Future Interns — Task 1
Web Application Security Testing

Name: Sasi Kumar Medabalimi

Summary

This report summarizes the findings from a web application security test performed on a local DVWA instance. Testing used a mix of automated scanning (OWASP ZAP) and manual verification for highconfidence proofofconcept (PoC). The goal: identify real vulnerabilities, map them to OWASP Top 10 categories, provide impact/risk, and suggest remediation.

Target: DVWA (local lab via XAMPP)
Tools: OWASP ZAP, Browser (proxy), DVWA modules

Scope & Methodology

Scope: Local DVWA application only (XAMPP). Testing tools: OWASP ZAP, Firefox(browser), DVWA modules.

Methodology:
 Reconnaissance: Identify input points (forms, parameters).
 Automated scan: Run ZAP spider + active scan (local lab only).
 Manual testing: SQLi, XSS (reflected + stored), CSRF, Command Injection.
 Evidence collection: screenshots, raw request/response logs, ZAP report.

Findings (Summary Table)

Finding	OWASP Category	Impact	Status / Evidence(uploaded at the end of this file)
SQL Injection	A03 Injection	High	sqli_input_payload & result; zap_sqli_request.txt
Reflected XSS	A07 XSS	MediumHigh	xss(r)_input_payload & result; zap_xss_reflected_request.txt
Stored XSS	A07 XSS	High	xss(s)_input_payload & result; zap_xss_stored_request.txt
CSRF	A08 CSRF	Medium	csrf_input_payload & result;

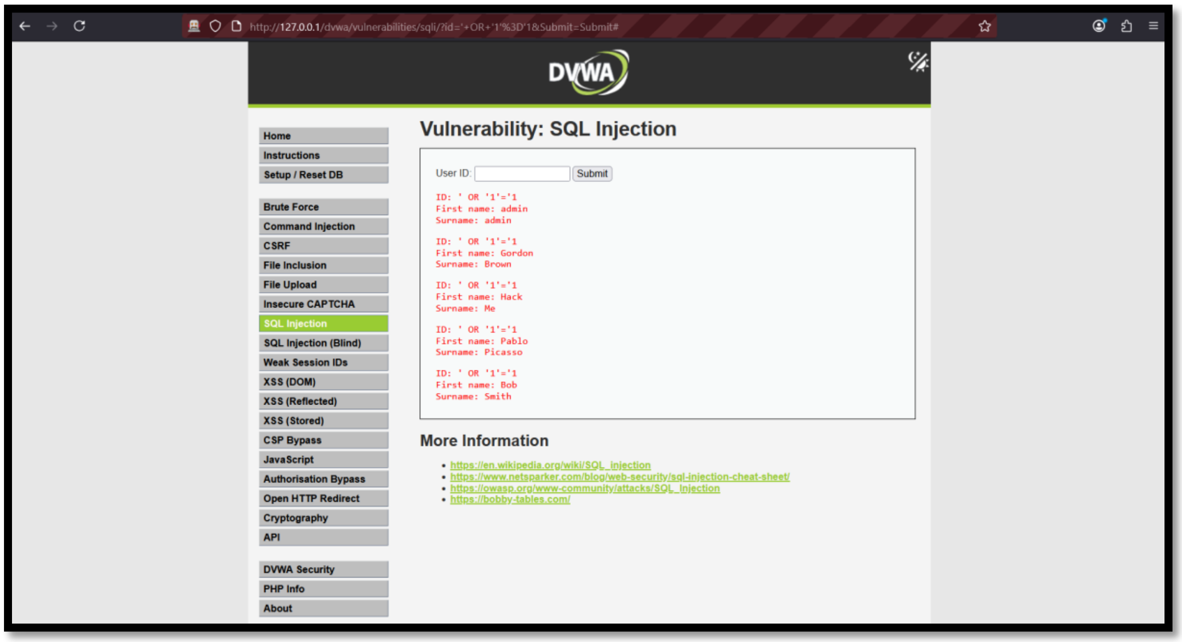
			zap_csrf_original_request.txt
Command Injection	A03/A10 Command Injection	High	cmd_injection_input_payload & result; zap_cmd_request.txt

Detailed Findings

Finding 1 — SQL Injection (A03)

Summary: An SQL injection vulnerability exists in the `id` parameter of `/dvwa/vulnerabilities/sqli/` allowing attacker input to modify SQL queries.

Evidence :



Vulnerability: SQL Injection

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

ID: ' OR '1'='1
First name: Gordon
Surname: Brown

ID: ' OR '1'='1
First name: Hack
Surname: Me

ID: ' OR '1'='1
First name: Pablo
Surname: Picasso

ID: ' OR '1'='1
First name: Bob
Surname: Smith
```

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/

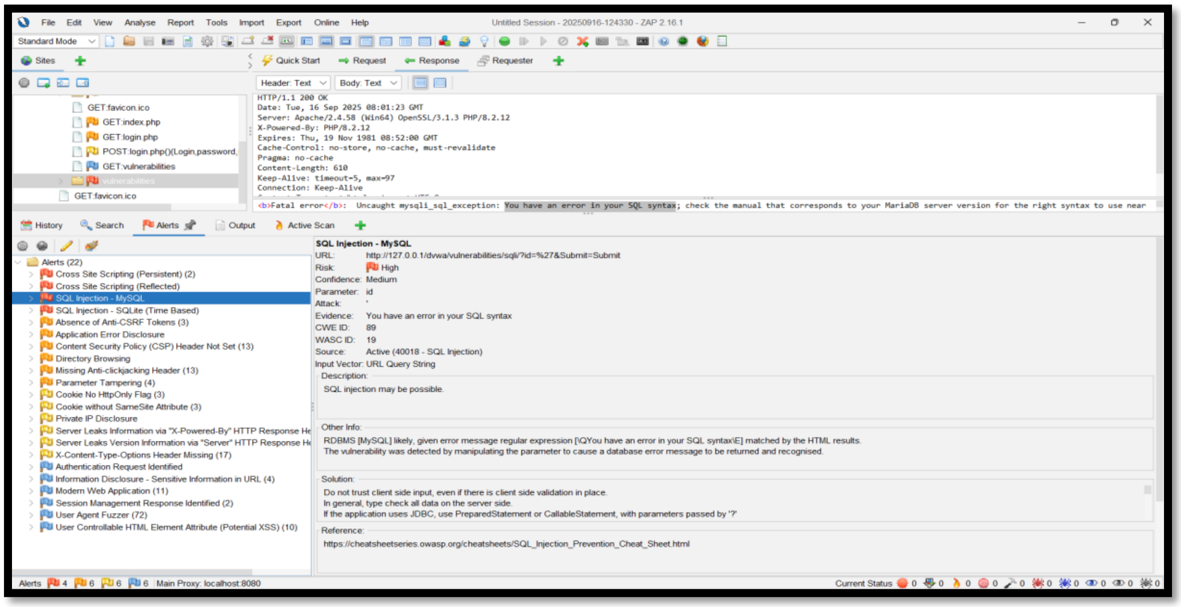
Impact: High

Reproduction Steps:

- Open DVWA → SQL Injection module.
- Submit baseline input `1` and observe normal output.
- Submit payload: `1' OR '1'='1' ` and observe modified output / error message.
- Capture screenshot and save raw request/response from ZAP.

Recommendation: Use parameterized queries, input validation, leastprivileged DB user, and disable verbose DB error messages.

Sqli_Scan_report :



```
GET:favicon.ico
GET:index.php
GET:login.php
POST:login.php()(Login,password,
GET:vulnerabilities
vulnerabilities
GET:favicon.ico
```

```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2025 08:01:23 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12
X-Powered-By: PHP/8.2.12
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 610
Keep-Alive: timeout=5, max=97
Connection: Keep-Alive
```

```
<b>Fatal error</b>:  Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near
```

```
Alerts (22)
  Cross Site Scripting (Persistent) (2)
  Cross Site Scripting (Reflected)
  SQL Injection - MySQL
  SQL Injection - SQLite (Time Based)
  Absence of Anti-CSRF Tokens (3)
  Application Error Disclosure
  Content Security Policy (CSP) Header Not Set (13)
  Directory Browsing
  Missing Anti-clickjacking Header (13)
  Parameter Tampering (4)
  Cookie No HttpOnly Flag (3)
  Cookie without SameSite Attribute (3)
  Private IP Disclosure
  Server Leaks Information via "X-Powered-By" HTTP Response He
  Server Leaks Version Information via "Server" HTTP Response He
  X-Content-Type-Options Header Missing (17)
  Authentication Request Identified
  Information Disclosure - Sensitive Information in URL (4)
  Modern Web Application (11)
  Session Management Response Identified (2)
  User Agent Fuzzer (72)
  User Controllable HTML Element Attribute (Potential XSS) (10)
```

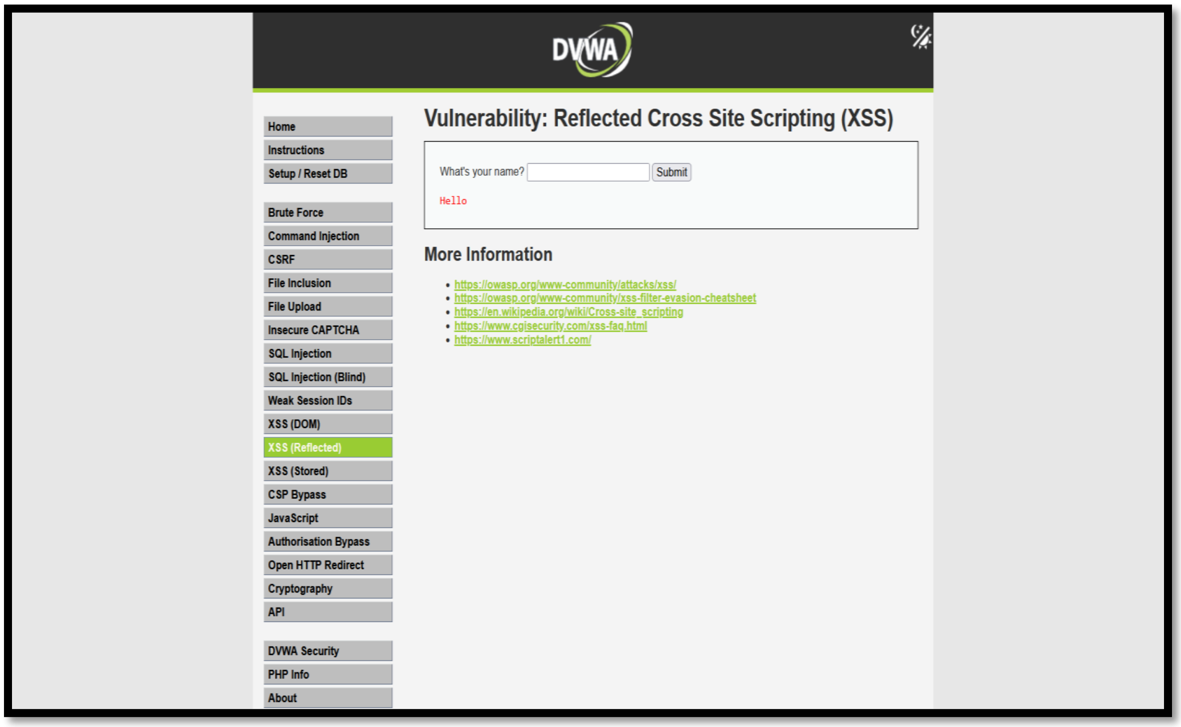
```
SQL Injection - MySQL
URL:          http://127.0.0.1/dvwa/vulnerabilities/sqli/?id=%27&Submit=Submit
Risk:         High
Confidence:   Medium
Parameter:    id
Attack:       '
Evidence:     You have an error in your SQL syntax
CWE ID:       89
WASC ID:      19
Source:       Active (40018 - SQL Injection)
Input Vector: URL Query String
Description:
 SQL injection may be possible.
Other Info:
 RDBMS [MySQL] likely, given error message regular expression [\QYou have an error in your SQL syntax\E] matched by the HTML results.
 The vulnerability was detected by manipulating the parameter to cause a database error message to be returned and recognised.
Solution:
 Do not trust client side input, even if there is client side validation in place.
 In general, type check all data on the server side.
 If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'
Reference:
 https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
```

```
Alerts 4 6 6 6 Main Proxy: localhost:8080
Current Status 0 0 0 0 0 0 0 0 0 0
```

Finding 2 — Reflected XSS (A07)

Summary: A reflected XSS vulnerability where user input is reflected unsanitized in the response.

Evidence: [xss(r)_input_payload & result]



Vulnerability: Reflected Cross Site Scripting (XSS)

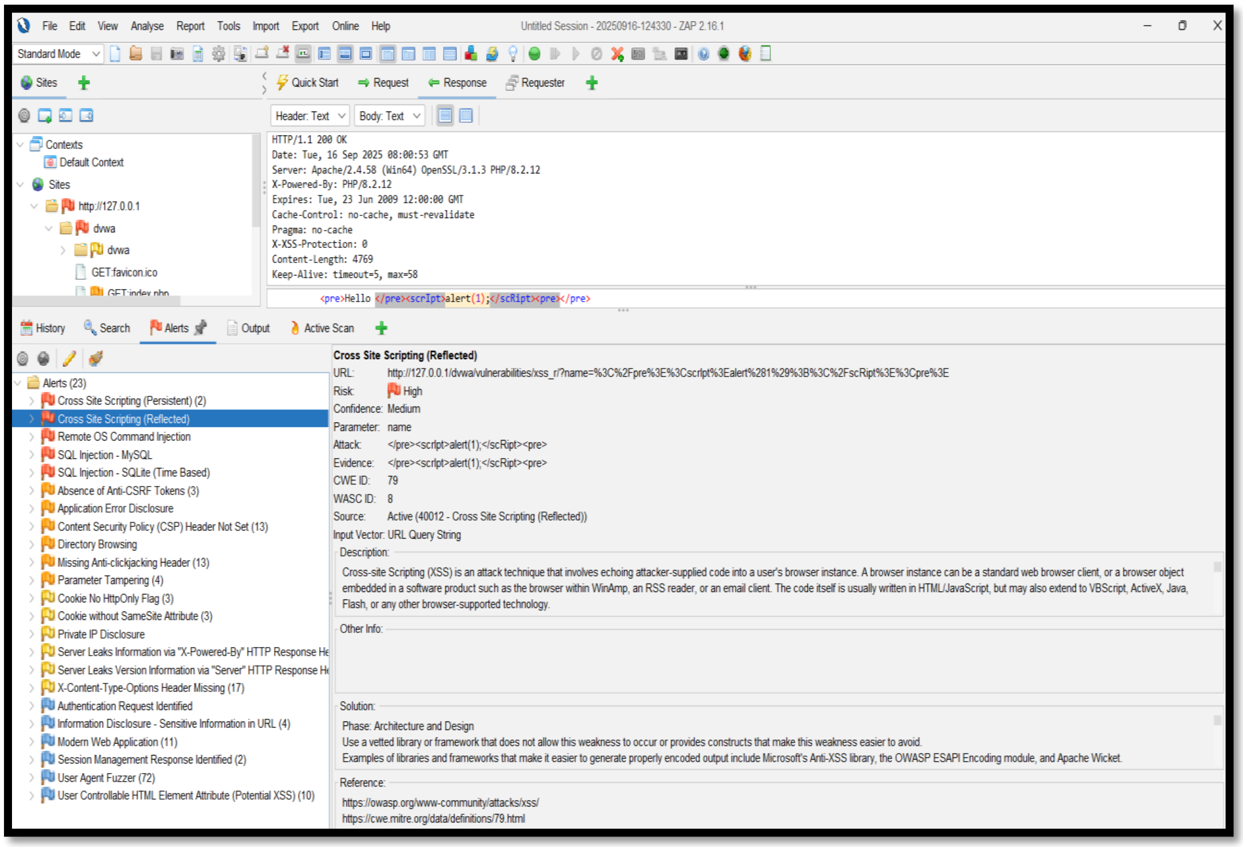
What's your name?

Hello

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/

xss(r)_Scan_report :



```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2025 08:00:53 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12
X-Powered-By: PHP/8.2.12
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
X-XSS-Protection: 0
Content-Length: 4769
Keep-Alive: timeout=5, max=58
```

```
<pre>Hello </pre><scrIpt>alert(1);</scRipt><pre></pre>
```

Alerts (23)
- Cross Site Scripting (Persistent) (2)
- Cross Site Scripting (Reflected)
- Remote OS Command Injection
- SQL Injection - MySQL
- SQL Injection - SQLite (Time Based)
- Absence of Anti-CSRF Tokens (3)
- Application Error Disclosure
- Content Security Policy (CSP) Header Not Set (13)
- Directory Browsing
- Missing Anti-clickjacking Header (13)
- Parameter Tampering (4)
- Cookie No HttpOnly Flag (3)
- Cookie without SameSite Attribute (3)
- Private IP Disclosure
- Server Leaks Information via "X-Powered-By" HTTP Response He
- Server Leaks Version Information via "Server" HTTP Response He
- X-Content-Type-Options Header Missing (17)
- Authentication Request Identified
- Information Disclosure - Sensitive Information in URL (4)
- Modern Web Application (11)
- Session Management Response Identified (2)
- User Agent Fuzzer (72)
- User Controllable HTML Element Attribute (Potential XSS) (10)

Cross Site Scripting (Reflected)

URL:	http://127.0.0.1/dvwa/vulnerabilities/xss_r/?name=%3C%2Fpre%3E%3Cscrlpt%3Ealert%281%29%3B%3C%2FscRipt%3E%3Cpre%3E
Risk:	High
Confidence:	Medium
Parameter:	name
Attack:	</pre><scrlpt>alert(1);</scRipt><pre>
Evidence:	</pre><scrlpt>alert(1);</scRipt><pre>
CWE ID:	79
WASC ID:	8
Source:	Active (40012 - Cross Site Scripting (Reflected))
Input Vector:	URL Query String

Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:

Solution:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Reference:
https://owasp.org/www-community/attacks/xss/
https://cwe.mitre.org/data/definitions/79.html

Impact: MediumHigh

Reproduction Steps:

- Open DVWA → XSS (Reflected).
- Submit payload: `<script>alert('XSS')</script>` (or use `<img src=x onerror="alert('XSS')">`).
- Observe script execution or reflected payload in response.

Recommendation: Perform contextual output encoding, implement CSP, and sanitize inputs.

Finding 3 — Stored XSS (A07)

Summary: Stored XSS where injected content is persisted and executed when rendered to users.

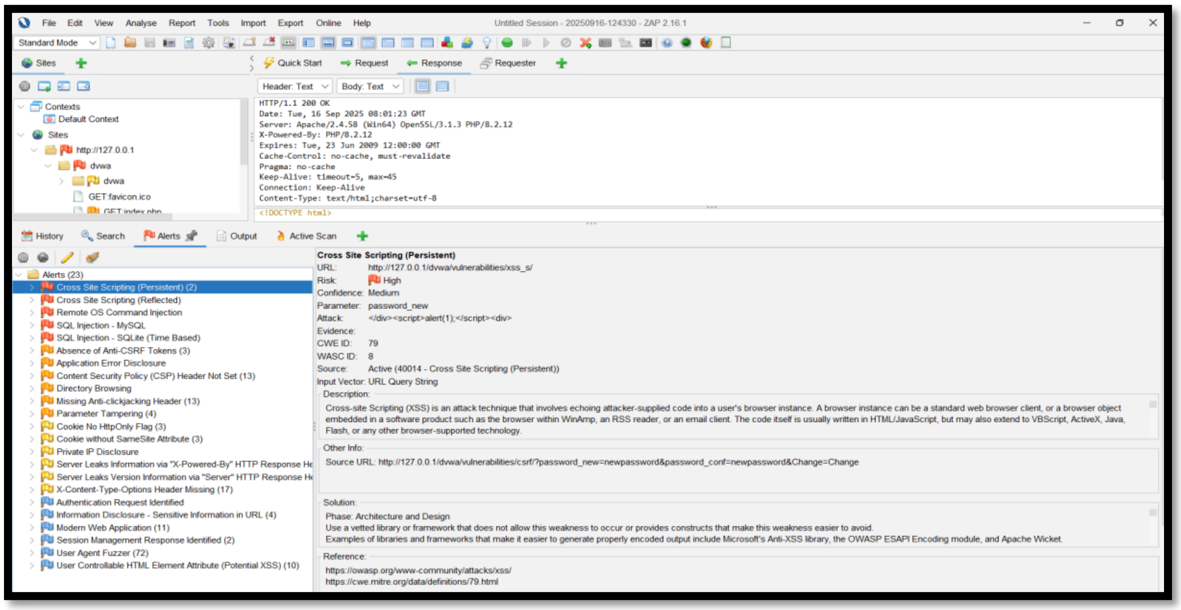
Impact: High

Reproduction Steps:

- Open DVWA → XSS (Stored).
- Post payload into comment/feedback: `<img src=x onerror="alert('XSS')">`.

- Visit page that displays stored entries and verify execution.

xss(s)_Scan_report :



Contexts
Default Context
Sites
http://127.0.0.1
dvwa
dvwa
GET:favicon.ico
GET:index.php

```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2025 08:01:23 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12
X-Powered-By: PHP/8.2.12
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Keep-Alive: timeout=5, max=45
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
<!DOCTYPE html>
```

Alerts (23)
- Cross Site Scripting (Persistent) (2)
- Cross Site Scripting (Reflected)
- Remote OS Command Injection
- SQL Injection - MySQL
- SQL Injection - SQLite (Time Based)
- Absence of Anti-CSRF Tokens (3)
- Application Error Disclosure
- Content Security Policy (CSP) Header Not Set (13)
- Directory Browsing
- Missing Anti-clickjacking Header (13)
- Parameter Tampering (4)
- Cookie No HttpOnly Flag (3)
- Cookie without SameSite Attribute (3)
- Private IP Disclosure
- Server Leaks Information via "X-Powered-By" HTTP Response He
- Server Leaks Version Information via "Server" HTTP Response He
- X-Content-Type-Options Header Missing (17)
- Authentication Request Identified
- Information Disclosure - Sensitive Information in URL (4)
- Modern Web Application (11)
- Session Management Response Identified (2)
- User Agent Fuzzer (72)
- User Controllable HTML Element Attribute (Potential XSS) (10)

Cross Site Scripting (Persistent)

URL:	http://127.0.0.1/dvwa/vulnerabilities/xss_s/
Risk:	High
Confidence:	Medium
Parameter:	password_new
Attack:	</div><script>alert(1);</script><div>
Evidence:	
CWE ID:	79
WASC ID:	8
Source:	Active (40014 - Cross Site Scripting (Persistent))
Input Vector:	URL Query String

Description:
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Other Info:
Source URL: http://127.0.0.1/dvwa/vulnerabilities/csrf/?password_new=newpassword&password_conf=newpassword&Change=Change

Solution:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

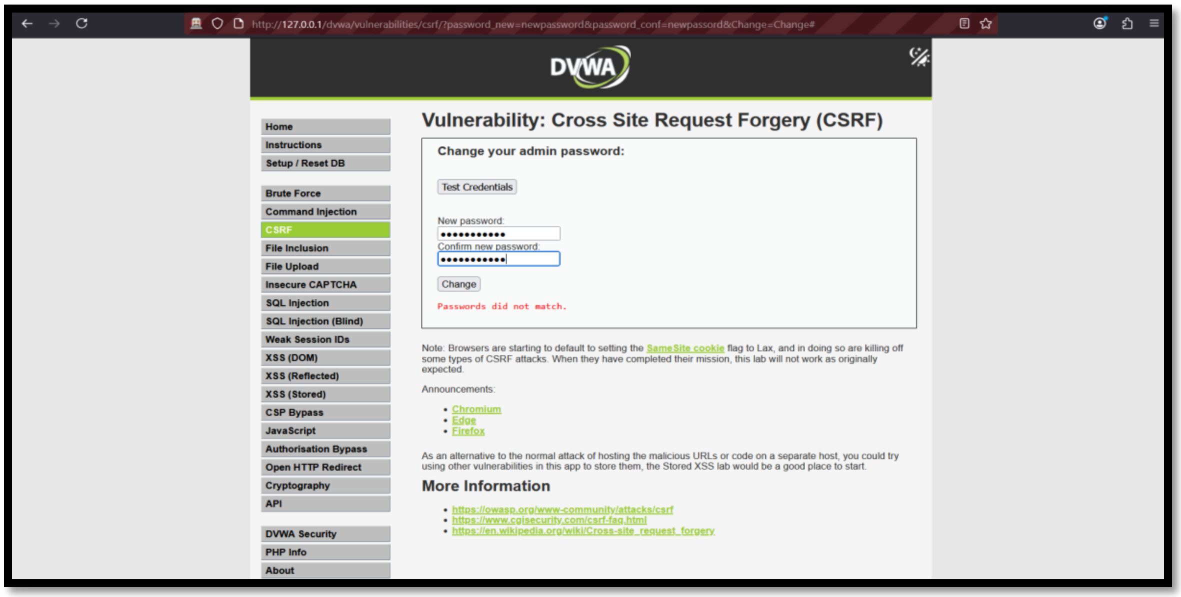
Reference:
https://owasp.org/www-community/attacks/xss/
https://cwe.mitre.org/data/definitions/79.html

Recommendation: Sanitize and encode stored inputs on output; use CSP and HttpOnly cookies.

Finding 4 — CSRF (A08)

Summary: CSRF vulnerability on statechanging endpoint allowing actions without unpredictable CSRF token.

Evidence: [csrf_input_payload & result]



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password: ●●●●●●●●●●●
Confirm new password: ●●●●●●●●●●●

Passwords did not match.

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:
- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

More Information
- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

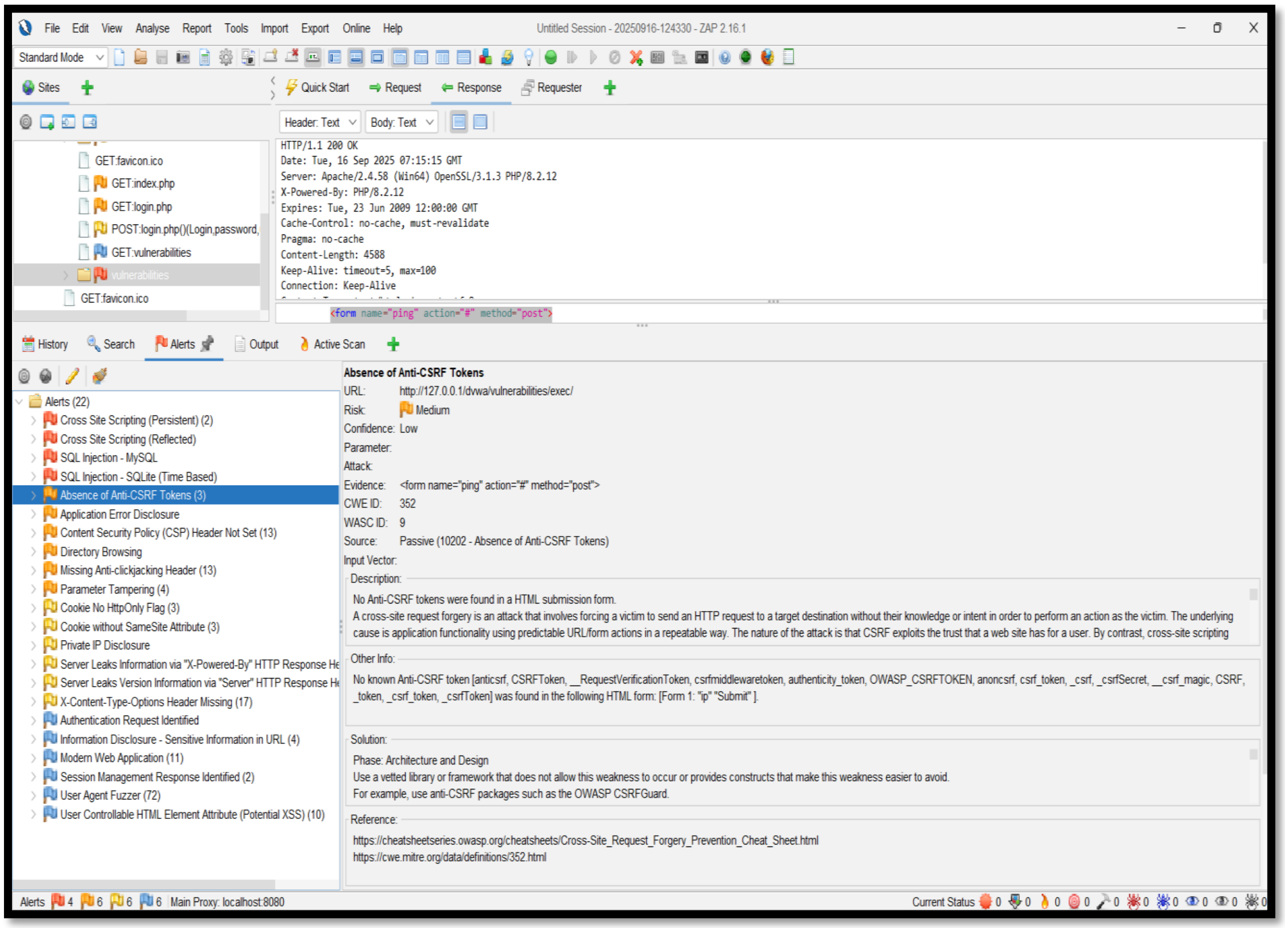
Impact: Medium

Reproduction Steps:

- Capture original statechanging request via ZAP while logged in.
- Confirm missing unpredictable CSRF token in request or form.
- Create a local HTML form that replicates the POST/GET action and autosubmit it while victim is authenticated.
- Observe the action completing

Recommendation: Implement perrequest CSRF tokens, SameSite cookies, and reauthentication for critical actions.

CSRF Scan_report :



```
HTTP/1.1 200 OK
Date: Tue, 16 Sep 2025 07:15:15 GMT
Server: Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12
X-Powered-By: PHP/8.2.12
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Content-Length: 4588
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

```
<form name="ping" action="#" method="post">
```

Alerts (22)
- Cross Site Scripting (Persistent) (2)
- Cross Site Scripting (Reflected)
- SQL Injection - MySQL
- SQL Injection - SQLite (Time Based)
- Absence of Anti-CSRF Tokens (3)
- Application Error Disclosure
- Content Security Policy (CSP) Header Not Set (13)
- Directory Browsing
- Missing Anti-clickjacking Header (13)
- Parameter Tampering (4)
- Cookie No HttpOnly Flag (3)
- Cookie without SameSite Attribute (3)
- Private IP Disclosure
- Server Leaks Information via "X-Powered-By" HTTP Response He
- Server Leaks Version Information via "Server" HTTP Response He
- X-Content-Type-Options Header Missing (17)
- Authentication Request Identified
- Information Disclosure - Sensitive Information in URL (4)
- Modern Web Application (11)
- Session Management Response Identified (2)
- User Agent Fuzzer (72)
- User Controllable HTML Element Attribute (Potential XSS) (10)

Absence of Anti-CSRF Tokens
URL: http://127.0.0.1/dvwa/vulnerabilities/exec/
Risk: Medium
Confidence: Low
Parameter:
Attack:
Evidence: <form name="ping" action="#" method="post">
CWE ID: 352
WASC ID: 9
Source: Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:

Description:
No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting

Other Info:
No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token, _csrfToken] was found in the following HTML form: [Form 1: "ip" "Submit"].

Solution:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

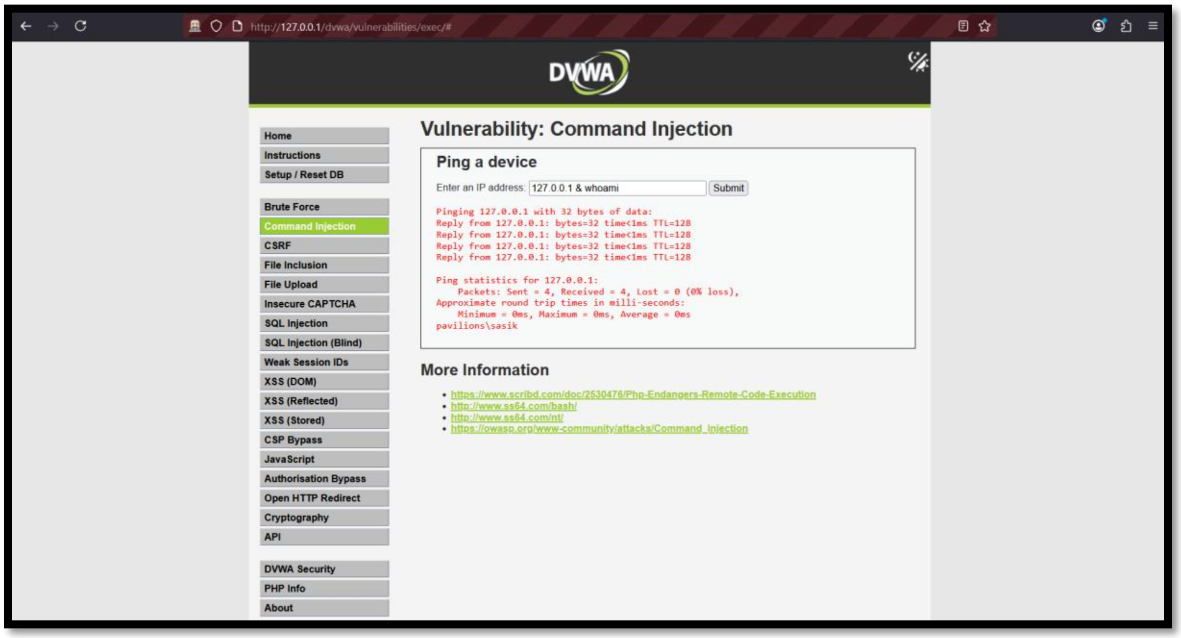
Reference:
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
https://cwe.mitre.org/data/definitions/352.html

Alerts 4 6 6 6 Main Proxy: localhost:8080
Current Status 0 0 0 0 0 0 0 0 0 0

Finding 5 — Command Injection (A03/A10)

Summary: Command injection on `/dvwa/vulnerabilities/exec/` that executes usersupplied input in OS commands.

Evidence: [cmd_injection_input_payload & result]



Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1 & whoami

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
pavilions\sasik
```

More Information

- https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://owasp.org/www-community/attacks/Command_Injection

Impact: High

Reproduction Steps:

- Open DVWA → Command Injection.
- Submit payload (Windows): `127.0.0.1 & whoami` or (Linux): `127.0.0.1; whoami`.
- Observe the OS command output in the response.

Recommendation:

Avoid executing shell commands with concatenated user input; validate and whitelist inputs; run under leastprivileged accounts.

OWASP Top 10 Checklist:

A01 Broken Access Control	
A02 Cryptographic Failures / Sensitive Data Exposure	✓
A03 Injection	✓
A04 Insecure Design/XXE	
A05 Security Misconfiguration	
A06 Vulnerable and Outdated Components	
A07 CrossSite Scripting (XSS)	✓
A08 CrossSite Request Forgery (CSRF)	✓
A09 Using Components with Known Vulnerabilities	
A10 Insufficient Logging & Monitoring	✓
Other	✓

Logs (Paste raw request/response here)

sqli_Raw Request Response:

sqli_Raw Request - Response.txt

Xss(r)_Raw Request Response:

Xss(r)_Raw Request - Response.txt

Xss(s)_Raw Request Response:

Xss(s)_Raw Request - Response.txt

csrf_Raw Request Response:

csrf_Raw Request - Response.txt

cmd_RAW Request Response:

cmd_RAW Request - Response.txt

ZAP SCAN REPORT :

Dwwa_Report.pdf

Safety & rules reminder :

- Only test **DVWA on your local XAMPP or Docker**. Do not run these payloads against any real/Internet sites.

<------------------------   END   ------------------------>