10.3.14 Web Application Attack Facts

People today connect, learn, shop, provide services and information, and do business over the internet. All of this is made possible through web browsers and web applications. There are literally thousands of applications that are used in our everyday lives. With so many options, there are many ways that attackers have found to exploit them.

This lesson covers the following topics:

- Privilege escalation
- Pointer/object dereference
- Buffer overflows
- Resource exhaustion
- Memory leaks
- Race conditions
- Error handling
- Improper input handling
- Replay attacks
- Pass the hash
- Application programming interface (API) attacks
- Secure Sockets Layer (SSL) stripping
- Driver manipulation

Privilege Escalation

Most attacks are some form of privilege escalation. There are two types:

- Horizontal
  - This is when an attacker gains data that belongs to another user with the same privilege level as themselves (like a co-worker).
- Vertical
  - This is when an attacker uses a system vulnerabilities to escalate privileges to gain administrative access.

Pointer/Object Dereference

Dereferencing a pointer is retrieving the value stored in memory.

Some important facts:

- A pointer stores a memory address
- All operating systems embed the kernel in the user's workspace
  - The kernel is the operating system's core program that controls everything in the system
- Page protections protect the kernel from user access but can be exploited through by a DoS attack through a NULL pointer dereference
- If a DMA driver module doesn't have enough enough security protections in place, it can release user pages that are pinned to a pointer with a NULL value. This happens when:
  - An app dereferences an object that comes back NULL instead of valid
  - Null is exploited as a constant built in to evaluate to 0 in the C language

- An x86 system has a valid 0 address in the kernel address space

Buffer Overflows

Buffer overflow important facts:

- A buffer is a temporary data storage area with limited space
- Overflows occur when more data is attempting to be stored than the program was written for
- Can allow hackers to cause data to flow to other memory areas that may not be protected
  - Attackers may now access database files or system files and can replace executable code with malicious code. This is called arbitrary code execution.
- Can cause DoS attacks by crashing the program
  - Can occur in routers, IoT devices, and firewalls

Resource Exhaustion

Resource exhaustion is a form of attack that focuses on depleting the resources of a network to create a denial of service to legitimate users.

This attack can be done through:

- Slow header attacks:
  - Send HTTP headers so slowly it prevents other users from accessing the site
  - Can be prevented with HTTP header timeouts
- Slow post attacks:
  - Send HTTP POST body very slowly. This is done through forms, logins, and feedback input fields
  - Can be prevented by setting a maximum body size for each form and setting the web server setting with a max total transfer time

Resource exhaustion attacks can be focused on memory, file system storage, database-connection pool entries, or the CPU. When allocation of these resources is requested but the size of the resource or number is not controlled, a denial of service results from lack of resources.

Memory Leak

A memory leak happens when dynamic memory is allocated in a program, but no pointers are connected to it. This causes it to never be returned when requested.

- Programmers often create temporary memory allocations. This becomes a problem when they are not deleted after use.
- Whether unintentionally leftover from a project or intentionally created by an attacker, memory leaks can result in:
  - Resource exhaustion
  - DoS
  - Exploitation of other areas affected by low-memory conditions

To mitigate these attacks:

- Delete unneeded memory allocations when finished with a project

- Ensure that pointers are properly connected to memory values

Race Conditions

Another web application vulnerability is a time-of-check to time-of-use bug, or (TOCTTOU) bug. This happens when a system is programmed to run with certain processes dependent on a sequence of events or race conditions.

- Can happen when an attacker schedules an execution of operation between a time of check and a time of use and forces the user's process to pause or send an error. For example, in the moment between authenticating to a system and utilizing the system, the attacker can jump into the process and act as the authenticated user, leading to privilege escalation.
  - To mitigate:
    - Ensure your operating system's file system state is not allowed to change between two system calls
    - Use file system calls that run on file handles instead of file names when possible
    - Lock single files before the check

Error Handling

Improper error handling can create vulnerabilities in a system by revealing information that attackers can use to exploit the system. This display of too much information can result from coding practices that are not in alignment with security policies. Some examples are:

- An attacker may use a SQL injection attack that fails initially. But the error message discloses the malformed query, which could show the query logic or other sensitive data, like passwords. The attacker can use the new information from the error message to gain access to the system.
- The disclosure of the full pathname in an error message that is generated from a path-transversal weakness exploit attempt.

To mitigate, be sure to program the error message with minimal information that's only useful to the intended audience

Improper Input Handling

Improper input handling refers to lack of validation, sanitization, filtering, decoding input data, or encoding input data.

Processing of untrustworthy input data can lead to:

- Buffer overflows
- XSS
- Directory transversal
- NULL byte injections
- SQL injection
- Uncontrolled format string
- DoS
- OS commanding

To mitigate:

- Set specific parameters for acceptable data forms and types
- Accurately define data restrictions
- Sanitizing, validate, and filter properly

Replay Attack

Replay attacks happen when network traffic is intercepted by an unauthorized person who then delays or replays the communication to its original receiver, acting as the original sender. The original sender is unaware of this occurrence.

- Also known as session replay attacks
- They are a type of man-in-the-middle attack

To mitigate, implement:

- Strong digital signatures with timestamps
- Session keys that are time-bound and process-bound
- Sequence numbers
- Program authentication systems to accept network packets that have valid timestamps and sequence numbers

Pass the Hash

Pass the hash is so dangerous to an organization because once an attacker gains access, the whole organization can be compromised very quickly.

How it works:

1. An attacker gains access to an individual computer through malware or other techniques
2. The attacker accesses the memory in the workstation to find stored hashes of other users that have used the workstation
3. The attacker uses the stored hashes to gain access to other workstations in search of a station that grants privilege escalation

To mitigate:

- Use direct networking to prevent standard users and local admin users from having access to other user's workstations
- Use Group Policy Object (GPO) Editor to disable Remote Desktop Connections in an Active Directory network
- Limit domain admins access to only workstations with the same level of privileges
- Create separate standard user level accounts for admins to use when accessing lower-level privilege machines

Application Programming Interface (API) Attacks

Application programming interfaces (APIs) are the way businesses transfer information between systems within their organization or how a business communicates information to another organization. This is also a means of information transfer between companies and their customers as APIs are the way most applications communicate with websites.

Many APIs are openly published to promote customer usage and make interactions easy. But they do create an opportunity for a malicious user to exploit the interface to gain access to internal data and infrastructure. For example, an e-commerce site may use its API for product catalog pages on their website, in their mobile app, for a third-party reseller, and for search engine bots that bring customers to their website.

To mitigate potential API problems:

- Implement rate limiting. This limits the number of calls from a client within a time limit
- Use security logs to detect and analyze unauthorized access attempts
- Look for SQL injections. These happen when a SQL statement is entered in a data field and gets executed in the database
- Make sure that program notifications are sent when there is an excess of error messages

Secure Sockets Layer (SSL) Stripping

SSL stripping is an attack that focuses on stripping the security from HTTPS-enabled websites. This is how it works:

1. An attacker intercepts the initial request a user sends to a website.
2. The attacker establishes a secure connection with the intended server and an unsecure HTTP connection with the user where all communication goes through him or her.
3. The attacker can intercept the initial request when it comes through a 302 redirect or through a non-SSL site that provides a link to a proxy that looks like the intended site.

To mitigate:

- Encrypt all elements of your site with an SSL certificate.
- Add your domain to the HSTS preload list. This lets browsers know that your site is secure.

Driver Manipulation

A device driver is a small piece of software that provides an interface between the operating system and a hardware device such as a printer, keyboard, or network card. Attackers can manipulate a driver by adding malicious logic. Driver manipulation attacks often happen as a result of a web application attack such as a drive-by download or through social engineering or phishing. The goal is to replace a good driver with one that is malevolent or to add software that comes between a good driver and the operating system.

Common driver manipulation attacks include:

Attack	Description
Refactoring	▪ Software or code refactoring is usually considered a beneficial practice. The external behavior of refactored software code does not change. Internally, the code is modified to improve readability, reduce complexity, or improve efficiency. ▪ Attackers refactor device drivers so that their external behavior does not change. The printer, keyboard, network card, or hardware controlled by the driver still function properly. This makes it hard to detect any problems. Internally, the refactored driver now has hidden functions that benefit the attacker.

Shimming	▪ Like refactoring, shimming is usually beneficial. As operating systems and other software libraries are updated, their application programming interface (API) may change. The API specifies how other programs should interact with the software library or operating system. If the API is updated with new specification, other programs using older API specifications may not work. To remedy this, a shim can be used. A shim is software that is placed between the newer API and software that conforms to the older API. The shim intercepts calls to the older API, translate them, and pass them to the newer API. In some cases, they can redirect the API calls elsewhere to complete the expected operation called for in the older API. ▪ Attackers can modify existing shims by injecting malicious code. They can also create a shim that intercepts valid API calls. However, the shim executes malicious code before it passes the valid calls through to the API.

To mitigate:

- Use the latest browser version and patch level.
- Verify that the operating system is at the latest patch level.
- Install antivirus, anti-spyware, pop-up blocking, and firewall software.
- Use input validation when programming services.
  - Client-side validation should first be used on the local system to identify input errors before the data is ever sent to the server.
    - For example, if the user enters an invalid value in an email address field, the error can be detected before the data is submitted.
  - Server-side validation should be used for error detection after the data is sent to the server. Experienced attackers can circumvent client-side validation techniques to send malicious information to the server.
    - For example, an attacker could send data to the server from outside the application's standard user interface, bypassing any input validation measures that may have been implemented on the client. It is unwise to rely solely on client-side input validation techniques.
- Implement DNS Security Extensions, or DNSSEC. This is a security measure that only allows connection to your computer from servers that have previously been given a digital certificate.
- Use HTTPS. This transfer protocol encrypts the HTTP over Transport Layer Security (TLS) or over Secure Socket Layer (SSL), protecting your browser against threats.
- Use add-ons to increase the security of browsing activities:
  - NoScript blocks all active content except from sites you trust.
  - Adblock Plus blocks advertisements and ad banners (which could contain malicious code) on the internet.
- Train users to log out of websites when finished. Users should never allow applications to remember their authentication information.