MODULE 5

COMMON WEB APPLICATION SECURITY VULNERABILITIES

- Web application vulnerabilities involve a system flaw or weakness in a web based application
- They have been largely due to not validating inputs, misconfigured web servers, and application design flaws, and they can be exploited to compromise the applications security
- Here are some common web application security vulnerabilities:

    - **Injection** : It is a type of security attack in which the malicious attacker inserts or injects a query via input data from the client side to the server

    - **Cross Site Scripting (XSS)** : When an application has untrusted data on a web page without proper validation , allowing attackers to execute scripts in a victim's web browser.

    - **Broken Authentication** : When authentication credentials and session identifiers are not protected at all times , allowing on attacker to hijack an active session and assume the identity of a user

    - **Sensitive Data Exposure :** When applications and APIs do not properly protect sensitive data such as financial data, social security numbers, usernames, passwords or health information

    - **Broken Access Control :** Access control specifies limits or boundaries in which a user is allowed to operate.

    - **Security Misconfiguration :** This usually gives full access to the system to the attacker thus resulting in a complete system compromise.

    - **XML External Entities :** This type is common to web applications that parse XML input. It is carried out when the input in the form of XML references an external entity but it is processed by a weak XML parser.

    - **Insecure Deserialization :** It is usually used for databases, caching, preserving, file systems, cache systems, interprocess communication, web services etc. If web application deserializes hostile or tampered objects that are supplied by the adversary, the application become vulnerable to this attack.

INJECTION FLAWS

- An injection flaw is a vulnerability in that applications allow an attacker to relay malicious code through an application to another system
- It allows hackers to inject client side or server side commands
- These are the flaws through which hackers can take control of web application
- Depending on the type of vulnerability an attacker might inject SQL queries , javascript , or OS commands and so on

Effects of Injection Flaws

- Allows an attacker to compromise the victim's system
- Allows hackers to execute malicious codes
- Allows attackers to do attacks cross site attackers request forgery
- Allows hackers to compromise databases
- Arbitrary file upload vulnerability may result in compromise of the entire database
- Loss of confidentiality, integrity and availability

Type of Injection Attacks

- There are so many types of injection attacks, some of them are :
    - **SQL Injection** : Attackers insert harmful SQL code through user inputs. This can allow them to access sensitive data, change database contents or even take control of the system
    - **Cross site scripting** : Allows a third party to execute a script in the user's browser on behalf of the web applications. The exploitation of XSS against a user can lead to various consequences such as account compromise, account deletion, privilege escalation, malware infection and many more
    - **HTML Injection** : Similar to text injection and as the name suggests it allows HTML content to be injected.
    - Fuzzing
    - Arbitrary File Upload Vulnerability

How to prevent Injection Flaws

- Use of prepared Statements (with parametrized queries)
- Use of properly constructed stored procedures
- Allow list input validation escaping all user supplied input
- Performing allow list input validation as a secondary defense
- Use strong web application firewalls to make exploitation difficult

BROKEN AUTHENTICATION

- The essence of broken authentication is where web application allow users to get into website by creating a new account and handling it for specific reasons
- In broken authentication, whenever a user login into its account, a session id is being created and that session id is allowed to that particular account only
- if the web application is crafted securely in terms of Authentication, then it is well and good but in case if it is not then the attacker may use several under given techniques.
    - **Credentials stuffing:**
        - In Credential Stuffing an attacker has a standard list of default passwords and usernames.
        - By this list, they can brute-force the accounts and can log in into legitimate accounts.
        - It is hardly recommended for users to change their default usernames and passwords to get secure from such attacks.
        - An attacker can generate a list of Custom passwords also depending upon his prior information to the target by various tools in Linux such as CRUNCH.
    - **Unhashed Passwords:**
        - Changement of clear-text password into scrambled words through which an attacker can be tricked is called hashing of passwords.
        - . Using this technique user can lose his Account Authorization & Confidentiality.

- **Misconfigured Session Timeouts:**

    - The scenario where a user had log out of the account and an attacker has the cookie of that user.
    - Using the cookie, an attacker can still have access to that account.
    - Using this type of loophole Cookie Tampering, Session hijacking and other attacks can be chained into one single loophole, which is also known as chaining of bug.
    - Such type bugs are referred to as Misconfigured Session Timeout.

Impacts of Broken Authentication Vulnerability:

- Exposed numerous User Accounts
- Data Breaches
- Administrative Access
- Sensitive Data Exposure
- Identity Theft

Remediation of Broken Authentication Vulnerability

- Broken Authentication Vulnerability is a severe issue if it is prevailing in a Web Application because such loopholes can cause the company a million dollar attack in terms of Data Breaches.
- Some of the remediation that a web application can impose on itself to get safe from such attacks.

    - Multifactor Authentication Must be implemented to bypass such attacks.
    - Password Complexity must be high for the user accounts.
    - Rotation of Session Id's after Successful login.
    - Validation Of Session Id's.

SENSITIVE DATA EXPOSURE

- Sensitive Data Exposure Vulnerability exists in a web application when it is poorly designed.
- It allow attacker to apply various security practices and find the sensitive data that is related to particular website.
- By Sensitive Data Exposure vulnerability, attackers may be able to find sensitive data such as session tokens, authentication credentials, databases etc.
- By such sensitive data an attacker will be able to exploit the web application and the security of website will be breached.

Vulnerability can be exploited by attackers.

- **Clear Text Transmission:**

    If there is clear transmission of data in background in a web application then there might be a risk of data exposure to the attacker.
    Example – clear transmission of text may includes the credentials of user.

- **Cryptographic Algorithm:**

    Old cryptographic algorithms that were used in old web apps might be a risk factor.
    There may be a chance that attackers could have bypass that algorithm and get access to sensitive data.

- **Cryptographic Keys:**

    Cryptographic keys always play a vital role in a web applications.
    If Cryptographic keys are not properly rotated or old & weak keys are used then in that case web application will be at risk of exposure of data.

- **Encryption:**

  Web application must enforce proper encryption techniques in order to prevent attacks and to safeguard the confidential information.

How To Prevent 'Sensitive Data Exposure'?

- Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.
- Don't store sensitive data unnecessarily. Discard it as soon as possible. Data you don't have can't be stolen.
- Ensure strong standard algorithms and strong keys are used, and proper key management is in place.
- Ensure passwords are stored with an algorithm specifically designed for password protection,
- Disable auto complete on forms collecting sensitive data and disable caching for pages that contain sensitive data.
- Consult Information Security Experts for detailed and thorough checks of all sensitive web applications.
- Consider investing in DLP solutions or for Web Applications a WAF with custom rules (mask credit card numbers, SIN numbers) with targeted policies to prevent sensitive data exposure to clients

XML EXTERNAL ENTITY (XXE)

- XML external entity (XXE) vulnerabilities are a security risk that allows attackers to manipulate a web application's XML data processing.
- An XXE vulnerability is a security vulnerability that allows attackers to access sensitive data or execute malicious code in a web application. This happens when the application accepts XML input from an untrusted source and doesn't properly validate it.
- XXE vulnerabilities occur when a web application accepts XML input from an untrusted source and the XML parser is not configured correctly. This can lead to a number of consequences, including:
  - **Data disclosure**
  - **Remote code execution**
  - **Denial of service**
  - **Server-side request forgery**
- To prevent XXE vulnerabilities, you can configure the XML parser to disallow custom document type definitions (DTD) and external XML entities.

How XXE Attacks Work

1. **Untrusted XML Input:** An attacker submits malicious XML data to the application. This data might contain a seemingly harmless reference to an external entity.

2. **Improper Validation:** The application processes the XML data without adequately validating the external entity reference.

3. **Exploiting the Reference:** The application attempts to access the external resource specified in the entity reference. An attacker can manipulate this behavior for malicious purposes.

Potential Impacts of XXE Attacks

- **Information Disclosure:** Attackers can use XXE to retrieve files from the server, potentially exposing sensitive data like user credentials, financial information, or internal documents.
- **Server-Side Request Forgery (SSRF):** XXE attacks can be used to trick the server into making unauthorized requests to other internal systems or even external websites.
- **Denial-of-Service (DoS):** By crafting malicious XXE entities that consume excessive resources, attackers can overload the server and make it unavailable to legitimate users.
- **Code Execution (Rare):** In some cases, XXE vulnerabilities can be exploited to execute arbitrary code on the server, allowing attackers complete control over the system.

Prevention of XXE vulnerabilities:

Developers can significantly improve the security posture of their web applications and prevent XXE vulnerabilities:

- **Disable External Entity Processing:** This is the most secure approach but may limit functionality. Configure the XML parser to disallow external entity resolution entirely.
- **Whitelist Allowed Entities:** Specify a list of authorized external entities that the parser can access. This restricts potential attack vectors but requires maintaining the whitelist.
- **Input Validation and Sanitization:** Rigorously validate and sanitize user input before processing XML documents. This involves checking for malicious XML constructs like external entity references and removing them.
- **Update XML Libraries:** Regularly update XML processing libraries used in your application. Updates often include patches for identified XXE vulnerabilities.
- **Secure Development Practices:** Implement secure coding practices throughout the development lifecycle. This includes using secure coding libraries and frameworks, and following best practices for handling user input.

BROKEN ACCESS CONTROL

- A broken access control vulnerability is a security flaw that allows unauthorized users to access, modify, or delete data they shouldn't have access to.
- Poor implementation of access control mechanism leads to flawed access control, which can be easily exploited. This is called broken access control.

- Because of broken access control, unauthorized users can view content that they are not allowed to view, can perform unauthorized functions, even an attacker can delete the content, or take over site administration.
- We can split access control vulnerabilities mainly into **three categories** :
    - **Horizontal privilege escalation :** When users can access data of other users who have the same level of permissions as them.
    - **Vertical privilege escalation :** When users can access data of those users who have permissions to perform some actions that normal users don't, with vertical access controls, different types of users have access to different application functions.
    - **Context-dependent privilege escalation :** When a user is allowed to perform actions in the wrong order.

Impacts of broken access control

1. Whenever we make an account on a website, we are given a unique ID. By using that ID, we can access the database where all our sensitive contents are saved.
2. Hackers can take advantage of access flaws to gain access to the resources and services that should not be accessible to normal users.
3. There are possibilities of Distributed Denial of Service attacks (DDOS). With access to those user accounts, attackers can attempt to disrupt the normal traffic of a targeted server, by deploying bots. This makes it difficult for the server to distinguish legitimate user traffic from attack traffic.

How to Prevent Broken Access Control?

- **OWASP (Open Web Application Security Project)**, an online community that analyzes weaknesses and attacks on web applications
- . To prevent broken access control, the security team can adopt the following practices-

1. Continuous Inspection and Testing Access Control:

- Efficient continuous testing and inspecting the access control mechanism is an effective way to detect the newer vulnerabilities and correct them as soon as possible.

2. Deny Access By Default:

- Design access control in such a way that not everyone can get access to the resources and functionalities unless it is intended to be publicly accessible.
- You can apply **JIT (Just-in-Time)** access, which helps to remove the risks associated with standing privileges.

3. Limiting CORS Usage:

- **CORS (Cross-Origin Resource Sharing)** protocol provides a controlled way to share cross-origin resources.
- The implementation of the CORS relies on the Hypertext Transfer Protocol (HTTP) headers used in the communication between the client and the target application.
- When CORS protocol is misconfigured, it makes it possible for a domain to be controlled by a malicious party to send requests to your domain.

4. Enable Role-based Access Control:

- This is a widely used access control mechanism.
- According to this, users are given permissions based on their roles.
- Instead of identifying each user individually, users are assigned to a group of roles, this way the struggle of IT support and administration can be reduced, and operational efficiency will be maximized.

5. Enable Permission-Based Access Control:

- This is an access control method, where the authorization layer checks if the user has permission to access particular data or to perform a particular action, typically by checking if the user's roles have this permission or not.

6. Enable Mandatory access control:

- It is a security method, that restricts the ability to access resources based on the sensitivity of the information that the resource contains.
- This security policy can only be controlled by the administrator; regular users don't have the ability to change that policy. Because of this centralized administration, it is considered to be very secure.

SECURITY MISCONFIGURATION

- Security misconfigurations are security controls that are inaccurately configured or left insecure, putting your systems and data at risk.
- Basically, any poorly documented configuration changes, default settings, or a technical issue across any component in your endpoints could lead to a misconfiguration.
- Security misconfigurations are not hard to fix, but they are unavoidable in an enterprise operating at scale.
- Finding them is a needle in the haystack, as they can be located across any component in an organization's systems, such as its servers, operating systems, applications, and browsers.
- Lack of visibility and centralized means to remediate misconfigurations makes organizations fall victim to misconfiguration attacks.

Security misconfiguration attacks can be prevented by

- Using Dynamic application security testing (DAST)
- Disabling the use of default passwords
- Keeping an eye on cloud resources, applications, and servers

CROSS-SITE SCRIPTING (XSS)

- Cross Site Scripting (XSS) is a vulnerability in a web application that allows a third party to execute a script in the user's browser on behalf of the web application.
- Cross-site Scripting is one of the most prevalent vulnerabilities present on the web.
- The exploitation of XSS against a user can lead to various consequences such as account compromise, account deletion, privilege escalation, malware infection and many more.
- The possibility of getting XSS ed arises when a website does not properly handle the input provided to it from a user before inserting it into the response.
- An attacker exploits this by injecting on websites that doesn't or poorly sanitizes user-controlled content. By injecting vulnerable content a user can perform
    - Cookie Stealing.
    - Defacing a website.
    - Bypassing CSRF Protection etc.
- **Cross-site scripting attacks can be prevented by**
    - Using appropriate response headers
    - Filtering the input and encoding the output
        - Blacklist filtering.
        - Whitelist filtering.
        - Contextual Encoding.
        - Input Validation.
        - Content Security Policy
    - Using the content security policy
    - Applying a zero-trust approach to user input

INSECURE DESERIALIZATION

- Insecure deserialization is a vulnerability that occurs when untrusted data is used to manipulate an application's deserialization process.
- This allows attackers to perform malicious actions, such as: Executing code, Manipulating objects, Performing injection attacks, Bypassing authentication, and Launching denial-of-service (DoS) attacks.
- Insecure deserialization occurs when user-controllable data is deserialized without any input validation.
- To prevent insecure deserialization, you can check how the unserialize() function is used and review how external parameters are accepted.
- Insecure deserialization is sometimes called an "**object injection**" vulnerability.
- If the attack is successful, the attacker will be able to carry out remote code execution which is one of the most significant attacks.

INSUFFICIENT LOGGING AND MONITORING

- Insufficient logging and monitoring refers to a security event not being correctly detected, logged and monitored to ensure adequate and timely response to the incident or breach.
- This is the most common reason for most major breaches to occur.
- Since most organizations do not invest in monitoring and effective logging or responding in a timely manner to the threat, the attackers can easily break the security system
- Insufficient logging, detection, and monitoring occur in the following cases:
    - Verifiable events such as logins, failed logins and high-value transactions are not logged.

Insufficient logging and monitoring is when an organization doesn't properly record and monitor events and activities on its systems and networks. This can lead to security events going undetected, which can have serious consequences for an organization.

Insufficient logging and monitoring are said **to occur when**:

- Logs are only stored locally with no redundancy.
- Logs of application, devices and/or APIs are not monitored for anomalous behavior.
- There are of lack of support mechanism in place to export log data.
- There is obscure error logging.
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

Here are **some examples** of insufficient logging and monitoring:

- Not logging important events, like failed login attempts
- Not monitoring logs and alerts for suspicious activity
- Not responding to alerts or investigating suspicious activity
- Not having a centralized logging system
- Not retaining logs for a sufficient amount of time
- Logs are only stored locally with no redundancy
- Logs of application, devices, and/or APIs are not monitored for anomalous behavior
- The application is unable to detect, escalate, or alert for active attacks in real time or near real time

To prevent insufficient logging and monitoring

- Create audit logs for sensitive events
- Store logs in a centralized location
- Retain logs for a defined amount of time
- Regularly audit logs.