**Case Study 5**

**Aim** - To study and implement at least two methods of open web application security project (OWASP).

**Theory -**

**Introduction -**

**OWASP** stands for the **Open Web Application Security Project**. It is a non-profit organization focused on improving the security of software. OWASP provides resources, tools, documentation, and best practices to help developers, security professionals, and organizations build more secure web applications and APIs.

The primary goal of OWASP is to raise awareness about web application security risks and provide guidance on how to address them. They do this through various initiatives, projects, and community collaborations. OWASP is known for its "Top Ten" list, which highlights the ten most critical security risks faced by web applications, and it updates this list periodically to reflect the evolving threat landscape.

**List of few OWASP Vulnerability.**

**1. Injection -** An injection is a security risk that you can find on pretty much any target. Basically, it happens when a server-side interpreter processes untrusted user input as part of a command or a query. There are many vulnerabilities which cause injection. Here are some examples:

* SQL injection: You can find a SQL injection when the developer runs a SQL query that takes a parameter you control as an input. If you successfully exploit it, you steal data from the database, edit it or delete it altogether.
* OS command injection: It happens when user input is used as part of an insecure call to operating system commands. If you find one, you can run arbitrary operating system commands on the vulnerable server.
* XPATH injection: It targets the query language typically used in XML. When you can control part of the query. Therefore, you can bypass restrictions, read unauthorized XML nodes, etc.
* LDAP Injection: When your target insecurely uses some user input to query an LDAP directory, you can perform an injection to bypass restrictions, read unauthorized data, etc.

**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.

**2. Broken authentication and session management -** Authentication is the process of verifying a user's identity, typically through methods like username and password. It's a crucial aspect of web applications, as it establishes trust in user identities and allows for secure access. Mistakes or flaws in the authentication process can result in broken authentication, potentially leading to unauthorized access or security breaches.

**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.

**3. Sensitive data exposure -** If IT assets disclose data which is not meant to be publicly accessible, they suffer from sensitive data exposure. On the one hand, this data can be at rest, like databases or files. On the other hand, it can be in transit, especially if unencrypted or weak encrypted data is used for transmission.

**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 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.

**4. Broken access control -** Broken access control occurs when an application permits users to perform actions they shouldn't be authorized to do. This vulnerability can result from issues like failing to validate permissions for identifiers, leading to Insecure Direct Object Reference (IDOR) risks. Other related vulnerabilities include Cross-site Request Forgery (CSRF), Cross-Origin Resource Sharing (CORS) misconfigurations, and forced browsing.

**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 need 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.
* Make sure you have an access control set up that will enforce the right to every user like what each user can perform and not that the user can create, update, delete and read any record.
* There is a need for domain models that will enforce business limit requirements.

**5. Security misconfiguration -** It refers to vulnerabilities stemming from inadequate configurations in IT assets. This risk extends beyond web assets and can affect various components, including network devices and hardware. For example, leaving default administration settings unchanged on a smart door lock can lead to unauthorized access and configuration changes. In web applications, security misconfigurations might involve enabling directory listing or failing to disable debug mode, inadvertently exposing sensitive information.

**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.
* Create automated process environments to verify the effectiveness of the configuration settings.

**6. Cross-site Scripting (XSS)** - This is one of the famous client-side vulnerabilities. It allows an attacker to run arbitrary Javascript code on the victim’s web browser. XSS becomes possible when user input ends up inside an HTML page or a piece of Javascript code without proper encoding. There are basically three types of XSS, all of them along with hands-on tutorials are explained further:

* Stored XSS happens when the user input gets stored in the application’s datastore, then retrieved back and rendered in a page without proper encoding.
* Reflected XSS happens when user input gets directly returned into the HTML page without proper encoding.
* DOM XSS happens when user input gets inside a Javascript code. Here, it is possible to exploit XSS even if there is no request made to the server.

**Cross Site Scripting Mitigation:**

We can encode the following characters with HTML entity encoding to prevent any execution of any form.

* & –> & amp;
* < –> & lt;
* –> & gt;
* ” –> & quot;
* ‘ → &# x27;
* CSS encode and make sure it’s validated before Inputting untrusted data into HTML Style Property Values.
* Using frameworks like Ruby on Rails and React JS that escape XSS with ease.
* JavaScript encode Before Inputting untrusted data into JavaScript data values.
* HTML encodes JSON values in an HTML context and reads the data with JSON.parse.
* URL encode Before Inputting Untrusted Data into HTML URL Parameter Values.
* Implement Content Security Policy.
* Use the HTTPOnly cookie flag.
* Deploy firewall that protects against XSS.

**7. Insecure deserialization -** Insecure deserialization happens when the developer doesn’t check serialized data that a user sends to the application. This is another vulnerability where a lack of user input validation can lead to serious security problems. It is hard to exploit, but when it works, it can lead to either remote code execution or denial of service.

**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.

**8. Using components with known vulnerabilities -**

This vulnerability arises when developers incorporate external dependencies into their code without thoroughly assessing their security. Such dependencies may contain known vulnerabilities that hackers can exploit. This oversight has historically resulted in significant breaches and should be diligently monitored and managed to prevent future security incidents.

**Components with Known Vulnerability Mitigation:**

* Always remove any unused dependencies, unnecessary features, components, and files.
* Always obtain your application components from approved and official sources with secure links. This will reduce the chance of including any malicious component in your application.
* Always check and avoid frameworks, libraries, and components that are not maintained and do not have security patches for older versions.
* Always use library scanners to test for any vulnerabilities in the application packages you are using.

**9. Insufficient logging and monitoring -**

When a hacker infiltrates a network, IT systems will generate traffic which usually doesn’t correspond to the normal one, unless you are dealing with highly skilled hackers who have time and money to go after your IT infrastructure. If you can’t detect this abnormal behavior as soon as possible, you are essentially giving them enough time to achieve their goal.

**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 deleted.
* There is a need to implement a very effective and efficient monitoring and alerting system which will always inform us of any malicious or suspicious activities so that they can be immediately remediated.
* Make sure the audit log does not store your password and any other sensitive content, as this is very risky.

**Top OWASP vulnerabilities**

**A01: Broken Access Control (2021):**

This vulnerability, now ranked as the most critical, exposes web applications to unauthorized access. Data indicates that it affects a significant number of applications, making it a prominent security concern.

**A02: Cryptographic Failures (2021):**

Formerly known as Sensitive Data Exposure, this vulnerability focuses on cryptographic issues leading to sensitive data exposure or system compromise, highlighting the importance of proper encryption and data protection.

**A03: Injection (2021):**

Injection vulnerabilities, including SQL injection and Cross-site Scripting (XSS), are still a major concern, as they allow attackers to manipulate applications through malicious input, potentially compromising data and system integrity.

**A04: Insecure Design (2021):**

A new category emphasizing design flaws, it underscores the need for secure design patterns and principles to address vulnerabilities early in the development process.

**A05: Security Misconfiguration (2021):**

This vulnerability, now higher in rank, highlights the risk posed by misconfigured applications, which can expose sensitive data and other security issues due to highly configurable software.

**A06: Vulnerable and Outdated Components (2021):**

This category, moving up in importance, points out the challenge of identifying and addressing vulnerabilities in third-party components, which can lead to security breaches.

**A07: Identification and Authentication Failures (2021):**

Previously Broken Authentication, this category focuses on issues related to identifying and authenticating users, highlighting the importance of robust identification processes in applications.

**A08: Security Logging and Monitoring Failures (2021):**

Formerly Insufficient Logging & Monitoring, this expanded category addresses various types of failures in monitoring and alerts, crucial for detecting and responding to security incidents.

**Conclusion -** In conclusion, this case study explores the Open Web Application Security Project (OWASP) and its methods for improving web application security. It covers various OWASP vulnerabilities, their potential risks, and mitigation strategies. The content aligns with the study's aim to understand and implement at least two OWASP methods. It provides valuable insights into web application security, making it a valuable resource for enhancing the security of web applications and APIs.