What Is OWASP Top 10

OWASP Top 10 Vulnerabilities is the list of the 10 topmost vulnerabilities that affect applications in the world. These are arranged according to their impact, the security risk involved, and how to mitigate against these vulnerabilities.

OWASP TOP 10 Each Vulnerability Mitigations

1. Injection

Injection attack is all about injecting SQL, NoSQL, OS, and LDAP into the application. It can be as SQL queries through the input interface of the application. Successful SQL injection can lead to sensitive data from the database being exposed.

Injection Mitigation:

- Making use of Prepared Statements with Parameterized queries.
- Making use of Stored Procedures.
- Implement input validation and sanitization.
- Make sure you are escaping all user-supplied input.
- Using safer API which avoids the use of the interpreter
- Using parameterized queries when coding
- Segregating commands from data to avoid exposure to attacks.

2. Broken Authentication

Broken Authentication is a vulnerability that allows an attacker to use manual or automatic methods to try to gain control over any account they want in a system. In worse conditions, they could also gain complete control over the system. This vulnerability is also more dangerous because websites with broken authentication vulnerabilities are very common on the web.

Example: Session Time-out, Credential Stuffing

- Allows automated attacks such as brute-force attacks, where an attacker tries different usernames and passwords to carry out an attack.
- Allows default and weak passwords. Example: password, admin.
- Allows a weak credential recovery system or forgot password workaround.
- Allows plain text, unencrypted passwords.

Broken Authentication Mitigation:

- Making use of captcha.
- Reduce the number of tries for a particular user based on the session ID or the IP.
- Blocking multiple requests coming from the same IP.
- Making the admin login page inaccessible to the public.
- Implement multi-factor authentication to prevent brute-forcing and credential theft.
- Implementing multi-factor authentication
- Protecting user credentials
- Sending passwords over encrypted connections

3. Sensitive Data Exposure

This is also known as information disclosure or information leakage. This usually occurs when an application or website unknowingly discloses sensitive data to users who do not have the privilege of view or access.

Example: Unencrypted data

- Financial information
- Login credentials
- Commercial or business data
- Health record
- Technical details about the application or website.

Sensitive Data Exposure Mitigation:

- Always identify and classify which data is sensitive according to the privacy laws and the regulatory requirements.
- Immediately remove any data that is not needed to be stored.
- If you are going to be store any sensitive data, make sure it is encrypted at rest.
- Use proper key management.
- Make sure you encrypt all data in transit with security protocols such as TLS and SSL.
- You can enforce encryption on your application simply by using HTTP Strict Transport Security (HSTS).
- Do not cache sensitive data.
- Always store passwords with different encryption methods.
- Using the secure URL's
- Using strong and unique passwords
- Encrypting all sensitive information that does need to be stored

4. XML External Entities (XXE)

This vulnerability occurs for web applications that parse XML input. It happens when poorly configured XML processors evaluate external entity references within the XML documents and send sensitive data to an unauthorized external entity, i.e., a storage unit such as a hard drive. By default, most XML parsers are vulnerable to XXE attacks.

Example: File Retrieval, Blind XXE

- According to the OWASP Top 10, the XML external entities (XXE) attack can exploit these:
- Vulnerable XML parser that allows an attacker to upload XML or include a hostile command in an XML document.
- Vulnerable integration.
- Vulnerable dependencies.
- Vulnerable code.

XXE Injection Mitigation:

- You must disable DTD and XML external entity features.
- All the XML processors and libraries used within the application must always be updated and patched.

- There is a need to put a proper validation in place for every user inputs.
- Use good XML parsers that are not vulnerable by default.
- Make use of a very good SAST tool that can help detect XXE in your source code.
- Make sure XML and XSL file upload functionality validates every incoming XML by using XSD validation.
- Make use of API security gateways
- Make use of WAF (Web Application Firewall) to detect, block any XXE attacks and can be used for closed monitoring.
- You can start making use of JSON data format, which is less complex.

5. Broken Access Control

Access control sometimes referred to as authorization, is how a web application grants access to users while denying access to other users. It can also control access to contents and functions on an application.

Broken access control always provides access to user accounts to an attacker, and the attacker can act or operate like the administrator and access unauthorized data, sensitive files, and even change access rights.

Example: Privilege Elevation, JWT Invalidation

The following are examples of Broken Access Control:

- Access to an admin panel.
- Access to a server via FTP / SFTP / SSH.
- Access to a website's control panel.
- Access to other restricted applications on your server.
- Access to a database.

Broken Access Control Mitigation:

- Use a proper session management method.
- Use a token for authorization of users like JWT.
- Always deny public access by default except in rare cases for some resources that needed to be accessed publicly.
- Regular audit and test access controls should be conducted to confirm its functionality.
- Disable the web-server directory listing and confirm backup files are not present in the web roots.

6. Security Misconfiguration

Security Misconfiguration arises when Security settings are defined, implemented, and maintained as defaults. Good security requires a secure configuration defined and deployed for the application, web server, database server, and platform.

Example: Misconfigured HTTP Headers, Default configuration

- Missing an important security hardening across any part of the application stack, or the cloud services permissions is not configured well.
- When you enabled or installed features that are not required (e.g. ports, services, privileges, accounts, and pages).

- When all default accounts and their passwords are still enabled and can be reached.

Security Misconfiguration Mitigation:

- A regular hardening of the application environment is very important, and it's fast and easy to deploy another environment that is properly locked down. Each environment should be configured identically, but with different credentials.
- Make sure you review and update all the configuration settings appropriate to all security updates and patches that are part of the patch management process.
- Making sure you send security directives to clients, e.g. Security Headers.

7. Cross-Site Scripting (XSS)

Cross-site scripting occurs when an attacker is able to insert untrusted data/scripts into a web page. The data/scripts inserted by the attackers get executed in the browser can steal user's data, deface websites etc.

Example: Session Hijacking

Cross Site Scripting Mitigation:

- Using appropriate response headers
- Filtering the input and encoding the output
- Using the content security policy
- Applying a zero-trust approach to user input
- Implement Content Security Policy.
- Use HTTP Only cookie flag.
- Deploy firewall that protects against XSS.

8. Insecure Deserialization

Insecure Deserialization vulnerability allows an attacker to remotely execute code in the application, tamper or delete serialized (written to disk) objects, conduct injection attacks, replay attacks, and elevate privileges. This attack is also known as untrusted Deserialization.

Example: Remote Execution

Insecure Deserialization Mitigation:

- Do not allow serialized objects from unreliable sources.
- Always carry out some integrity checks like digital signatures on serialized objects in order to prevent compromising of data and hostile object creation.
- Always carry out enforcement of strict type constraints when doing deserialization before the creation of the object.
- Make sure you isolate and run code that deserializes in low privilege environments when possible.
- Make sure that deserialization exceptions and failures are properly logged, like where the incoming type is not the same as the expected type.
- Isolating the code that deserializes and running it in low privilege environments to prevent unauthorized actions

9. Using Components With Known Vulnerabilities

This vulnerability results from a developer using a component, framework, library, or some dependencies that already have a known vulnerability that may compromise the entire system.

When such components are executed with full privileges and it's vulnerable, this will make the exploitation from an intruder very easy, which may cause some serious data loss or server takeover.

Example: Open-Source Components

Components With Known Vulnerability Mitigation:

- Always remove any unused dependencies, unnecessary features, components, and files.
- Removing all unnecessary dependencies
- Using virtual patching
- Using components only from official and verified sources

10. Insufficient Logging and Monitoring

The importance of securing a website or application through sufficient logging and monitoring cannot be underemphasized because improper logging can result in information disclosure.

Example: Missing Security Information, Missing Log

Insufficient Logging & Monitoring Mitigation:

- Always ensure that every log is captured in a way that a management tool can easily use it.
- Every transaction on an application should have an audit trail with integrity controls in place so that transactions cannot be manipulated, even delete.
- Implementing logging and audit software
- Establishing an effective monitoring system
- Thinking like an attacker and use a pen testing approach