8.4.7 Web Application Attack Facts

Common web application attacks include:

Attack	Description
Drive-By Download	*A drive-by download* is an attack where software or malware is downloaded and installed without explicit consent from the user. It occurs when a user is redirected to a malicious website. The website downloads malicious software or executes client-side scripts in the user's browser. Drive-by downloads can occur in a few different ways: ▪ Through social engineering, the user is tricked into downloading the software. The user might not realize that clicking a link will install software. Or, the user might know that something is being installed, but does not fully understand what it is or what it does. ▪ By exploiting a browser or operating system bug, a site is able to install software without the user's knowledge or consent.
Typosquatting/URL Hijacking	*Typosquatting* occurs when an attacker registers domain names that correlate to common typographical errors made by users when trying to access a legitimate website. This exploit is also called *URL hijacking*. Common strategies include the following: ▪ Using domain names that are based on common typing errors made when entering the legitimate site's domain name (for example, amazno.com instead of amazon.com). ▪ Using domain names that are slightly different from the legitimate site's domain name (for example, blackhorses.com instead of blackhorse.com). ▪ Using a top-level domain that is different from the legitimate site's domain (for example, whitehouse.com instead of whitehouse.gov). The typosquatter's intentions may be benign or malicious in nature. They may be simply trying to coerce the legitimate site owner to buy the domain name from them. Alternatively, they may be attempting to trick unsuspecting users by redirecting them to a phishing site that looks like the legitimate website; they may even use this exploit to install drive-by malware.
Buffer Overflow	A *buffer overflow* occurs when the operating system or an application does not properly enforce boundaries for how much and what type of data can be inputted. This happens when a server or application receives more data input than it is expecting. This can overwrite the return address within a program to execute malicious code. Hackers submit data beyond the size reserved for the data in the memory buffer, and the extra data overwrites adjacent memory locations. The extra data sent by the attacker could control and exploit the web server, including executable code that might be able to execute in privileged mode. Buffer overflow is a common attack on web servers, and it is one of the most common exploits of all internet exposed network services. Countermeasures for buffer overflow attacks include: ▪ Apply all security patches to workstations. ▪ Limit user input to less than the size of the buffer. ▪ Validate input by looking for certain symbols that may be program instructions. ▪ Implement strict coding standards to eliminate the potential for weaknesses.
Integer Overflow	An *integer overflow* occurs when a computational operation by a running process results in a numeric value that exceeds the maximum size of the integer type used to store it in memory. When this occurs, the value will "wrap around" and start again at its minimum value, in much the same way a mechanical odometer in a car rolls over to zero when it exceeds the maximum number of miles it can record. An integer overflow condition can allow an attacker to manipulate the value of variables, leading to unintended behavior by the system. For example, when purchasing items from an online store, an integer overflow attack could be used to convert the total due from a positive value to a negative value, refunding money to the customer's credit card when the transaction is executed.
Cross-Site Scripting (XSS)	*Cross-site scripting* (XSS) is an attack that injects scripts into web pages. When the user views the web page, the malicious scripts run, allowing the attacker to capture information or perform other actions. This can be considered a form of drive-by download because malware is downloaded to the client machine, or malicious client-side scripts from suspect internet sites are executed on the client machine. This attack is called cross-site scripting because, during an attack, everything can look fine to the user. The user may be redirected to the correct site, while a malicious script runs in the background. ▪ XSS often relies on social engineering or phishing to entice users to click on links to web pages that contain the malicious scripts. ▪ Some scripts redirect users to legitimate websites but run the script in the background to capture information sent to the legitimate site. ▪ Scripts can be written to read (steal) cookies that contain identity information, such as session information.

	▪ Scripts can also be designed to run under the security context of the current user. For example, scripts might execute with full privileges on the local system, or the scripts might run using the credentials used on a financial website.
Cross-Site Request Forgery (CSRF/XSRF)	*Cross-site Request Forgery* (CSRF/XSRF) is also known as a one-click attack or session riding. CSRF is a type of malicious exploit whereby unauthorized commands are transmitted from the user to a website that currently trusts the user, by way of authentication, cookies, etc. This is almost the opposite of an XSS attack, except CSRF exploits the trust that a site has in a user's browser.
LDAP Injection	An *LDAP injection* attack uses LDAP statements with arbitrary commands to exploit web-based applications with access to a directory service. Arbitrary commands could grant permissions to unauthorized queries, or modify content inside the directory service tree. If the application fails to properly validate the input in the form fields, then it is possible for the wrong commands to be constructed. This could potentially provide an attacker with unauthorized access to the LDAP directory tree. They may be able to view and even modify information that they should not have access to.
XML Injection	An *XML injection* attack uses malicious content and/or structures in an XML message to alter the intended logic of the application.
Command Injection	A *command injection* attack injects and executes unwanted commands on the application. The commands are executed with the same privileges and environment that are granted to the application. ▪ A *code injection* attack extends the default functionality of application, without executing system commands. ▪ An *OS command injection* attack executes system level commands through a vulnerable application. In an *arbitrary code execution* exploit, a vulnerability in a running process allows an attacker to inject malicious code and execute it. Because it can be carried out from a remote computer, this attack is sometimes called a *remote code execution* exploit. The exploit is typically executed by manipulating the instruction pointer of the vulnerable process, which tells the CPU which instruction to run next. Malicious code is injected by the attacker, usually masquerading as simple input data for the running process. This redirects the instruction pointer to execute the attacker's injected code, allowing the attacker to take control of the process.
SQL Injection	An *SQL injection* attack occurs when an attacker includes database commands within user data input fields on a form, and those commands subsequently execute on the server. The injection attack succeeds if the server does not properly restrict entry of characters that could trigger a database command.
Pointer Dereference	A pointer is a programming feature that references another location in a computer's memory. Potentially, programs can dereference a null pointer. While this can cause reliability problems, an attacker could intentionally trigger the pointer dereference. The attacker could then use the exception to bypass security logic or cause the program to reveal debugging information. This information may give the attacker leverage in subsequent attacks.
DLL Injection	A *DLL injection* attack occurs when a program is forced to load a dynamic-link library (DLL). The DLL then executes malicious code under the security context of the running application.
Directory Traversal	A *directory traversal* (also known as *path traversal*) attack uses specific characters to access the parent directory in a file system. In this case, characters representing the path to the parent directory on the application server could be manipulated to gain access to a restricted file. This attack exploits insufficient security validation/sanitization of user-supplied input file names.
Header Manipulation	*Header manipulation* is the process of including invalid data in an HTTP response header. The header of the response usually contains some type of status message. Header manipulation is the method used by threat agents to conduct additional exploits. Two things usually happen when a header manipulation attack occurs. First, the data enters a web application through an untrusted source, and the data is included in an unvalidated HTTP response header that is sent back to the user. Second, the attacker passes malicious data to a vulnerable application, and the application includes that data in the HTTP response header. ▪ *HTTP response splitting* is where a single HTTP request is interpreted by the target to be two separate responses, which gives control to subsequent headers and the response body to the threat agent. The browser interprets this HTTP response as two separate HTTP responses instead of one. This type of vulnerability can be exploited to perform several different types of web application attacks. ▪ *DNS cache poisoning* can occur with header manipulation. To carry out a cache poisoning attack, the attacker first finds the vulnerability in the web application that allows them to fill the HTTP header response field with multiple headers. Then it forces the proxy cache server to flush its real cache content and replace it with the new, poisoned cache content from the attack. The only way to fix it is to flush the cache, either on the proxy cache server or on the local cache. ▪ *Cross-user defacement* is when a credible website may look defaced to a particular user, or it allows a threat agent to steal login information by forging a fake login screen for the website. Using response splitting, this attack is a

	temporary form of defacement. The multiple headers within the malformed HTTP header can make a website appear to be temporarily defaced when the user accesses it. ▪ *Cookie manipulation* can fraudulently authenticate threat agents to a website by injecting a custom HTTP header, planting cookies that should not be there, or by injecting a META tag to the browser.
Zero-Day	A *zero-day* attack (also known as a zero-hour or day zero attack) is an attack that exploits computer application vulnerabilities before they are known and patched by the application's developer. There are two primary vehicles used to conduct these types of attacks: a web browser and an email with a link that, when clicked, launches the attack. Using a quality assurance process, such as the Unknown Vulnerability Management Process, helps to identify and correct potential weaknesses in the application before it is released. Steps in the process are: 1. Analyze. Developers try to break the application themselves to find its vulnerabilities, looking for weaknesses and attack vectors. 2. Test. Fuzz testing is conducted against the attack vectors identified during the analysis phase. 3. Report. Personnel try to reproduce any issues found during the testing phase to make sure that the situation is replicable and represents a real vulnerability. 4. Mitigate. A fix is developed for the vulnerability, based on the results of the testing and reporting phases. A zero-day attack occurs during the window between the time when the vulnerability is first exploited by the attacker and when the software developer creates a fix for it. The window may vary between a couple of days to many years.
Client-Side	A *client-side* attack exploits vulnerabilities in client applications that interact with a malicious server. These attacks focus on the client, as opposed to the server. A typical example is a malicious web page targeting a specific browser vulnerability that would give the malicious server complete control of the client system. JavaScript is an example of client-side scripting, where the client system runs the scripts that are embedded in web pages. When pages download, the scripts are executed. Exploits can also utilize locally shared objects (LSOs). LSOs are also referred to as flash cookies. Adobe Flash uses LSOs to save data locally on a computer, such as information for a Flash game being played or user preferences. However, LSOs can also be used to collect information about the user's browsing habits without their permission. The Flash Player Settings Manager can be used to configure Flash to prevent LSOs from being saved on the local computer.

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 functions 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 will intercept 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 will intercept valid API calls. However, the shim will execute malicious code before it passes the valid calls through to the API.

Mitigation practices to protect internet-based activities from web application attacks include:

- Using the latest browser version and patch level.
- Verifying that the operating system is at the latest patch level.
- Installing antivirus, anti-spyware, pop-up blocking, and firewall software.
- Using 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.
- Implementing DNSSEC, or DNS Security Extensions. This is a security measure that only allows connection to your computer from servers that have previously been given a digital certificate.

- Using HTTPS. This transfer protocol encrypts the HTTP over Transport Layer Security (TLS) or over Secure Socket Layer (SSL), protecting your browser against threats.
- Using 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.
- Training users to log out of websites when finished. Users should never allow applications to remember their authentication information.
