**Task 2: Introduction to Web Application Security**

**Web Application Security Assessment Report**

Documentation of Vulnerabilities Found Using OWASP ZAP

1. SQL Injection

Discovery:

- Tool Used: OWASP ZAP

- Method: Active scan of the web application

- Description: During the active scan, OWASP ZAP identified a potential SQL injection vulnerability in the login form of the web application.

- Evidence: The alert indicated that user input was not properly sanitized, allowing SQL commands to be executed in the backend database.

Exploitation:

- Steps:

1. Identified the login form where user credentials (username and password) are submitted.

2. Inserted a simple SQL injection payload (e.g., `' OR '1'='1`) into the username or password field.

3. Submitted the form and observed the application's response.

- Impact: The application returned an error message indicating that the SQL query was malformed, confirming the presence of a SQL injection vulnerability.

Why It Is Dangerous:

- Data Theft: An attacker can exploit this vulnerability to retrieve sensitive data from the database, including user credentials, personal information, and other confidential data.

- Data Manipulation: An attacker can modify, delete, or insert data into the database, leading to unauthorized actions and potential data corruption.

- Privilege Escalation: In some cases, an attacker can leverage SQL injection to gain administrative access to the database or even the underlying server.

2. Cross-Site Scripting (XSS)

Discovery:

- Tool Used: OWASP ZAP

- Method: Active scan of the web application

- Description: During the active scan, OWASP ZAP identified a potential Cross-Site Scripting (XSS) vulnerability in a comment field on one of the web pages.

- Evidence: The alert indicated that user input was not properly sanitized or encoded, allowing JavaScript code to be executed in the context of other users' browsers.

Exploitation:

- Steps:

1. Identified a comment field where users can submit text.

2. Inserted a simple JavaScript payload (e.g., `<script>alert('XSS')</script>`) into the comment field.

3. Submitted the form and observed the application's response.

- Impact: When another user viewed the page with the injected script, a JavaScript alert box was displayed, confirming the presence of an XSS vulnerability.

Why It Is Dangerous:

- Session Hijacking: An attacker can steal session cookies, allowing them to hijack the user's session and impersonate the user.

- Phishing: An attacker can redirect users to malicious websites or create fake login forms to steal credentials.

- Data Theft: An attacker can extract sensitive data from the user's browser, such as personal information or authentication tokens.

Conclusion

These vulnerabilities, SQL Injection and Cross-Site Scripting (XSS), were discovered using OWASP ZAP during an active scan of the web application. Both vulnerabilities pose significant security risks and can be exploited to compromise the integrity and confidentiality of the application and its users' data. Addressing these vulnerabilities is crucial to enhancing the overall security posture of the web application and protecting it from potential attacks.

Recommendations

1. SQL Injection:

- Input Validation: Implement strong input validation and parameterized queries to prevent SQL injection.

- Regular Updates: Keep the application and its dependencies up to date to address known vulnerabilities.

2. Cross-Site Scripting (XSS):

- Output Encoding: Encode user inputs to prevent the execution of malicious scripts.

- Content Security Policy (CSP): Implement CSP headers to restrict the sources of executable scripts.

By addressing these vulnerabilities, the web application can be made more secure, reducing the risk of data

**Provide screenshots and a brief explanation of the exploitation process.**

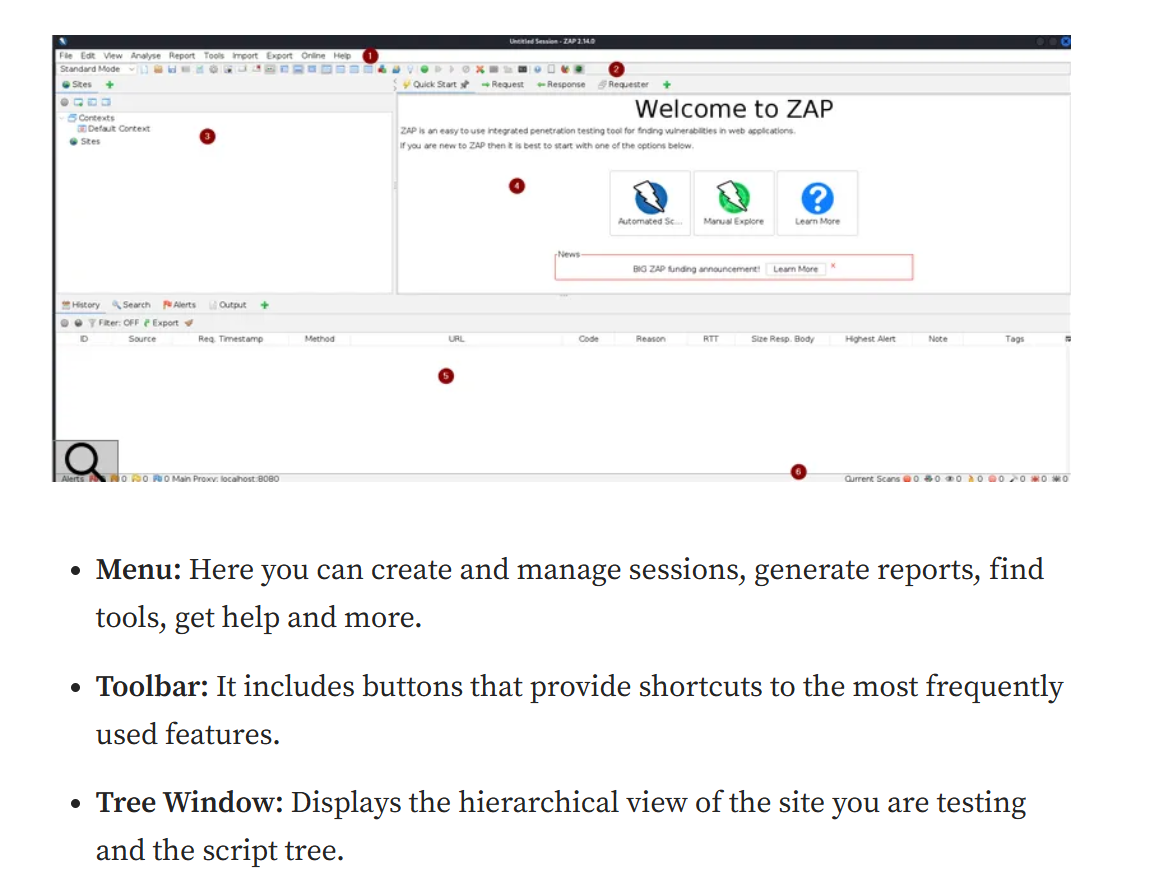
Exploitation Process Using OWASP ZAP and XSS via SQL Injection

Step 1: Setting Up OWASP ZAP

1. Install and Start OWASP ZAP: Download and install OWASP ZAP from the [official website](https://www.zaproxy.org/download/). Once installed, start the application, which will display a main dashboard with options for target scanning and vulnerability detection.

2. Configure Proxy: Configure your browser’s proxy settings to use OWASP ZAP as a proxy. Generally, this involves setting your browser’s HTTP and HTTPS proxies to `localhost` on port `8080` (or whatever port ZAP is using).

3. Add Target Application URL: In OWASP ZAP, enter the target web application URL under the "Sites" section or use the “Quick Start” tab to begin scanning.

****

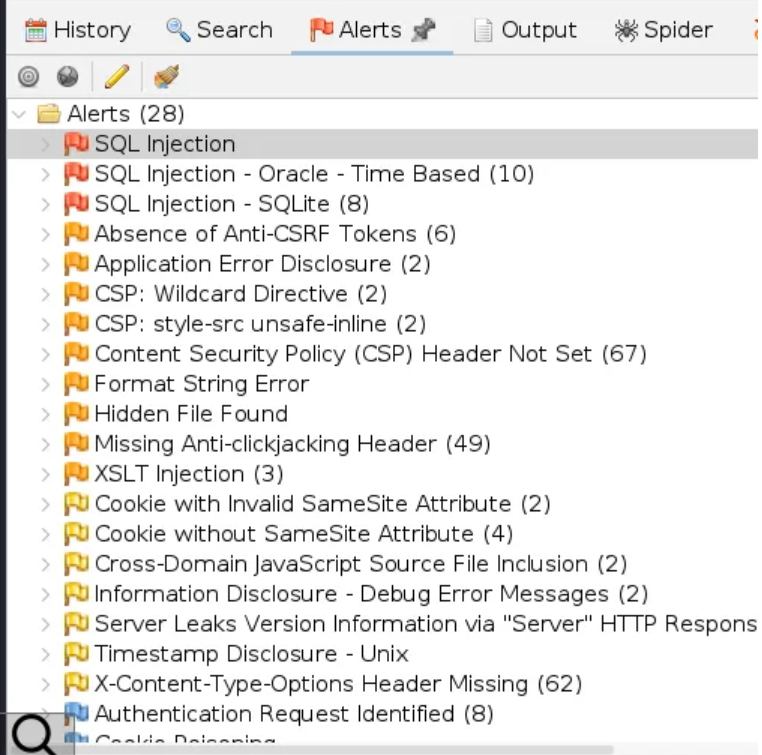
**---**

Step 2: Discovering Vulnerabilities with OWASP ZAP

1. Passive Scan: Begin with a passive scan by simply browsing through the target application. OWASP ZAP will capture requests and responses, noting potential vulnerabilities such as exposed inputs.

2. Active Scan: To perform an in-depth scan, select the target in the OWASP ZAP interface, right-click, and select Attack > Active Scan. This scan probes for vulnerabilities like SQL injection or XSS on input fields and forms.

3. Analyze Alerts: Review the “Alerts” tab in OWASP ZAP to find specific vulnerabilities like Reflected XSS or SQL Injection. Alerts indicate exploitable fields and parameters.

****

**---**

Step 3: Exploiting SQL Injection

1. Identify Vulnerable Fields: Using OWASP ZAP, identify input fields or URL parameters that are vulnerable to SQL injection. Typically, this involves fields that accept SQL payloads, like `login.php?user=<payload>`.

2. Test with Payloads: Manually test SQL injection with common payloads, such as:

```sql

' OR 1=1 --

```

or more complex payloads for dumping database contents, for example:

```sql

' UNION SELECT username, password FROM users --

```

3. Automated Exploitation: Alternatively, you can use SQL injection tools integrated into OWASP ZAP or external tools like `sqlmap` to automate exploitation of the SQL injection vulnerability:

```bash

sqlmap -u "http://target.com/vulnerable\_endpoint" --dbs

```



Step 4: Exploiting XSS (Cross-Site Scripting)

1. Locate XSS Vulnerable Input: Use OWASP ZAP to detect XSS by reviewing the “Alerts” tab for XSS vulnerabilities. Vulnerable fields may include search boxes or comment fields that reflect input in the output.

2. XSS Payloads: Test for XSS by injecting scripts into vulnerable fields, such as:

```html

<script>alert("XSS");</script>

```

Upon submission, check if the payload is executed (e.g., if an alert box pops up, the XSS vulnerability is confirmed).

3. Automate XSS Detection: OWASP ZAP’s active scanner will test for XSS across multiple fields automatically. Adjust payloads to capture cookies or execute actions on behalf of users.

**Simple steps that could mitigate these vulnerabilities.**

Certainly! Here are some simple steps that can help mitigate SQL Injection and Cross-Site Scripting (XSS) vulnerabilities:

Mitigating SQL Injection

1.Input Validation and Sanitization:

- Description: Ensure that all user inputs are validated and sanitized before they are processed.

- Steps:

1. Validate the type and format of input data.

2. Use libraries or frameworks that provide built-in protection against SQL injection.

3. Implement parameterized queries or prepared statements to ensure that user input is treated as data and not executable code.

2.Use of Stored Procedures:

- Description: Stored procedures can help separate the logic from the data, reducing the risk of SQL injection.

- Steps:

1. Create stored procedures for database operations.

2. Call these procedures from the application code instead of embedding SQL queries directly.

3. Least Privilege Principle:

- Description: Ensure that the database account used by the application has the minimum privileges required.

- Steps:

1. Create a dedicated database account for the application.

2. Grant only the necessary permissions to this account.

4. Regular Updates and Patching:

- Description: Keep the application and its dependencies up to date to address known vulnerabilities.

- Steps:

1. Regularly update the application and its libraries.

2. Apply security patches as soon as they are available.

Mitigating Cross-Site Scripting (XSS)

1. Output Encoding:

- Description: Encode user inputs before displaying them on web pages to prevent the execution of malicious scripts.

- Steps:

1. Use libraries or frameworks that provide built-in protection against XSS.

2. Encode output data using appropriate encoding methods (e.g., HTML encoding).

2. Content Security Policy (CSP):

- Description: Implement a Content Security Policy to restrict the sources of executable scripts.

- Steps:

1. Add CSP headers to your HTTP responses.

2. Define the allowed sources for scripts, styles, and other resources.

3. Input Validation and Sanitization:

- Description: Validate and sanitize user inputs to ensure that only safe data is processed.

- Steps:

1. Validate the type and format of input data.

2. Use libraries or frameworks that provide built-in protection against XSS.

4. Security Headers:

- Description: Use security headers to add an extra layer of protection against XSS.

- Steps:

1. Add headers like `X-XSS-Protection` to enable the browser's built-in XSS filter.

2. Use headers like `X-Content-Type-Options` to prevent MIME-sniffing.

Conclusion

By implementing these simple steps, you can significantly reduce the risk of SQL Injection and Cross-Site Scripting (XSS) vulnerabilities in your web application. Regular monitoring, code reviews, and security audits will further enhance your application's security posture.