSS attacks send emails with malicious links within them. The XSS attack fails if users do not click the link.

Remember this

Cross-site scripting (XSS) attacks allow attackers to capture user information such as cookies. Input validation techniques at the server help prevent XSS attacks.

Cross-Site Request Forgery

Cross-site request forgery (***XSRF*** or CSRF) is an attack where an attacker tricks a user into performing an action on a web site. The attacker creates a specially crafted HTML link and the user performs the action without realizing it.

As an innocent example of how HTML links create action, consider this HTML link: *http://www.google.com/search?q=Success.* If users click this link, it works just as if the user browsed to Google and entered Success as a search term. The *?q=Success* part of the query causes the action.

Many web sites use the same type of HTML queries to perform actions. For example, imagine a web site that supports user profiles. If users wanted to change profile information, they could log on to the site, make the change, and click a button. The web site may use a link like this to perform the action:

http://getcertifiedgetahead.com/edit?
action=set&key=email&value=you@home.com

Attackers use this knowledge to create a malicious link. For example, the following link could change the email address in the user profile, redirecting the user's email to the attacker:

http://getcertifiedgetahead.com/edit?
action=set&key=email&value=hacker@hackersrs.com

Although this shows one possibility, there are many more. If a web site supports any action via an HTML link, an attack is possible. This includes making purchases, changing passwords, transferring money, and much more.

Web sites typically won't allow these actions without users first logging on. However, if users have logged on before, authentication information is stored on their system either in a cookie or in the web browser's cache. Some web sites automatically use this information to log users on as soon as they visit. In some cases, the XSRF attack allows the attacker to access the user's password.

Users should be educated on the risks related to links from sources they don't recognize. Phishing emails (covered in Chapter 6) often include malicious links that look innocent enough to users, but can cause significant harm. If users don't click the link, they don't launch the XSRF attack.

However, just as with cross-site scripting, the primary burden of protection from XSRF falls on the web site developers. Developers need to be aware of XSRF attacks and the different methods used to protect against them. One method is to use dual authentication and force the user to manually enter credentials prior to performing actions. Another method is to expire the cookie after a short period, such as after 10 minutes, preventing automatic logon for the user.

Many programming languages support XSRF tokens. For example, Python and Django, two popular web development languages, require the use of an XSRF token in any page that includes a form, though these languages call them CSRF tokens. This token is a large random number generated each time the form is displayed. When a user submits the form, the web page includes the token along with other form data. The web application then verifies that the token in the HTML request is the same as the token included in the web form.

The HTML request might look something like this: *getcertifiedgetahead.com/edit? action=set&key=email&value=you@home.com&token=1357924* The token is typically much longer. If the website receives a query with an incorrect error, it

typically raises a 403 Forbidden error. Attackers can't guess the token, so they can't craft malicious links that will work against the site.

Remember this

Cross-site request forgery (XSRF) scripting causes users to perform actions on web sites, such as making purchases, without their knowledge. In some cases, it allows an attacker to steal cookies and harvest passwords.

Understanding Frameworks and Guides

Within the context of cybersecurity, there are multiple references available that describe best practices and provide instructions on how to secure systems. Some of these are industry- standard frameworks, while

others are platform- or vendor-specific guides.

A ***framework*** is a structure used to provide a foundation. Cybersecurity frameworks typically use a structure of basic concepts and they provide guidance to professionals on how to implement security in various systems. Chapter 8 discusses various exploitation frameworks often used by penetration testers. Some generic categories of frameworks are:

- **Regulatory.** Regulatory frameworks are based on relevant laws and regulations. As an example, the Health Insurance Portability and Accountability Act (HIPAA) mandates specific protections of all health-related data. The Office of the National Coordinator for Health Information Technology (ONC) and the HHS Office for Civil Rights (OCR) created the HIPAA Security Risk Assessment (SRA) Tool. This tool provides a framework that organizations can use to help ensure compliance with HIPAA.
- **Non-regulatory.** A non-regulatory framework is not required by any law. Instead, it typically identifies common standards and best practices that organizations can follow. As an example, COBIT (Control Objectives for Information and Related Technologies) is a framework that many organizations use to ensure that business goals and IT security goals are linked together.
- **National versus international.** Some frameworks are used within a single country (and referred to as national frameworks), while others are used internationally. As an example, NIST created the Cybersecurity Framework, which focuses on cybersecurity activities and risks within the United States. In contrast, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) create and publish international standards. For example, ISO/IEC 27002 provides a framework for IT security.
- **Industry-specific.** Some frameworks only apply to certain industries. As an example, organizations that handle credit cards typically comply with the Payment Card Industry Data Security Standard (PCI DSS). PCI DSS includes 12 requirements and over 220 sub-requirements that organizations follow to protect credit card data.

In addition to frameworks, you can also use various guides to increase security. This includes benchmarks or secure configuration guides, platform-

or vendor-specific guides, and general- purpose guides. On the surface, this is quite simple. When configuring Windows systems, use a Windows guide to identify secure settings. When configuring Linux systems, use a Linux guide.

Additionally, when configuring a system for a specific role (such as a web server, application server, or network infrastructure device), follow the appropriate guide for that role. As an example, a web server would need ports 80 and 443 open for HTTP and HTTPS, respectively. However, a database application server would not typically need these ports open, so they should be closed on a database application server. The individual guides for each of the roles provide this information.

Chapter 7 Exam Topic Review

When preparing for the exam, make sure you understand the key concepts covered in this chapter.

Comparing Common Attacks

- A DoS attack is an attack launched from a single system and attempts to disrupt services.
- DDoS attacks are DoS attacks from multiple computers. DDoS attacks typically include sustained, abnormally high network traffic.
- Spoofing attacks attempt to impersonate another system. MAC address spoofing changes the source MAC address and IP spoofing changes the source IP address.
- ARP poisoning attacks attempt to mislead computers or switches about the actual MAC address of a system. They can be used to launch a man-in-the-middle attack.
- DNS poisoning attacks modify DNS data and can redirect users to malicious sites. Many DNS servers use DNSSEC to protect DNS records and prevent DNS poisoning attacks.
- Amplification attacks send increased traffic to, or request additional traffic from, a victim.
- Password attacks attempt to discover passwords. A brute force attack attempts to guess all possible character combinations and a dictionary attack uses all the words and character combinations stored in a file. Account lockout policies thwart online brute force attacks and complex passwords thwart offline password attacks.
- Passwords are often stored as a hash. Weak hashing algorithms are susceptible to collisions, which allow different passwords to create the same hash.
- In a pass the hash attack, the attacker discovers the hash of the user’s password and then uses it to log on to the system as the user.
- In a birthday attack, an attacker is able to create a password that produces the same hash as the user’s actual password. This is also known as a hash collision.

- A hash collision occurs when the hashing algorithm creates the same hash from different passwords.
- Password salting adds additional characters to passwords before hashing them and prevents many types of attacks, including dictionary, brute force, and rainbow table attacks.
- Replay attacks capture data in a session with the intent of using information to impersonate one of the parties. Timestamps and sequence numbers thwart replay attacks.
- A known plaintext attack is possible if an attacker has both the plaintext and the ciphertext created by encrypting the plaintext. It makes it easier to decrypt other data using a similar method.
- Attackers buy domain names with minor typographical errors in typo squatting (also called URL hijacking) attacks. The goal is to attract traffic when users enter incorrect URLs. Attackers can configure the sites with malware to infect visitors or configure the site to generate ad revenue for the attacker.
- Clickjacking tricks users into clicking something other than what they think they're clicking.
- Attackers utilize the user's session ID to impersonate the user in a session ID attack.
- Domain hijacking attacks allow an attacker to change the registration of a domain name without permission from the owner.
- A man-in-the-browser is a proxy Trojan horse that exploits vulnerable web browsers. When successful, it allows attacks to capture keystrokes and all data sent to and from the browser.
- A driver shim is additional code that can be run instead of the original driver.
- Attackers exploiting unknown or undocumented vulnerabilities are taking advantage of zero-day vulnerabilities. The vulnerability is no longer a zero-day vulnerability after the vendor releases a patch to fix it.
- Buffer overflows occur when an application receives more data, or unexpected data, than it can handle and exposes access to system memory. Integer overflow attacks attempt to use or create a numeric value bigger than the application can handle.

- Buffer overflow attacks exploit buffer overflow vulnerabilities. A common method uses NOP instructions or NOP sleds such as a string of x90 commands. Two primary protection methods against buffer overflow attacks are input validation and keeping a system up to date.

Summarizing Secure Coding Concepts

- Compiled code has been optimized by an application and converted into an executable file. Runtime code is code that is evaluated, interpreted, and executed when the code is run.
- A common coding error in web-based applications is the lack of input validation.
- Input validation checks the data before passing it to the application and prevents many types of attacks, including buffer overflow, SQL injection, command injection, and cross-site scripting attacks.
- Server-side input validation is the most secure. Attackers can bypass client-side input validation, but not server-side input validation.
- Race conditions allow two processes to access the same data at the same time, causing inconsistent results. Problems can be avoided by locking data before accessing it.
- Error-handling routines within applications can prevent application failures and protect the integrity of the operating systems. Error messages shown to users should be generic, but the application should log detailed information on the error.
- Code signing uses a digital signature within a certificate to authenticate and validate software code.
- Code quality and testing techniques include static code analysis, dynamic analysis (such as fuzzing), stress testing, sandboxing, and model verification.
- Software development life cycle (SDLC) models provide structure for software development projects. Waterfall uses multiple stages with each stage feeding the next stage. Agile is a more flexible model and it emphasizes interaction with all players in a project.
- Secure DevOps is an agile-aligned methodology. It stresses security throughout the lifetime of the project.

Identifying Application Attacks

- Common web servers are Apache (running on Linux) and Internet Information Services (running on Microsoft servers).
- Databases are optimized using a process called normalization. A database is considered normalized when it conforms to the first three normal forms.
- SQL injection attacks provide information about a database and can allow an attacker to read and modify data within a database. Input validation and stored procedures provide the best protection against SQL injection attacks.
- Cross-site scripting (XSS) allows an attacker to redirect users to malicious web sites and steal cookies. It uses HTML and JavaScript tags with < and > characters.
- Cross-site request forgery (XSRF) causes users to perform actions on web sites without their knowledge and allows attackers to steal cookies and harvest passwords.
- XSS and XSRF attacks are mitigated with input validation techniques.

Understanding Frameworks and Guides

- Frameworks are references that provide a foundation. Cybersecurity frameworks typically use a structure of basic concepts and provide guidance on how to implement security.
- Regulatory frameworks are based on relevant laws and regulations. A non-regulatory framework is not required by any law.
- Some frameworks are used within a single country (and referred to as national frameworks), while others are used internationally.
- Some frameworks only apply to certain industries. As an example, organizations that handle credit cards typically comply with the Payment Card Industry Data Security Standard (PCI DSS).
- Vendor-specific guides should be used when configuring specific systems.

Online References

- Remember, you have additional resources available online. Check them out at *http://gcgapremium.com/501-extras.*

Chapter 7 Practice Questions

1. Attackers have launched an attack using multiple systems against a single target. Which type of attack is this?
   A. DoS
   B. DDoS
   C. SYN flood
   D. Buffer overflow

2. An attacker has captured a database filled with hashes of randomly generated passwords. Which of the following attacks is MOST likely to crack the largest number of passwords in this database?
   A. Dictionary attack
   B. Birthday attack
   C. Brute force attack
   D. Rainbow tables

3. An application stores user passwords in a hashed format. Which of the following can decrease the likelihood that attackers can discover these passwords?
   A. Rainbow tables
   B. MD5
   C. Salt
   D. Input validation

4. An attacker has been analyzing encrypted data that he intercepted. He knows that the end of the data includes a template sent with all similar messages. He uses this knowledge to decrypt the message. Which of the following types of attacks BEST describes this attack?
   A. Known ciphertext
   B. Known plaintext
   C. Brute force
   D. Rainbow table

5. An attacker is attempting to write more data into a web application's memory than it can handle. Which type of attack is this?
   A. XSRF
   B. DLL injection
   C. Pass the hash

D. Buffer overflow

6. Management at your organization is planning to hire a development firm to create a sophisticated web application. One of their primary goals is to ensure that personnel involved with the project frequently collaborate with each other throughout the project. Which of the following is an appropriate model for this project?

A. Waterfall
B. SDLC
C. Agile
D. Secure DevOps

7. A web developer is adding input validation techniques to a web site application. Which of the following should the developer implement during this process?

A. Perform the validation on the server side.
B. Perform the validation on the client side.
C. Prevent boundary checks.
D. Implement pointer dereference techniques.

8. Developers have created an application that users can download and install on their computers. Management wants to provide users with a reliable method of verifying that the application has not been modified. Which of the following methods provides the BEST solution?

A. Code signing
B. Input validation
C. Code obfuscation
D. Stored procedures

9. Your organization is preparing to deploy a web-based application, which will accept user input. Which of the following will BEST test the reliability of this application to maintain availability and data integrity?

A. Model verification
B. Input validation
C. Error handling
D. Dynamic analysis

10. You are overseeing a large software development project. Ideally, developers will not add any unauthorized changes to the code. If they do, you want to ensure that it is easy to identify the developer who made the change. Which of the following provides the BEST solution for this need?

A. Agile SDLC
B. Version control
C. Secure DevOps
D. Static code analysis

11. Database administrators have created a database used by a web application. However, testing shows that the application is taking a significant amount of time accessing data within the database. Which of the following actions is MOST likely to improve the overall performance of a database?

A. Normalization
B. Client-side input validation
C. Server-side input validation
D. Obfuscation

12. Looking at logs for an online web application, you see that someone has entered the following phrase into several queries:

‘ or ‘1’=’1’ --

Which of the following is the MOST likely explanation for this?

A. A buffer overflow attack
B. An XSS attack
C. A SQL injection attack
D. A DLL injection attack

13. While creating a web application, a developer adds code to limit data provided by users. The code prevents users from entering special characters. Which of the following attacks will this code MOST likely prevent?

A. Man-in-the-browser
B. Amplification
C. XSS
D. Domain hijacking

14. Homer recently received an email thanking him for a purchase that he did not make. He asked an administrator about it and the administrator noticed a pop-up window, which included the following code:

```
<body  onload=”document.getElementByID(‘myform’).submit()”>
    <form  id=”myForm” action=”gcgapremium.com/purchase.php”
    method=”post”
```

<input name=”Buy Now” value=”Buy Now” />

</form>

</body>

Which of the following is the MOST likely explanation?

A. XSRF
B. Buffer overflow
C. SQL injection
D. Dead code

15. Your organization recently purchased a new hardware-based firewall. Administrators need to install it as part of a DMZ within the network. Which of the following references will provide them with the MOST appropriate instructions to install the firewall?

A. A regulatory framework
B. A non-regulatory framework
C. A general-purpose firewall guide
D. A vendor-specific guide

Chapter 7 Practice Question Answers

1. **B.** A distributed denial-of-service (DDoS) attack includes attacks from multiple systems with the goal of depleting the target's resources. A DoS attack comes from a single system and a SYN flood is an example of a DoS attack. A buffer overflow is a type of DoS attack that attempts to write data into an application's memory.

2. **D.** A rainbow table attack attempts to discover the password from the hash. However, they use rainbow tables, which are huge databases of precomputed hashes. A dictionary attack compares passwords against words in a dictionary of words, but a dictionary of words wouldn't include randomly generated passwords. A birthday attack relies on hash collisions. However, it wouldn't necessarily be effective depending on what hashing algorithm is used. A brute force attack attempts to guess all possible character combinations but is very time-consuming for each password.

3. **C.** A password salt is additional random characters added to a password before hashing the password, and it decreases the success of password attacks. Rainbow tables are used by attackers and contain precomputed hashes. Message Digest 5 (MD5) is a hashing algorithm that creates hashes, but the scenario already states that passwords are hashed. Input validation techniques verify data is valid before using it and they are unrelated to protecting hashed passwords.

4. **B.** This describes a known plaintext attack because the attacker knows some of the plaintext data used to create the encrypted data. More specifically, this is a chosen plaintext attack (but that wasn't available as an answer) because the attacker knew a portion of the plaintext. In a known ciphertext attack, the attacker doesn't have any information on the plaintext. A brute force attack attempts to guess a password. A rainbow table attack uses a table of hashes to identify a password from a matched hash.

5. **D.** One type of buffer overflow attack attempts to write more data into an application's memory than it can handle. None of the other answers are directly related to overloading the application's memory. A

cross-site request forgery (XSRF) attack attempts to launch attacks with HTML code. A Dynamic Link Library (DLL) injection attack injects a DLL into memory and causes it to run. A pass the hash attack attempts to discover a password.

6. **C.** The agile software development model is flexible, ensures that personnel interact with each other throughout a project, and is the best of the available choices. The waterfall model isn't as flexible and focuses instead on completing the project in stages. Both agile and waterfall are software development life cycle (SDLC) models, which is a generic concept designed to provide structure for software development projects. Secure DevOps is an agile-aligned development methodology that focuses on security considerations throughout a project.

7. **A.** Input validation should be performed on the server side. Client-side validation can be combined with server-side validation, but it can be bypassed, so it should not be used alone. Boundary or limit checks are an important part of input validation. Pointer dereference techniques use references to point to values and are unrelated to input validation techniques.

8. **A.** Code signing provides a digital signature for the code and verifies the publisher of the code and verifies that it hasn't been modified since the publisher released it. None of the other answers verify the application hasn't been modified. Input validation verifies data is valid before using it. Code obfuscation makes the code more difficult to read. Stored procedures are used with SQL databases and can be used for input validation.

9. **D.** Dynamic analysis techniques (such as fuzzing) can test the application's ability to maintain availability and data integrity for some scenarios. Fuzzing sends random data to an application to verify the random data doesn't crash the application or expose the system to a data breach. Model verification ensures that the software meets specifications and fulfills its intended purpose, but it doesn't focus on reliability or integrity. Input validation and error-handling techniques protect applications, but do not test them.

10. **B.** A version control system will track all changes to a system, including who made the change and when. Change management

processes (not available as a possible answer) typically provide the same solution. An agile software development life cycle (SDLC) model focuses on interaction from all players in a project, but doesn't necessarily include a version control system. Secure DevOps is an agile-aligned software development methodology that focuses on security throughout the process. Static code analysis examines the code without executing it as a method of code testing.

11. **A.** Normalization techniques organize tables and columns in a database and improve overall database performance. None of the other answers improve the database performance. Input validation techniques help prevent many types of attacks, and server-side input validation techniques are preferred over client-side techniques. Obfuscation techniques make the code more difficult to read.

12. **C.** Attackers use the phrase (' or'1'='1'--) in SQL injection attacks to query or modify databases. A buffer overflow attack sends more data or unexpected data to an application with the goal of accessing system memory. A cross-site scripting (XSS) attack attempts to insert HTML or JavaScript code into a web site or email. A Dynamic Link Library (DLL) injection attack attempts to inject DLLs into memory, causing DLL commands to run.

13. **C.** A cross-site scripting (XSS) attack can be blocked by using input validation techniques to filter special characters such as the < and > characters used in HTML code. None of the other listed attacks require the use of special characters. A man-in-the-browser attack exploits vulnerabilities in browsers to capture user data entries. An amplification attack increases the amount of data sent to a victim to overwhelm it. A domain hijacking attack changes the domain registration of a domain name without permission of the owner.

14. **A.** A cross-site request forgery (XSRF) attack causes users to perform actions without their knowledge. This scenario indicates the user visited a web site, most likely through a malicious link, and the link initiated a purchase. None of the other attacks cause unsuspecting users to make purchases. A buffer overflow attacks a web site and attempts to access system memory. A SQL injection attack attempts to access data on a database server. Dead code is code that never executes

and is unrelated to this scenario.

15. **D.** A vendor-specificguideforthenewhardware-basedfirewallwillhavethemostappropriate instructions for installing it. Frameworks (regulatory or non-regulatory) provide structures that can be followed for different purposes, but they wouldn’t be available for a specific firewall. A general-purpose guide will provide general instructions, but not instructions for a specific vendor’s firewall.