TASK – 2

1. Set Up OWASP ZAP and Target

- Open OWASP ZAP.
- Configure your browser to use ZAP as a proxy (default is 127.0.0.1:8080).
- Navigate to the web application in your browser to capture traffic.

2. Identify SQL Injection

SQL Injection occurs when malicious SQL commands are inserted into input fields to manipulate the database.

Steps:

- Spider the Application:
    - Right-click on the site in the left panel under Sites and select Attack > Spider to map the application.
- Run Active Scan:
    - Right-click on the target URL in the Sites tab and select Attack > Active Scan.
    - ZAP will send payloads to test for vulnerabilities, including SQL Injection.
- Review Alerts:
    - Open the Alerts tab to check for SQL Injection findings.
    - Look for alerts such as SQL Injection or SQLi.
    - Example Evidence:
        - Payloads like ' OR '1'='1 or '; DROP TABLE users;-- might cause errors or unexpected behavior.
- Verify:
    - Click on the alert for details. It will show:
        - Affected URL and parameter.
        - Exploit evidence, such as SQL error messages in responses.

SQL Injection manipulates database queries through user inputs.

How to Exploit:

1. Locate an input field (e.g., login form, search bar) where SQL Injection was flagged.
2. Test it manually:
    - Enter ' OR '1'='1 or admin' -- into the username or password field.
    - Observe the response: successful login or database errors indicate a vulnerability.

Mitigation Steps:

1. Use Parameterized Queries (Prepared Statements):
    - Avoid dynamically constructing SQL queries using user input.
2. Input Validation:
    - Validate user inputs to ensure they match expected formats (e.g., alphanumeric, specific length).
    - Reject special characters like ', --, ;, etc., if not needed.
3. Database Permissions:
    - Restrict database accounts to minimum privileges.
    - Avoid using admin accounts for application queries.

3. Identify Cross-Site Scripting (XSS)

XSS allows attackers to inject malicious scripts into web pages viewed by other users.

Steps:

- Active Scan for XSS:
    - Ensure ZAP has crawled pages with input fields (e.g., search bars, comment boxes).
    - Right-click the target URL and perform an **Active Scan**.
- Review Alerts:
    - In the **Alerts** tab, look for XSS-related findings, such as:
        - **Persistent XSS**: Script remains stored on the server.
        - **Reflected XSS**: Script is reflected in the response.
    - Example Evidence:
        - Payloads like <script>alert('XSS')</script> or "><svg/onload=alert(1)>.

- **Test in Browser** (Optional):
    - Copy the vulnerable request from ZAP and replay it in the browser to observe the script execution.

XSS injects malicious scripts into web pages, executed by other users.

How to Exploit:

1. Locate an input field (e.g., comment box or search bar) flagged for XSS.
2. Inject a basic script: <script>alert('XSS')</script>
3. Submit the input and observe:
    - If the alert box pops up, the input was executed, confirming XSS

Mitigation Steps:

1. Sanitize and Validate User Input:
    - Remove or escape special characters like <, >, " from inputs before storing or rendering.
    - Use libraries like OWASP **ESAPI** for proper encoding.
2. Output Encoding:
    - Encode user-generated content before displaying it in HTML, JavaScript, or other formats.
3. Content Security Policy (CSP):
    - Implement a CSP header to limit sources for scripts.

4. Identify Cross-Site Request Forgery (CSRF)

CSRF exploits a user's authenticated session to perform unauthorized actions.

Steps:

- Scan for CSRF:
    - Look for forms or requests that perform sensitive actions (e.g., password changes, transactions).
    - Right-click the target URL or form and run an **Active Scan**.
- Review Alerts:
    - Check the **Alerts** tab for vulnerabilities like **Cross-Site Request Forgery**.

    - ZAP flags forms without CSRF tokens or with predictable tokens.
- Verify Evidence:
    - Alerts may show:
        - Lack of anti-CSRF tokens in form submissions.
        - Recommendations to implement random CSRF tokens for protection.

CSRF tricks authenticated users into performing unwanted actions.

How to Exploit:

1. Locate sensitive actions flagged for missing CSRF protection (e.g., password change forms).
2. Create a malicious HTML form to simulate the action:

<form method="POST" action="http://example.com/change-password">

<input type="hidden" name="new_password" value="hacked123">

<input type="submit" value="Submit"> </form>

3. Send the form to the victim while they are logged in to the app.
4. If successful, the victim's password changes without their consent.

Mitigation Steps:

1. **Use CSRF Tokens**:
    - Add a unique token to forms or sensitive requests and validate it on the server.
2. Verify Request Origin:
    - Check the Referer or Origin headers to ensure requests come from trusted sources.
3. Require Authentication:
    - Ensure sensitive actions can only be performed by authenticated users.
    - Re-authenticate users for highly sensitive actions (e.g., password changes).

5. Test and Confirm Findings

- Use ZAP's **"Request Editor"** to replay attacks (e.g., injecting SQL payloads or testing XSS scripts) and verify behavior.
- Use **manual exploration** or **breakpoints** to analyze specific requests and responses.