1. SQL Injection (SQLi)

1.1 Explanation

SQL Injection is a web application vulnerability that occurs when **user-supplied input is directly embedded into SQL queries** without proper validation or sanitization. This allows attackers to alter the logic of database queries, resulting in unauthorized access to data, modification of records, or complete database compromise.

Web applications typically interact with databases to authenticate users, retrieve records, or store information. If the application does not strictly control how user input is processed, the database may interpret malicious input as part of the SQL command.

SQL Injection is ranked among the **OWASP Top 10 vulnerabilities** due to its high impact and ease of exploitation.

1.2 How to Perform SQL Injection (In a Legal Lab Environment)

Step 1: Set Up the Environment

- Install **Kali Linux**.
- Deploy **DVWA (Damn Vulnerable Web Application)** on a local server.
- Start Apache and MySQL services.
- Open DVWA in the browser and complete database setup.
- Set the DVWA security level to **Low** for learning.

Step 2: Identify Injection Points

- Navigate to the **SQL Injection** module in DVWA.

- Locate input fields that request user data (e.g., User ID).
- These fields are potential attack surfaces.

Step 3: Test Input Behavior

- Enter unexpected input instead of normal values.
- Observe application responses such as:
    - Display of multiple records
    - Error messages
    - Abnormal query results
- These behaviors indicate improper input handling.

Step 4: Analyze Query Logic

- Understand how the backend query behaves based on user input.
- Determine whether input directly affects database query execution.

Step 5: Confirm the Vulnerability

- If altering input changes the query result beyond intended behavior, SQL Injection is confirmed.

Step 6: Apply Security Controls

- Implement **prepared statements**.
- Enforce input validation.
- Restrict database privileges.
- Re-test to ensure vulnerability is mitigated.

---

2. Cross-Site Scripting (XSS)

2.1 Explanation

Cross-Site Scripting (XSS) occurs when a web application **accepts user input and displays it in a webpage without proper encoding or sanitization**, allowing execution of malicious scripts in the browser.

XSS attacks target **users**, not servers, and can lead to:

- Session hijacking
- Cookie theft
- Credential compromise
- Website defacement

XSS is categorized into **Stored XSS** and **Reflected XSS**.

2.2 Stored XSS

Explanation

Stored XSS occurs when malicious input is **permanently stored** in the application's database and executed whenever a user accesses the affected page.

How to Perform Stored XSS

Step 1: Identify Persistent Input Fields

- Locate forms that save user input (comments, feedback, messages).

Step 2: Submit Test Data

- Enter normal text to confirm data is stored and displayed.

Step 3: Inject Script Content

- Enter script-like input instead of plain text.
- Submit the form.

Step 4: Reload the Page

- If the browser executes the input, Stored XSS is confirmed.

Step 5: Mitigation

- Encode output before rendering.
- Apply server-side validation.
- Implement **Content Security Policy (CSP)**.

2.3 Reflected XSS

Explanation

Reflected XSS occurs when malicious input is **immediately reflected** in the server response without being stored.

How to Perform Reflected XSS

Step 1: Identify URL Parameters

- Analyze query strings in URLs (search, error messages).

Step 2: Modify Parameters

- Replace normal values with script-like input.

Step 3: Observe Browser Response

- If the script executes instantly, Reflected XSS is present.

Step 4: Mitigation

- Validate and sanitize all inputs.
- Encode output.
- Apply CSP rules.

---

3. Cross-Site Request Forgery (CSRF)

3.1 Explanation

CSRF is an attack where a **logged-in user is tricked into performing an unintended action** on a trusted website without their knowledge.

The attack exploits the fact that browsers automatically include session cookies with every request.

3.2 How to Perform CSRF

Step 1: Identify Sensitive Actions

- Locate actions such as password change, email update, or fund transfer.

Step 2: Analyze Request Structure

- Observe how the request is sent when the action is performed normally.

Step 3: Recreate the Request

- Duplicate the request externally.
- Ensure authentication cookies are still present.

Step 4: Execute the Forged Request

- When the user is logged in, the forged request succeeds without user interaction.

Step 5: Apply Protection

- Implement **CSRF tokens**.
- Verify request origin.
- Use SameSite cookies.

---

4. File Inclusion Vulnerabilities

4.1 Explanation

File Inclusion vulnerabilities occur when a web application dynamically includes files based on user input without validation.

This can lead to:

- Sensitive file disclosure
- Remote code execution
- Complete system compromise

File Inclusion is divided into **Local File Inclusion (LFI)** and **Remote File Inclusion (RFI)**.

4.2 Local File Inclusion (LFI)

How to Perform LFI

Step 1: Identify File Parameters

- Look for parameters controlling file loading.

Step 2: Manipulate File Paths

- Modify file paths to access unintended local files.

Step 3: Observe Output

- If sensitive files are displayed, LFI exists.

4.3 Remote File Inclusion (RFI)

How to Perform RFI

Step 1: Test External File Loading

- Check if external URLs are accepted as file input.

Step 2: Observe Execution

- If remote content is executed, RFI is confirmed.

Mitigation

- Disable remote file inclusion.
- Use strict allowlists.
- Validate all file inputs.

---

5. Burp Suite – Web Application Testing

5.1 Explanation

Burp Suite is a professional web security testing tool used to **intercept, analyze, and modify HTTP/HTTPS traffic** between client and server.

It is widely used for:

- Request analysis
- Parameter manipulation
- Vulnerability discovery

5.2 How to Use Burp Suite

Step 1: Configure Proxy

- Set browser proxy to Burp Suite.
- Enable traffic interception.

Step 2: Capture Requests

- Perform actions in the web application.
- Observe captured HTTP requests.

Step 3: Modify Requests

- Edit parameters manually.
- Forward modified requests to server.

Step 4: Automated Testing

- Use Burp Intruder to test multiple payloads.
- Analyze server responses.