

# 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
```

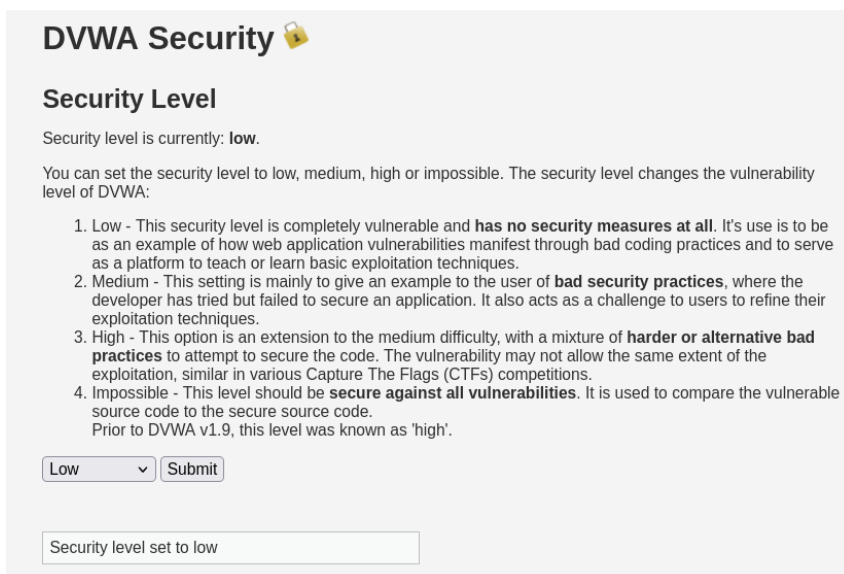
- **Login Credentials:**
  - Username: admin
  - Password: password

## Methodology


### 1. SQL Injection

#### Low Security

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



The screenshot shows the 'DVWA Security' page with a yellow padlock icon. The 'Security Level' section indicates the current level is 'low'. Below this, a paragraph explains that the security level can be set to low, medium, high, or impossible, and that it changes the vulnerability level of DVWA. A list of four levels is provided: 1. Low (completely vulnerable, no security measures), 2. Medium (example of bad security practices), 3. High (extension to medium difficulty with harder or alternative bad practices), and 4. Impossible (secure against all vulnerabilities). At the bottom, there is a dropdown menu set to 'Low' and a 'Submit' button. A message box at the very bottom states 'Security level set to low'.

**DVWA Security** 

**Security Level**

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.

Security level set to low

- Opened **SQL Injection** page.

- Entered payload:

`1' OR '1'='1`

- Result:** All user data was displayed.

## Vulnerability: SQL Injection

User ID:

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

## Medium Security

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

## DVWA Security

### Security Level

Security level is currently: **medium**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.

Security level set 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:

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**.

### DVWA Security

#### Security Level

Security level is currently: **high**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.

High

Submit

Security level set to high

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

## Vulnerability: SQL Injection

Click [here to change your ID](#).

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

SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) — Mozilla Firefox

127.0.0.1:42001/vulnerabilities/sqli/session-input.php# 120% ☆ ≡

Session ID: 1' OR '1'='1

1' OR '1'='1

Submit

Close

## 2. Cross-Site Scripting (Reflected XSS)

### Low Security

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

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

### Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

- **Result:** Browser displayed an alert “XSS”.



### Medium Security

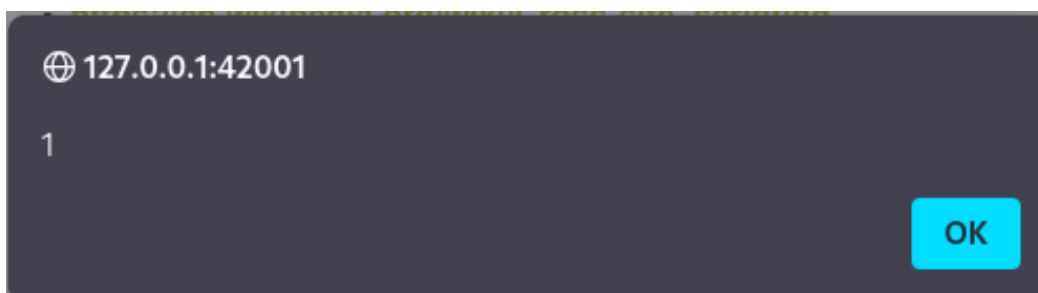
- Changed DVWA security level to **Medium**.
- Script tags were filtered; the previous payload no longer triggered an alert.
- Entered alternative payload:

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

### Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

- **Result:** Browser displayed an alert “1”.

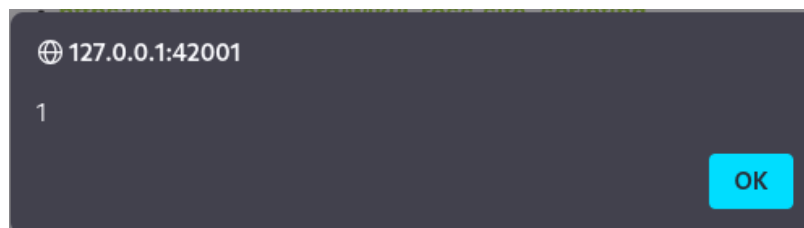


## High Security

- 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: Reflected Cross Site Scripting (XSS)

What's your name?



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.