DVWA Security Testing Report — SQLi, XSS, CSRF

SUMMARY

This document contains a compact but complete security testing report for DVWA covering:

1. SQL Injection (SQLi)
2. Cross-Site Scripting (Reflected & Stored XSS)
3. Cross-Site Request Forgery (CSRF)

All testing and attack code must be used only in an isolated lab (Kali / DVWA). Do not run these against third-party or production sites.

LAB ENVIORNMENT

- Host: Kali Linux (VM) running DVWA served by Apache on http://127.0.0.1/DVWA
- DB: MariaDB (local)
- Browser: Firefox in Kali
- Attacker files hosted on simple HTTP server (Python http.server) or in /var/www/html for convenience.

1. SQL Injection (SQLi) —

An attacker injects SQL code through user input (forms, URLs) so the database runs unintended queries.

Consequences: data theft, modification, or deletion.

Fix: use prepared statements (parameterized queries) and validate input.

1.1 Target page

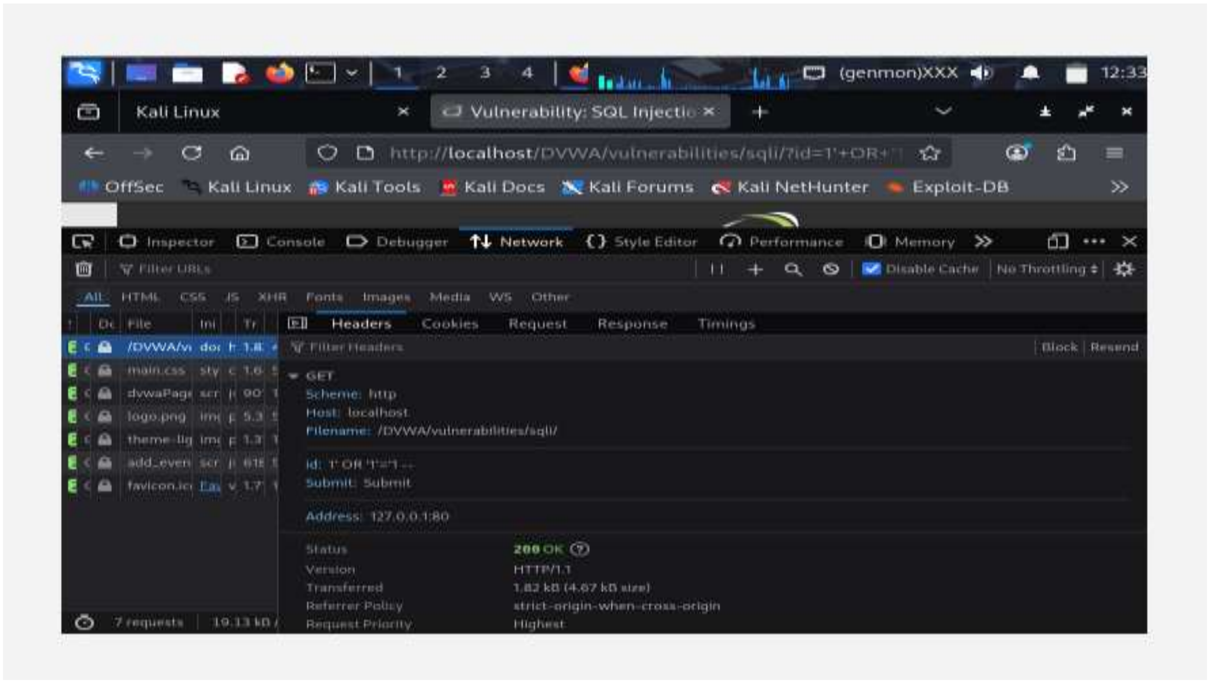
- DVWA SQL Injection module (Security = Low)

1.2 Steps (extract usernames/passwords)

1. Open DVWA → Vulnerabilities → SQL Injection.

2. Set Security to Low (DVWA → DVWA Security).
3. In the input field (e.g., “User ID”), submit a value to enumerate columns:
    - Test boolean injection: 1' OR '1'='1 (observe behavior)

1.3 Evidence to capture (screenshots)



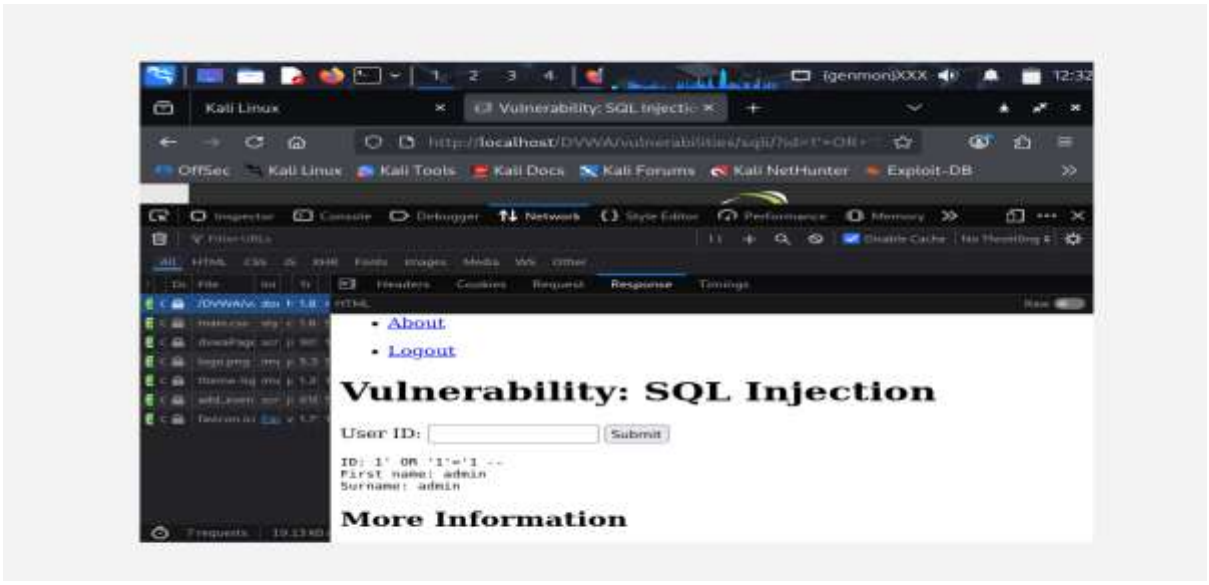
```
GET
Scheme: http
Host: localhost
Filename: /DVWA/vulnerabilities/sqli/

id: 1' OR '1'='1 --
Submit: Submit

Address: 127.0.0.1:80

Status              200 OK
Version             HTTP/1.1
Transferred         1.82 kB (4.67 kB size)
Referrer Policy     strict-origin-when-cross-origin
Request Priority    Highest
```

SQLi header and response



```
Vulnerability: SQL Injection

User ID: [          ] [Submit]

ID: 1' OR '1'='1 --
First name: admin
Surname: admin

More Information
```

1.4 Mitigation (Prepared Statements)

Prevents user input from being interpreted as SQL code.

PHP (mysqli) example — vulnerable code replaced with prepared statement:

```
// vulnerable: $query = "SELECT * FROM users WHERE id = $id";
$stmt = mysqli_prepare($GLOBALS['___mysqli_ston'], 'SELECT first_name, last_n
ame FROM users WHERE user_id = ? LIMIT 1');
mysqli_stmt_bind_param($stmt, 'i', $id);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $first_name, $last_name);
mysqli_stmt_fetch($stmt);
mysqli_stmt_close($stmt);
```

2. Cross-Site Scripting (Reflected & Stored XSS) -

2.1 Reflected XSS —

Malicious JavaScript is sent in a request (e.g., query string) and immediately reflected into the page without encoding, executing in the victim's browser.

Consequences: session theft, page defacement, phishing.

Fix: HTML-encode output (e.g., htmlspecialchars) and apply a strong Content Security Policy (CSP).

2.2 Stored XSS —

Malicious script is saved on the server (database, comments, guestbook) and served later to visitors. More dangerous than reflected XSS because it affects all viewers.

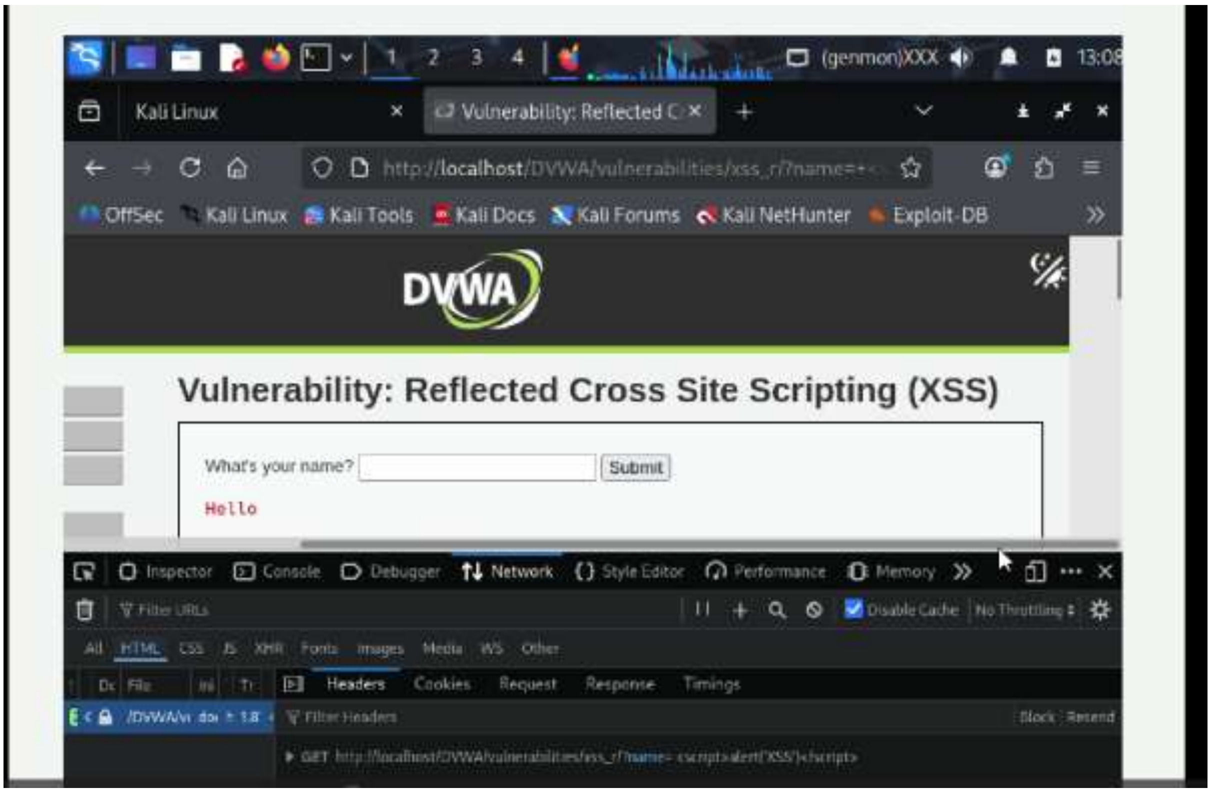
Fix: encode on output, sanitize input when appropriate, and use CSP.

2.1.1 Reflected XSS (vulnerable: Low)

Steps

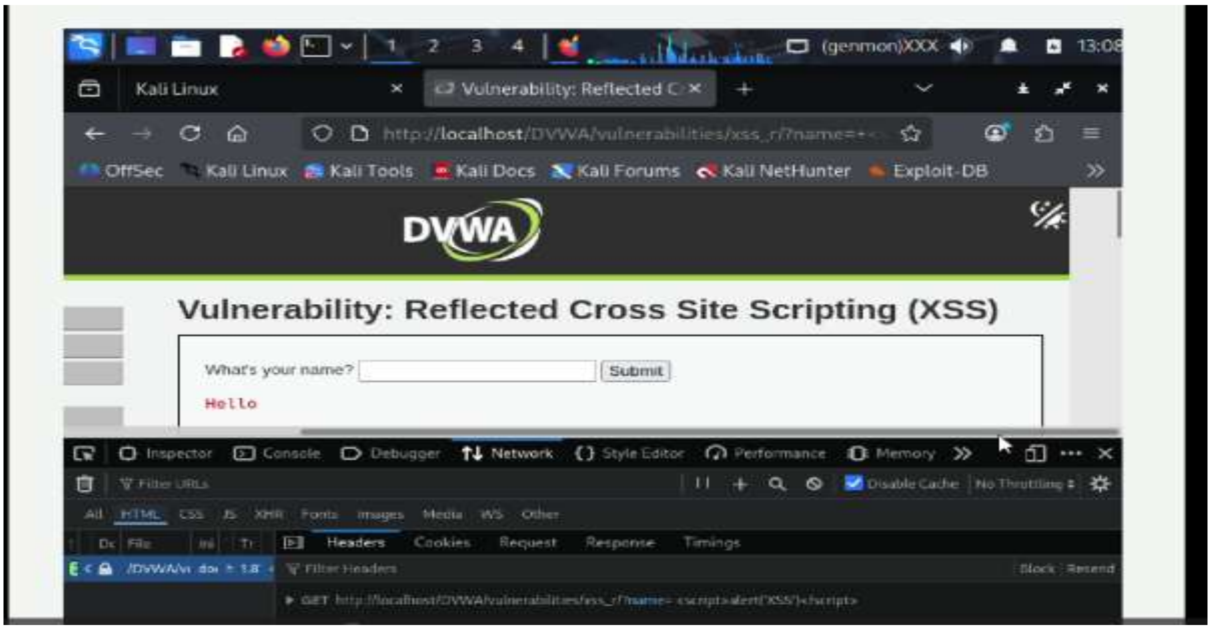
1. Open DVWA → Vulnerabilities → Reflected XSS.
2. Security = Low.
3. In the "What's your name?" field submit: <script>alert('XSS')</script>
4. Observe a JavaScript alert() popup — proof of reflected XSS.

2.1.2 Evidence to capture



```
▸ GET http://localhost/DVWA/vulnerabilities/xss_r/?name= <script>alert('XSS')</script>
```

XSS-Reflected headers



```
▸ GET http://localhost/DVWA/vulnerabilities/xss_r/?name= <script>alert('XSS')</script>
```

XSS-Reflected after mitigation

2.1.3 Mitigation

- On output encode user data with htmlspecialchars() in PHP:

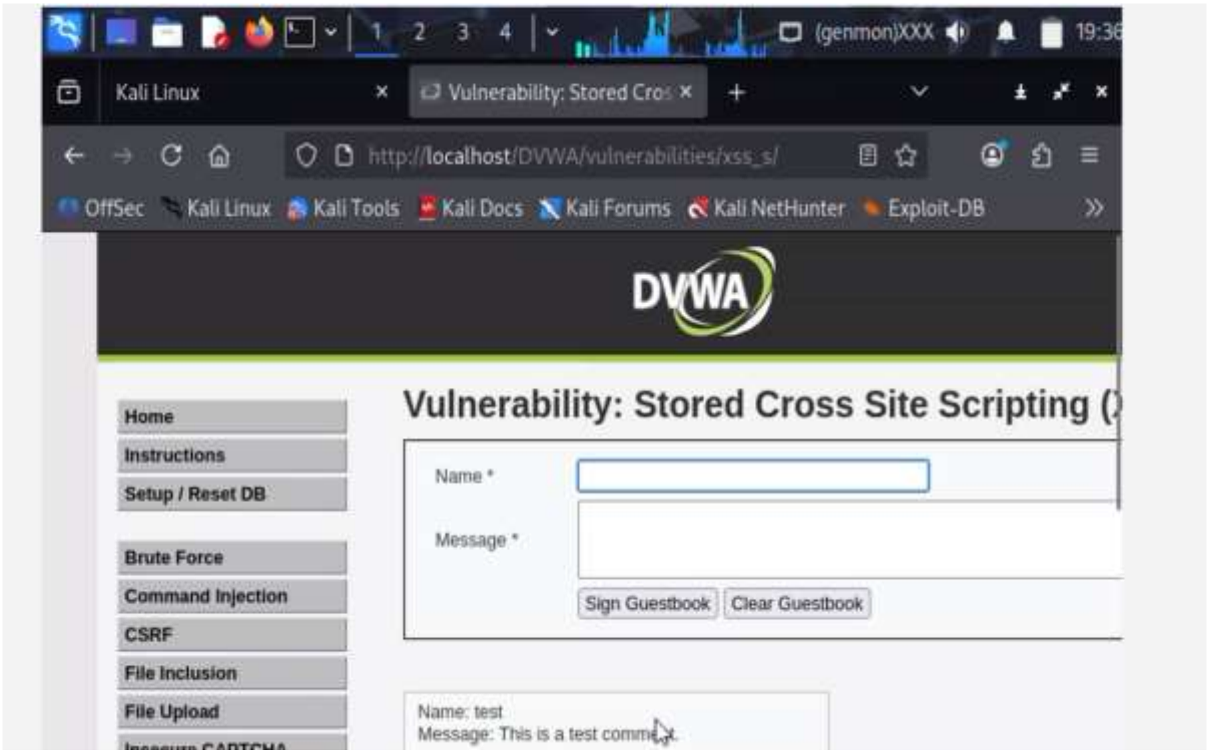
- $clean = htmlspecialchars($user_input, ENT_QUOTES | ENT_HTML5, 'UTF-8');
  echo "<div> Hello {$clean} </div>";

2.2.1 Stored XSS (guestbook)

Steps

1. DVWA → Vulnerabilities → XSS (Stored) → Security = Low.
2. Submit the message: <script>alert('STORED XSS')</script> in the message box.
3. Visit the page that displays guestbook entries — you should see the popup executed for viewers.

2.2.2Evidence to capture



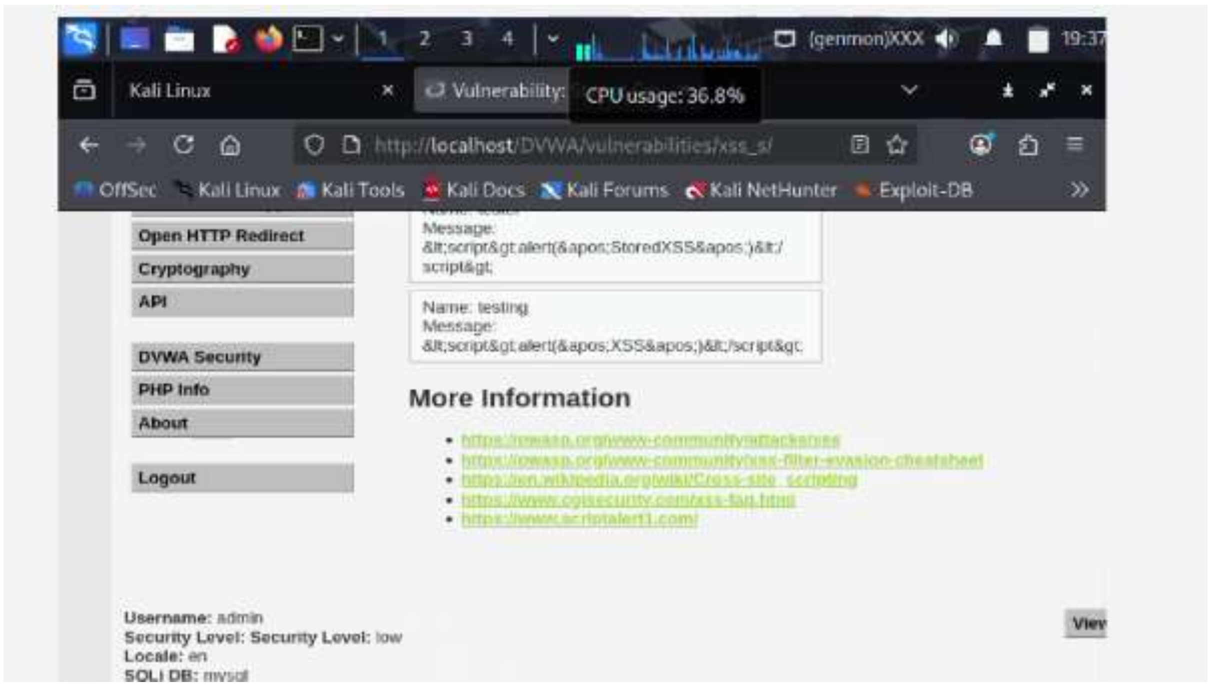
Vulnerability: Stored Cross Site Scripting (

Name *

Message *

Name: test
Message: This is a test comment.

XSS-Stored



Message:
&lt;script&gt;alert(&apos;StoredXSS&apos;)&lt;/script&gt;

Name: testing
Message:
&lt;script&gt;alert(&apos;XSS&apos;)&lt;/script&gt;

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/

Username: admin
Security Level: Security Level: low
Locale: en
SQLi DB: mysql

XSS-Stored after mitigation

2.2.3 Mitigation

- On input or output, encode with htmlspecialchars() as above.
- When writing to DB, escape safely with prepared statements to avoid SQLi on storage operations.
- Example fixed PHP snippet for storing and showing comment:

```
// store
$message = $_POST['mtxMessage'] ?? '';
$name = $_POST['txtName'] ?? '';
$message_db = mysqli_real_escape_string($GLOBALS['___mysqli_ston'], $mess
age);
$name_db = mysqli_real_escape_string($GLOBALS['___mysqli_ston'], $name);
$query = "INSERT INTO guestbook (comment, name) VALUES ('$message_db', '$
name_db')";
// display
echo '<div>' . htmlspecialchars($name, ENT_QUOTES|ENT_HTML5,'UTF-8') . '<b
r>' . htmlspecialchars($comment, ENT_QUOTES|ENT_HTML5,'UTF-8') . '</div>';
```

4. CROSS- SITE REQUEST FORGERY (CSRF)

CSRF (Cross-Site Request Forgery) is an attack where a malicious website tricks a logged-in user into performing an action on another site without their co nsent. For example, changing a password in DVWA without the user's knowle dge.

4.1 Steps:

1. Logged in to DVWA as admin (victim user).

2. Created a malicious HTML page containing an auto-submitting form that sends a password change request to DVWA.

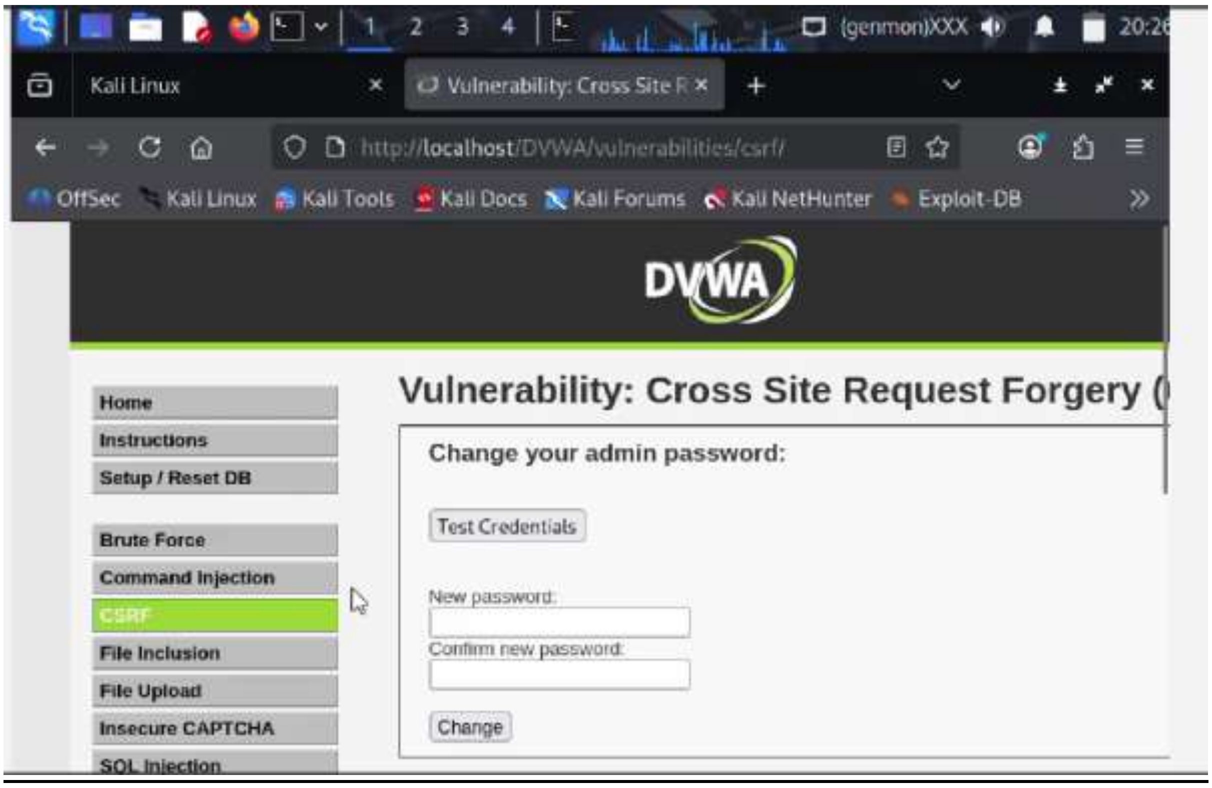
3. Hosted the HTML file on a local Python server (http.server).

4. Opened the malicious page in the victim's browser while logged into DVWA.

5. The password was changed successfully (in Low security level).

6. When tested in 'High' level, the attack failed because a valid CSRF token was required.

4.2 EVIDENCE TO CAPTURE

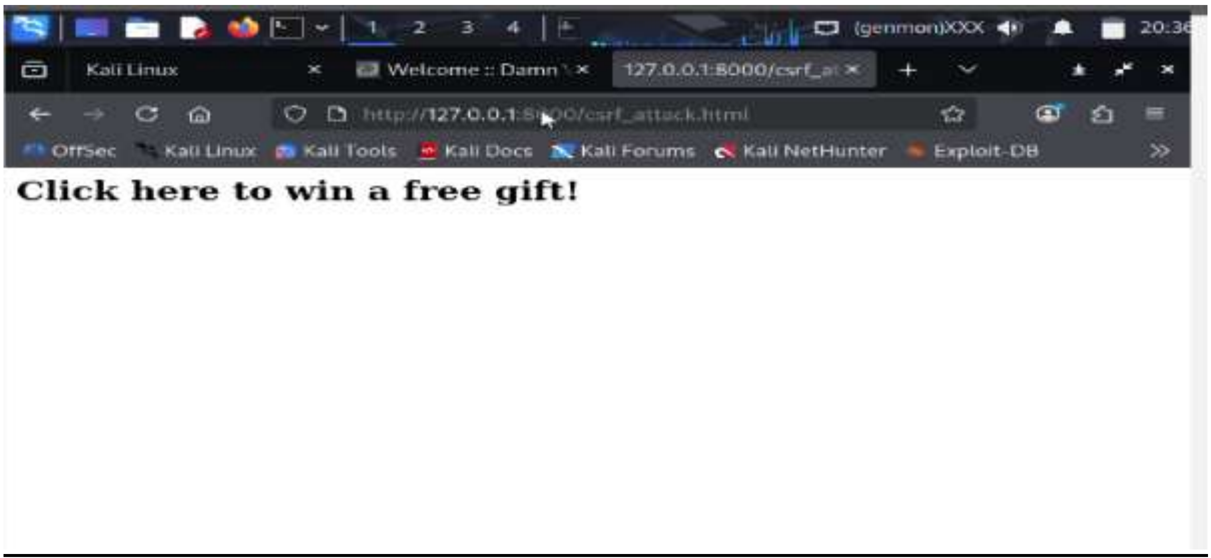


Vulnerability: Cross Site Request Forgery (

Change your admin password:

New password:

Confirm new password:


Click here to win a free gift!

4.3 Mitigation:

- Implemented token-based protection: Each form submission now requires a unique token validated by the server.
- Use of SameSite cookies and proper session management prevents unauthorized cross-origin requests.
- Educated users to avoid clicking on suspicious links or pages while logged into sensitive accounts.
- This confirms that CSRF protection in DVWA works as expected at higher security levels.