Project 1:

Title:
DVWA Security Level Comparison — SQL Injection & Stored XSS

Objective:

Compare DVWA behavior and implemented defenses across security levels (Low, Medium, High, Impossible) for two vulnerabilities (SQL Injection, Stored XSS). Document payloads that work at Low, test them at higher levels, observe differences, and identify what defense mechanism was added.

Chosen vulnerabilities

SQL Injection (SQLi) — classic injection into login/search or user ID field.

Stored Cross-Site Scripting (XSS) — stored XSS via comment/feedback fields.

Test environment / setup

Use local DVWA (Kali VM or similar). Ensure DVWA security level is changeable in the web UI.

Enable logging (browser devtools Network/Console + server logs).

For each test case: record DVWA security level, form used, exact payload, screenshot of response (or error), and any server-side/log evidence. Save all screenshots for the report.

Methodology (repeat for each vuln & payload)

Set DVWA to Low.

Enter payload; record behavior (success, error, output shown). Save screenshot.

Switch to Medium, High, Impossible and repeat same payloads (do not modify payloads). Record differences and failures.

For each level where payload no longer works, try to infer the defense (e.g., input sanitization, escaping, prepared statements). Note server response codes, error messages, and page differences.

Produce a results table summarizing which payloads worked where and note defense likely used.

Vulnerability 1 — SQL Injection
Common DVWA target

Login form or “User ID” lookup (e.g., id=1 GET param or id POST form).

Example payloads to try (Low)

' OR '1'='1 — a classic authentication bypass.

1' OR '1'='1' -- — comment remainder (if quotes used).

admin' -- (simple).

What you should see at Low

Login bypass (you’re logged in without valid credentials) OR the query returns all rows or unexpected data displayed on page.

Error messages may include SQL syntax — indicates lack of error suppression.

Behavior at higher levels (typical DVWA behavior & defenses)

Medium:

DVWA often applies basic input filtering/escaping — e.g., strips some special characters or replaces quotes, or applies naive regex that blocks ' or --. This will often break basic payloads. You might get an input validation message or sanitized output that shows the quote escaped.
Defense: simple input sanitization/blacklisting or basic escaping.

High:

DVWA may do stronger validation or use type checks (e.g., cast id to integer) or more robust escaping. Payloads fail; you may see normal page or an error without SQL details.
Defense: input validation, strong escaping, or integer-casting (type enforcement).

Impossible:

DVWA replaces vulnerable code with parameterized queries (prepared statements) or ORM-style safe calls. Injection attempts fail completely and input is treated as data.
Defense: parameterized queries / prepared statements (strong mitigation).

How to identify the defense

If quotes are removed/escaped in the page source → escaping or sanitization.

If passing id=abc returns error or is rejected while id=1 works → type casting / validation.

If payload no longer affects SQL output at all and no errors → prepared statements or ORM.

Vulnerability 2 — Stored XSS (Cross-Site Scripting)
Common DVWA target

Comment/feedback form, profile bio, message box that displays stored content.

Example payloads to try (Low)

<script>alert("xss")</script> — classic demonstration (shows an alert when page with stored comment is loaded).

<img src=x onerror=alert("xss")> — alternative event-based payload.

What you should see at Low

The alert box appears when viewing the page with the stored comment (i.e., script executes). Page source shows raw script tags — no encoding.

Behavior at higher levels (typical DVWA behavior & defenses)

Medium: DVWA usually applies basic output encoding or strips <script> tags or certain event attributes. The raw script will be escaped to <script>alert...</script> or stripped. The payload will not execute.
Defense: output encoding (HTML-escape) or basic tag stripping.

High: DVWA may use a stricter allowlist that removes most HTML that can run scripts (removes on* attributes, <script>, <img onerror>, etc.) or performs server-side sanitization via a library (e.g., removing all tags except safe ones).
Defense: robust sanitization + stricter encoding.

Impossible: DVWA may implement Content Security Policy (CSP), HttpOnly flags for cookies, and full output encoding or sanitization — making stored XSS practically impossible in that view.
Defense: CSP, comprehensive output encoding & sanitization.

How to identify the defense

Inspect page source: if payload appears escaped (<script>) then output encoding is used.

If payload removed entirely (no tags), input sanitization was applied.

If payload present but not executing and browser console shows CSP violation — CSP is present.