UNIT IV
Security of Applications

Improper Data/Input Validation

- **Input validation**, also known as data validation, is the proper testing of any input supplied by a user or application. Input validation prevents improperly formed data from entering an information system. Because it is difficult to detect a malicious user who is trying to attack software, applications should check and validate all input entered into a system.

- Input validation should occur when data is received from an external party, especially if the data is from untrusted sources. Incorrect input validation can lead to injection attacks, memory leakage, and compromised systems. While input validation can be either whitelisted or blacklisted, it is preferable to whitelist data.

- Whitelisting only passes expected data. In contrast, blacklisting relies on programmers predicting all unexpected data. As a result, programs make mistakes more easily with blacklisting.

What is input validation attack?

- An input validation attack occurs when an attacker deliberately enters malicious input with the intention of confusing an application and causing it to carry out some unplanned action.
- Malicious input can include code, scripts and commands, which if not validated correctly can be used to exploit vulnerabilities.
- The most common input validation attacks include Buffer Overflow, XSS attacks and SQL injection.

Improper input validation

- Improper input validation or unchecked user input is a type of vulnerability in computer software and application that may be used for security exploits.

- When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application/software. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

LIST OF VULNERABILITIES

- Injection: SQL Injection, Blind SQL Injection, nosql Injection, XXE Injection ,OS Command Injection, XPath Injection, HTTP Injection, HTML Injection, Null Injection, Server Side Include (SSI) Injection, ORM Injection, Format String Injection, LDAP Injection, Flash Injection, XML Injection, IMAP/SMTP Injection, Formula Injection, Code Injection, Resource Injection
- Cross-site scripting: Stored Cross-site scripting, Reflected Cross-site , scripting, DOM Cross-site scripting
- Cross Zone Scripting
- Directory/Path Traversal
- Overflow: Heap Overflow, Stack Overflow, Memory Leak

Authentication and Authorization Attacks

- Authentication and authorization might sound similar, but they are distinct security processes in the world of identity and access management (IAM). Authentication confirms that users are who they say they are. Authorization gives those users permission to access a resource.

Attack types	Attack description
• Brute Force	Allows an attacker to guess a person's user name, password, credit card number, or cryptographic key by using an automated process of trial and error.
• Insufficient Authentication	Allows an attacker to access a web site containing sensitive content or functions without having to properly authenticate with the web site.
• Weak Password Recovery Validation	Allows an attacker to access a web site that provides them with the ability to illegally obtain, change, or recover another user's password.

Security Misconfiguration

- Security misconfiguration is the implementation of improper security controls, such as for servers or application configurations, network devices, etc. that may lead to security vulnerabilities.

For example, insecure configuration of web applications could lead to numerous security flaws including:

- Incorrect folder permissions
- Default passwords or username
- Setup/Configuration pages enabled
- Debugging enabled

- A security misconfiguration could range from forgetting to disable default platform functionality that could grant access to unauthorized users such as an attacker to failing to establish a security header on a web server.

- Security misconfiguration can happen at any level of an application, including the web server, database, application server, platform, custom code, and framework.

- The impact of a security misconfiguration in your web application can be far reaching and devastating. According to Microsoft, cyber security breaches can now globally cost up to $500 billion per year, with an average breach costing a business $3.8 million.

Security Misconfiguration Examples

- To give you a better understanding of potential security misconfigurations in your web application, here are some of the best examples:

Example #1: Default Configuration Has Not Been Modified / Updated

- If you have not changed the configuration of your web application, an attacker might discover the standard admin page on your server and log in using the default credentials and perform malicious actions.

Example #2: Directory Listing is Not Disabled on Your Server

- In such cases, if an attacker discovers your directory listing, they can find any file. Hackers can find and download all your compiled Java classes, which they can reverse engineer to get your custom code. They can then exploit this security control flaw in your application and carry out malicious attacks.

Example #3: Insecure Server Configuration Can Lead Back to the Users, Exposing Their Personal Information

- Applications with security misconfigurations often display sensitive information in error messages that could lead back to the users. This could allow attackers to compromise the sensitive data of your users and gain access to their accounts or personal information.

Example #4: Sample Applications Are Not Removed From the Production Server of the Application

- Many times these sample applications have security vulnerabilities that an attacker might exploit to access your server.

Example #5: Default Configuration of Operating System (OS)

- The default configuration of most operating systems is focused on functionality, communications, and usability. If you have not updated or modified the default configuration of your OS, it might lead to insecure servers.

Information Disclosure

Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker, including:

- Data about other users, such as usernames or financial information
- Sensitive commercial or business data
- Technical details about the website and its infrastructure

Some basic examples of information disclosure are as follows:

- Revealing the names of hidden directories, their structure, and their contents.
- Providing access to source code files via temporary backups
- Explicitly mentioning database table or column names in error messages
- Unnecessarily exposing highly sensitive information, such as credit card details
- Hard-coding API keys, IP addresses, database credentials, and so on in the source code
- Hinting at the existence or absence of resources, usernames, and so on via subtle differences in application behavior

How do information disclosure vulnerabilities arise?

Information disclosure vulnerabilities can arise in countless different ways, but these can broadly be categorized as follows:

- **Failure to remove internal content from public content**. For example, developer comments in markup are sometimes visible to users in the production environment.
- **Insecure configuration of the website and related technologies**. For example, failing to disable debugging and diagnostic features can sometimes provide attackers with useful tools to help them obtain sensitive information. Default configurations can also leave websites vulnerable, for example, by displaying overly verbose error messages.
- **Flawed design and behavior of the application**. For example, if a website returns distinct responses when different error states occur, this can also allow attackers to enumerate sensitive data, such as valid user credentials.

The impact of information disclosure vulnerabilities

- Information disclosure vulnerabilities can have both a direct and indirect impact depending on the purpose of the website and, therefore, what information an attacker is able to obtain.
- In some cases, the act of disclosing sensitive information alone can have a high impact on the affected parties. For example, an online shop leaking its customers' credit card details is likely to have severe consequences.
- On the other hand, leaking technical information, such as the directory structure or which third-party frameworks are being used, may have little to no direct impact. However, in the wrong hands, this could be the key information required to construct any number of other exploits. The severity in this case depends on what the attacker is able to do with this information.

Prevent information disclosure vulnerabilities

Preventing information disclosure completely is tricky due to the huge variety of ways in which it can occur. However, there are some general best practices that you can follow to minimize the risk of these kinds of vulnerability creeping into your own websites.

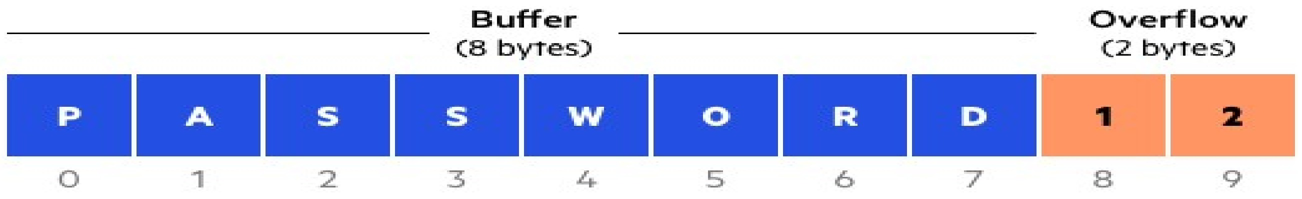
- Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive.
- Audit any code for potential information disclosure as part of your QA or build processes.
- Double-check that any debugging or diagnostic features are disabled in the production environment.
- Make sure you fully understand the configuration settings, and security implications, of any third-party technology that you implement.

Buffer Overflow Issues

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another.

A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.

For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary.



- Buffer overflows can affect all types of software. They typically result from malformed inputs or failure to allocate enough space for the buffer. If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes.

- **Buffer Overflow Attack:** Attackers exploit buffer overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT systems.

- If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite a pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

- Certain coding languages are more susceptible to buffer overflow than others. C and C++ are two popular languages with high vulnerability, since they contain no built-in protections against accessing or overwriting data in their memory. Windows, Mac OSX, and Linux all contain code written in one or both of these languages.
- More modern languages like Java, PERL, and C# have built-in features that help reduce the chances of buffer overflow, but cannot prevent it altogether.

Types of Buffer Overflow Attacks

- **Stack overflow attack** - This is the most common type of buffer overflow attack and involves overflowing a buffer on the call stack*.
- **Heap overflow attack** - This type of attack targets data in the open memory pool known as the heap*.
- **Integer overflow attack** - In an integer overflow, an arithmetic operation results in an integer (whole number) that is too large for the integer type meant to store it; this can result in a buffer overflow.
- **Unicode overflow** - A unicode overflow creates a buffer overflow by inserting unicode characters into an input that expect ASCII characters.

How to protect against buffer overflow attacks

- Luckily, modern operating systems have runtime protections which help mitigate buffer overflow attacks. Let’s explore 2 common protections that help mitigate the risk of exploitation:

- **Address space randomization** - Randomly rearranges the address space locations of key data areas of a process. Buffer overflow attacks generally rely on knowing the exact location of important executable code, randomization of address spaces makes that nearly impossible.
- **Data execution prevention** - Marks certain areas of memory either executable or non-executable, preventing an exploit from running code found in a non-executable area.

Broken authentication and session management

- Broken Authentication and Session Management as: **‘Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities.’** In other words, an attacker can get unauthorized access to a user's data due to flaws in the implementation.
- Broken Authentication and Session Management could lead to exposed user data, such as credentials or critical private data. It could also allow for privilege escalation attacks.

Broken authentication and session management

These types of weaknesses can allow an attacker to either capture or bypass the authentication methods that are used by a web application.

- User authentication credentials are not protected when stored.
- Predictable login credentials.
- Session IDs are exposed in the URL (e.g., URL rewriting).
- Session IDs are vulnerable to session fixation attacks.
- Session value does not timeout or does not get invalidated after logout.
- Session IDs are not rotated after successful login.
- Passwords, session IDs, and other credentials are sent over unencrypted connections.

The goal of an attack is to take over one or more accounts and for the attacker to get the same privileges as the attacked user.

Examples

Example #1: URL rewriting

A travel reservations application supports URL rewriting, putting session IDs in the URL.

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

Example #2: Application's timeout is not set properly

The user utilizes a public computer to access a site. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and that browser is still authenticated.

Example #3: Passwords are not properly hashed and salted
An insider or external attacker gains access to the system’s password database. User passwords are not properly hashed and salted, exposing every user’s   password.

Example #4: Predictable login credentials
Username and Password values that are easy to guess or that are used frequently can be guessed by attackers to obtain unauthorized access.

How to Prevent Broken Authentication and Session Management

SESSION MANAGEMENT

- Credentials should be protected: User authentication credentials should be protected when stored using hashing or encryption.
- Do not expose session ID in the URL: Session IDs should not be exposed in the URL (e.g., URL rewriting).
- Session IDs should timeout: User sessions or authentication tokens should be properly invalidated during logout.
- Recreate session IDs: Session IDs should be recreated after successful login.
- Do not send credentials over unencrypted connections: Passwords, session IDs, and other credentials should not be sent over unencrypted connections.

BROKEN AUTHENTICATION

- Password length: Minimum password length should be at least eight (8) characters long. Combining this length with complexity makes a password difficult to guess using a brute force attack.
- Password complexity: Passwords should be a combination of alphanumeric characters. Alphanumeric characters consist of letters, numbers, punctuation marks, mathematical and other conventional symbols.
- Username/Password Enumeration: Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code.
- Protection against brute force login: Enforce account disabling after an established number of invalid login attempts (e.g., five attempts is common). The account must be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed.

Improper Error Handling

- Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker).

- These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and such messages are also disturbing to normal users.

- Web applications frequently generate error conditions during normal operation. Out of memory, null pointer exceptions, system call failure, database unavailable, network timeout, and hundreds of other common conditions can cause errors to be generated.

- These errors must be handled according to a well thought out scheme that will provide a meaningful error message to the user, diagnostic information to the site maintainers, and no useful information to an attacker.

- One common security problem caused by improper error handling is the fail-open security check. All security mechanisms should deny access until specifically granted, not grant access until denied, which is a common reason why fail open errors occur.

- Other errors can cause the system to crash or consume significant resources, effectively denying or reducing service to legitimate users.

- Good error handling mechanisms should be able to handle any feasible set of inputs, while enforcing proper security. Simple error messages should be produced and logged so that their cause, whether an error in the site or a hacking attempt, can be reviewed.

- Error handling should not focus solely on input provided by the user, but should also include any errors that can be generated by internal components such as system calls, database queries, or any other internal functions.

How to Determine If You Are Vulnerable

- Simple testing can determine how your site responds to various kinds of input errors. More thorough testing is usually required to cause internal errors to occur and see how the site behaves.
- Another valuable approach is to have a detailed code review that searches the code for error handling logic. Error handling should be consistent across the entire site and each piece should be a part of a well-designed scheme.

- A code review will reveal how the system is intended to handle various types of errors. If you find that there is no organization to the error-handling scheme or that there appear to be several different schemes, there is quite likely a problem.

How to Protect Yourself

- A specific policy for how to handle errors should be documented, including the types of errors to be handled and for each, what information is going to be reported back to the user, and what information is going to be logged. All developers need to understand the policy and ensure that their code follows it.

- In the implementation, ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details.

- The OWASP Filters project is producing reusable components in several languages to help prevent error codes leaking into user’s web pages by filtering pages when they are constructed dynamically by the application.

Exception Management

Exceptions to any information security policies or procedures should be reviewed and approved by the senior management. Exceptions should be managed accordingly. In most cases, exceptions could be provided for the following:

- Legacy systems
- Third party applications
- Proprietary systems
- Physical security
- Emergencies
- Legal situations

Examples of exceptions

- A specialized application may be configured to require passwords that do not meet password policy requirements.
- A proprietary business system only allows for one administrator ID; however, multiple individuals support this system. Administrators must share this ID to manage the system.
- Some mobile device operating systems do not have the ability to meet the network device attachment requirements.
- A legacy system that does not meet the technical requirements.
- A lawsuit requires retaining information above and beyond the retention procedure.
- An emergency situation takes place that requires a workforce member to use the credentials of another workforce member to cover a time-critical business operation.

How exception is handled

The exception request should include:

- Requestors name or approving manager
- Explanation of the request
- The policy or procedure the request pertains
- The reason for the request
- Mitigating controls in place to mitigate any risks to the exception

The security management should review the request and determine whether or not to grant the exception. If an exception is made, other mitigating controls should be implemented. These mitigating actions can be administrative, physical, technical, or any combination of these types of controls.

Monitoring of exception

A determination should be made on how the exception should be monitored. This monitoring should be developed based on the exception made along with appropriate procedures for reviewing or auditing the exception.

An exception should be well documented. Documentation of an exception should include at least the following elements:

- Individuals or systems involved or scope of the exception
- Limitation of exception
- Mitigating controls required
- Dates/times
- Reasons for exception
- Approval