Week 07

The CWE/SANS Top 25 Most Dangerous Software Errors list provides insights into common programming errors leading to cybersecurity vulnerabilities. These errors are categorized into three groups:

1. Insecure interaction between components

2. Risky resource management

3. Porous defences

Critical **WEB APPLICATION SECURITY FLAWS** include five related to insecure software code.

* + 1. Handling Program handling - Unvalidated input
    2. Buffer overflow
    3. Injection flaws
    4. Cross-site scripting
* Improper error handling

Flaws occur because of insufficient checking and validation of data and error codes in programs.

**SQL INJECTION ATTACKS (SQLI) -** prevalent and dangerous network-based security threats.

An SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application.

Most common attack goal is bulk extraction of data.

A successful SQL injection exploit can:

* Read sensitive data from the database.
* Modify database data (Insert/Update/Delete)
* Execute administration operations on the database (such as shutdown the DBMS)
* Recover the content of a given file present on the DBMS file system.
* And in some cases, issue commands to the operating system.

**INJECTION TECHNIQUE: SQLI**

* Terminating a text string and appending a new command
* The inserted command may have additional strings appended to it before it is executed the attacker terminates the injected string with a comment mark “- -”
* Subsequent text is ignored at execution time.

In SQL injection (sqli) attacks, attackers exploit vulnerabilities in web applications that improperly handle user input or server variables, such as HTTP and network headers.

**TYPE OF SQL INJECTIONS**

**TAUTOLOGY** Injects code in one or more conditional statements so that they always evaluate to true.

**END-OF-LINE COMMENT** After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments.

**PIGGYBACKED QUERIES** the attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request.

**INFERENTIAL ATTACK** There is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behaviour of the Website/database server.

**SQLI ATTACK EXAMPLE: LOGIN AUTHENTICATION QUERY**

Standard query to authenticate users.

Special strings can be entered by attacker.

**XML EXTERNAL ENTITY PROCESSING**

- XML External Entity (XXE) processing involves exploiting vulnerabilities in XML parsers.

- XML is commonly used to structure data exchanged between clients and servers.

- XXE attacks occur when the server accepts unvalidated data from the client, allowing malicious XML entities to be injected.

- Attackers can tamper with existing XML pages to execute arbitrary commands or access sensitive information.

- This type of attack poses a significant security risk and can lead to unauthorized access or data leakage.

**COMMAND INJECTION ATTACK**

* Similar to an XML external entity injection attack.
* Takes a value from the user and passes it to a system function or an evaluate function.
* Focus on the operating system - Pass the parameters to the operating system to handle.
* **Consequences**: If there is no input validation, you can execute any operating system command into the input field.

**CROSS SITE SCRIPTING (XSS) ATTACKS**

One that uses the web server to attack the client side.

Injects a code fragment from a scripting language into an input feld to have that code executed within the browser of a user visiting a site.

Three types of cross-site scripting attack –

**PERSISTENT CROSS-SITE SCRIPTING**. Stored on the server and displayed for any user visiting a page.

**REFLECTED CROSS-SITE SCRIPTING**.: The script isn’t stored. Instead, it is included in a URL as a parameter you would send to a victim.

**DOM-BASED XSS ATTACK**: Document Object Model (DOM)-based XSS attack allow us to call for objects through the scrip which should result in the object being executed.

**FILE TRAVERSAL**

A way to get out of what the web server wanted you to originally see and be able to see more.

File Traversal is the ability to browse the web server and see files outside the contents of **/var/www/html,** for example root folder of the web server.

**BUFFER OVERFLOW ATTACK**

* Common attack mechanism.
* Dominant method of exploiting software vulnerabilities.
* Major concern in cybersecurity.
* These attacks often exploit buggy code found in widely used operating systems and applications. The persistence of buffer overflow vulnerabilities is attributed to continued careless programming practices by developers.
* Buffer overflow occurs when a process tries to store data beyond the capacity of a fixed-sized buffer.
* This error can lead to overwriting adjacent memory locations, potentially affecting program variables, parameters, or control flow data.
* Buffers can exist on the stack, heap, or data section of a process.
* Consequences include corrupted program data, unexpected control flow changes, memory access violations, and execution of attacker-chosen code.

**BUFFER OVERFLOW IN THE STACK:**

- Affects local variables, function arguments, and return addresses.

- Corrupts memory on the stack.

**BUFFER OVERFLOW IN THE HEAP:**

- Corrupts memory located on the heap, typically above program code.

- Memory is dynamically allocated for data structures like linked lists.

- Affects global variables and other program data.

**BUFFER OVERFLOW IN GLOBAL (OR STATIC) DATA AREA:**

- Involves buffers located in the program's global data area.

- Loaded from the program file and located above the program code in memory.

The ultimate goal for the attacker is getting a shell that allows to execute arbitrary commands with high privileges.

**INPUT SIZE & BUFFER OVERFLOW**

- Programmers may assume a maximum expected size of input.

- Allocated buffer size is not always confirmed.

- Results from exceeding allocated buffer size.

- Testing may not identify the vulnerability if inputs don't trigger the overflow.

- Treat all input as potentially dangerous to prevent vulnerabilities.

**WRITING SAFE PROGRAM CODE - Security issues -** Correct algorithm implementation; Correct machine instructions for algorithm; Valid manipulation of data

Issue of dynamic memory allocation

Unknown amounts of data

Allocated when needed, released when done.

Used to manipulate Memory leak.

Steady reduction in memory available on the heap to the point where it is completely exhausted.

**SQLI COUNTERMEASURES AND PREVENTION**

Three Types

**Defensive coding** Manual defensive coding practices; Parameterised query insertion

**Detection** Signature based Anomaly based Code analysis.

**Run-time prevention** Check queries at runtime to see if they conform to a model of expected queries.

**COUNTERMEASURES AND PREVENTION**

**CODE INJECTION ATTACK -** There are several defences available to prevent this type of attack.

Block assignment of form field values to global variables. Rather, they are saved in an array and must explicitly be retrieved by name.

Another defence is to only use constant values in include (and require) commands.

This ensures that the included code does indeed originate from the specified files.

If a variable has to be used, then great care must be taken to validate its value immediately before it is used.

**XSS ATTACK** To prevent this attack:

any user-supplied input should be examined, and any dangerous code removed or escaped to block its execution.

Buffer overflow defences

* widely exploited.

Two broad defence approaches

* Compile-time - Aim to harden programs to resist attacks in new programs.
* Run-time - Aim to detect and abort attacks in existing programs.

**SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC):**

  - Involves fundamental tasks: - Understanding the system requirements.  - Analyzing requirements in detail. - Determining suitable technology based on the system's purpose - Identifying and designing program functions. - Coding the programs. - Testing the programs individually and collectively. - Installing the system into a secure "production" environment.

**PHASES OF SDLC**

* Phase zero (project inception)
* System requirements
* System design
* Development
* Test
* Deployment

The earlier in the development life cycle security is implemented, the cheaper software development will be.

**SECURE SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC):**

Requirements:

- Mapping security and privacy requirements is essential.

- Business system analysis should consider organizational security policies and standards.

Design:

- Conducting threat modelling to assess the technical security posture of the application.

- Performing design reviews by security experts to iterate and improve security.

Development:

- Addressing development-related vulnerabilities:

Utilizing static analysis tools to automate finding issues in source code.

Implementing peer reviews where developers review each other's code.

Testing:

- Conducting critical testing to discover vulnerabilities not found earlier.

- Developing security test cases and performing dynamic analysis.

- Operating software in a test environment to assess its security.

Deployment:

- Conducting a final security review before deployment.

- Establishing application security monitoring and response plans.

- Providing security training to relevant personnel.