



# Vulnerability Assessment & Penetration Testing

**Report Title:** VAPT Report on DVWA

**Internship Batch:** Future Interns

**Report By:** Ashwanth P.G

# Table of Contents

1. Objective
2. Environment setup
3. Task Overview
4. Methodology & Tools
5. Findings & Evidence
6. Analysis, Impact & Remediation Suggestions
7. References

# Objective

Illustrate the successful setup on DVWA for Vulnerability Assessment and Penetration testing for demonstrate SQL injection, Cross-site scripting (xss) - Reflected, Command Injection, CSRF attacks and Brute Force attacks under “low” and “medium” security settings, along with PoC and remediation

## Environment Setup

Platform: DVWA installed on Linux (Kali or equivalent)

Access URL: <http://localhost/dvwa>

Credentials: admin / password (default from setup exercise)

Initial Security Level: Set to Low via DVWA Security panel

## Task Overview

According to Task 1:

- Log in to DVWA
- Switch DVWA security level between Low and Medium
- Explore Vulnerabilities
- Document of the findings

Methodology & Tools:

- I Browser (Firefox)
- I Burp Suite
- I DVWA

## Findings

Severity	Finding	Risk	CVSS v3 Score
Critical	SQL Injection in Login Form	Data breach	9.8
High	Insecure Direct Object References (IDOR)	Data exposure	8.7
High	Brute Force Attack	Password cracking	7.5
High	Cross-site request forgery (CSRF)	Social Engineering	7.5
High	Cross-site scripting (Xss reflected)	Impact on confidentiality	8.5

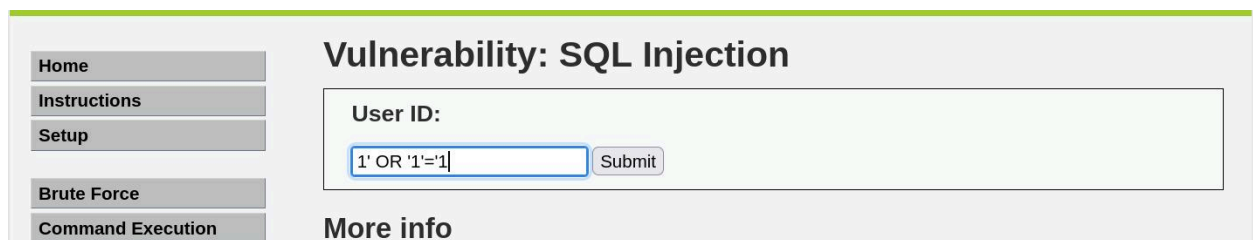
# Evidence

## 1. SQL Injection in Login Form

Location: <http://localhost/dvwa/sqli>

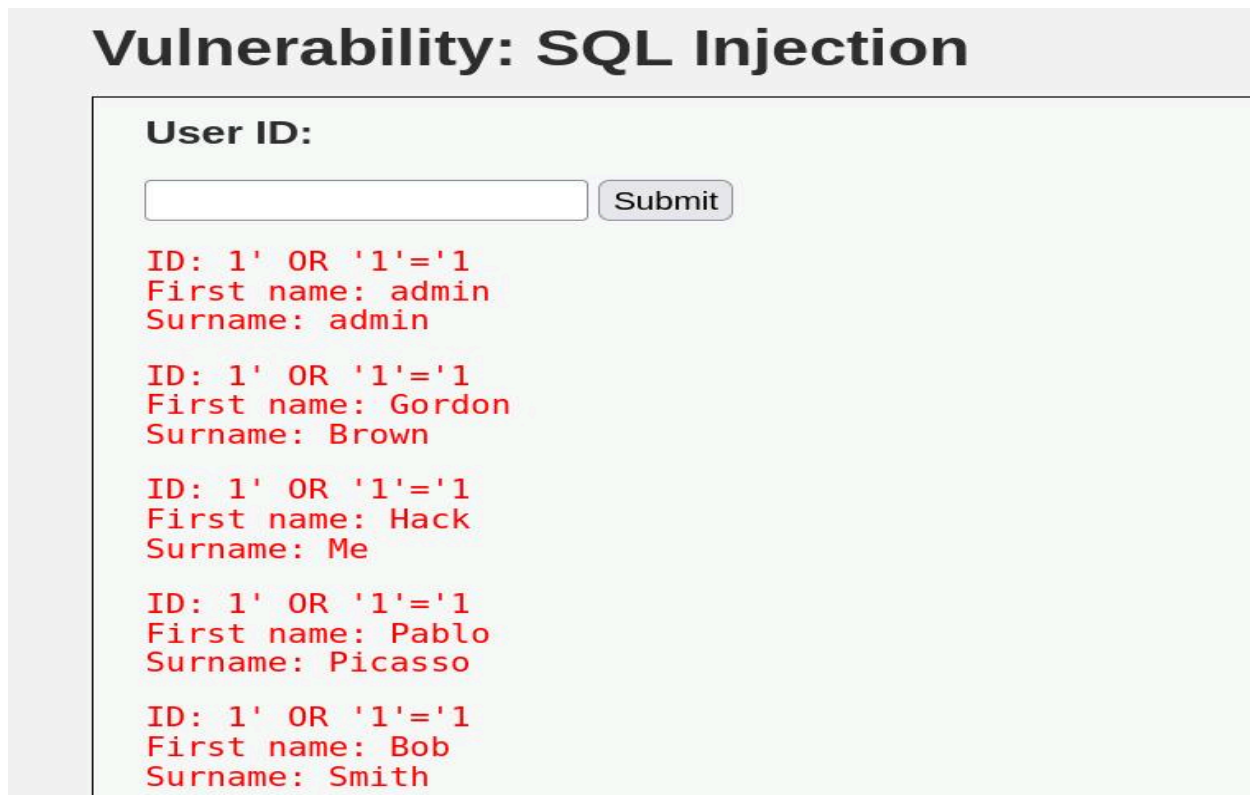
SQL Injection is a code injection technique that exploits improper input validation by inserting malicious SQL statements into an entry field, which are then executed by the backend database. This can lead to data leakage, authentication bypass, and in severe cases, full database compromise.

POC:



The screenshot shows the 'Vulnerability: SQL Injection' page. On the left is a sidebar with links: Home, Instructions, Setup, Brute Force, and Command Execution. The main area has a 'User ID:' label above a text input field containing the payload '1' OR '1'='1'. A 'Submit' button is to the right of the input field. Below the input field is a 'More info' link.

Injected the payload in the form field like **1' OR '1'='1** to check if the database have been triggered by this payload



The screenshot shows the 'Vulnerability: SQL Injection' page after submitting the payload. The 'User ID:' label is above an empty input field and a 'Submit' button. Below the input field, the results of the SQL injection are displayed in red text:

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

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

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

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

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

## 2. Insecure Direct Object Reference (IDOR)

Location: <http://localhost/dvwa/vulnerabilities/sqli/?id=1>

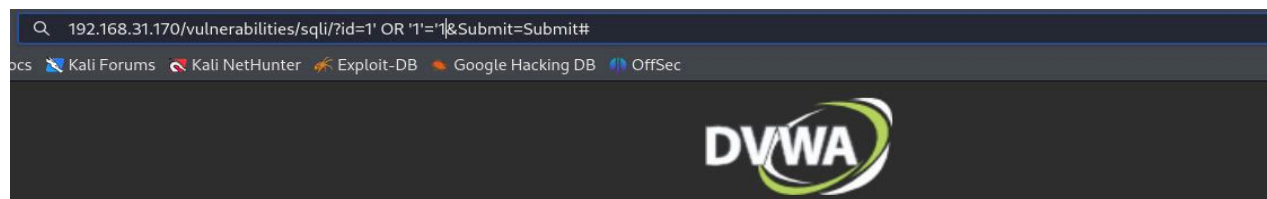
IDOR occurs when an application exposes references to internal implementation objects such as files, database records, or keys, and these can be manipulated to access unauthorized data without proper authorization or access control.

POC:

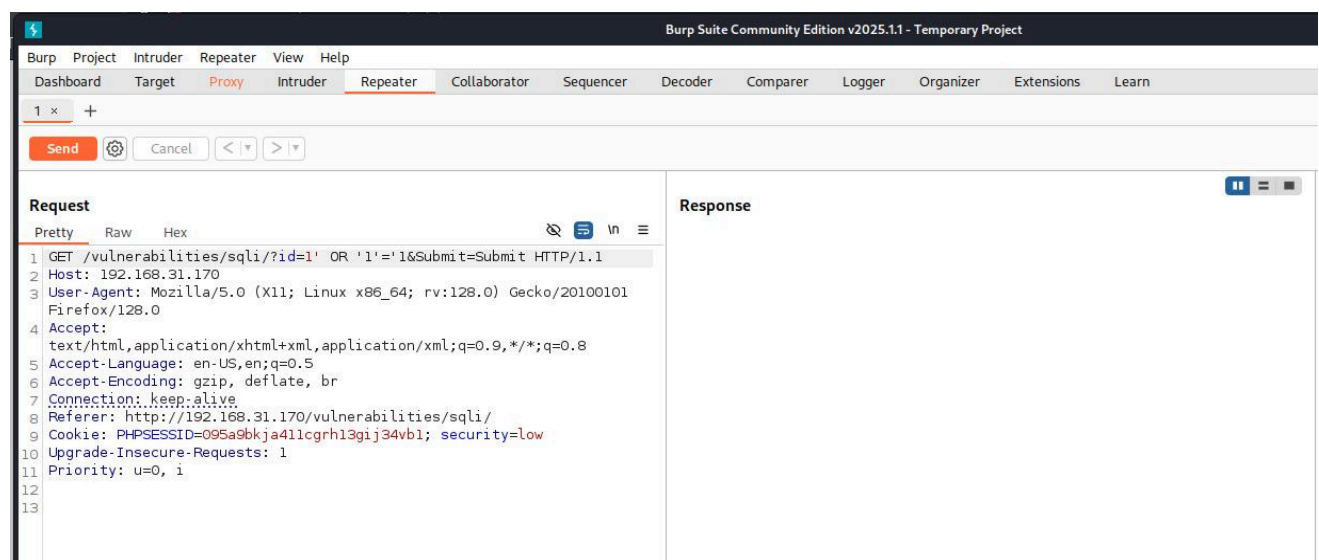
**Vulnerability: SQL Injection**

User ID:

Entered the user Id in the form field to check the website url changes to test the IDOR vulnerability, through injecting the payload command directly in the url creates a request in server



As we can see here the website's http request there is no https header so it is IDOR vulnerable



As we can see the request has been sent to the server and it renders the database of the user's first and sur name stored in the data base

The screenshot displays a web browser window with two main panes. The left pane, titled 'Request', shows an HTTP GET request to `/vulnerabilities/sqli/?id=1'OR'1'='1&Submit=Submit`. The right pane, titled 'Response', shows the rendered HTML output of the DVWA application. The response includes a navigation menu on the left with options like 'Home', 'Instructions', 'Setup', 'Brute Force', 'Command Execution', 'CSRF', 'File Inclusion', 'SQL Injection' (highlighted), 'SQL Injection (Blind)', 'Upload', 'XSS reflected', 'XSS stored', 'DVWA Security', and 'PHP Info'. The main content area is titled 'Vulnerability: SQL Injection' and contains a 'User ID:' form with a 'Submit' button. Below the form, the results of the SQL injection are displayed, showing the first and last names of users whose IDs are 1.

```
1 GET /vulnerabilities/sqli/?id=1'OR'1'='1&Submit=Submit HTTP/1.1
2 Host: 192.168.31.170
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://192.168.31.170/vulnerabilities/sqli/
9 Cookie: PHPSESSID=095a9bkja41lcgrh13gij34vbl; security=low
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
```

**Vulnerability: SQL Injection**

User ID:

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

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

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

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

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

### 3.Brute Force Attack

Location: <http://localhost/dvwa/vulnerabilities/login.php>

Brute Force attacks occur when an attacker attempts to gain unauthorized access by systematically guessing credentials (usernames/passwords) through repeated login attempts. Lack of rate limiting, CAPTCHA, or account lockout mechanisms enables this attack.

POC:

The screenshot shows the 'Vulnerability: Brute Force' page in the DVWA application. On the left is a navigation menu with options: 'Home', 'Instructions', 'Setup', 'Brute Force' (highlighted), 'Command Execution', and 'CSRF'. The main content area is titled 'Login' and contains a form with 'Username:' and 'Password:' labels, input fields, and a 'Login' button. The username field contains the text 'admin'.

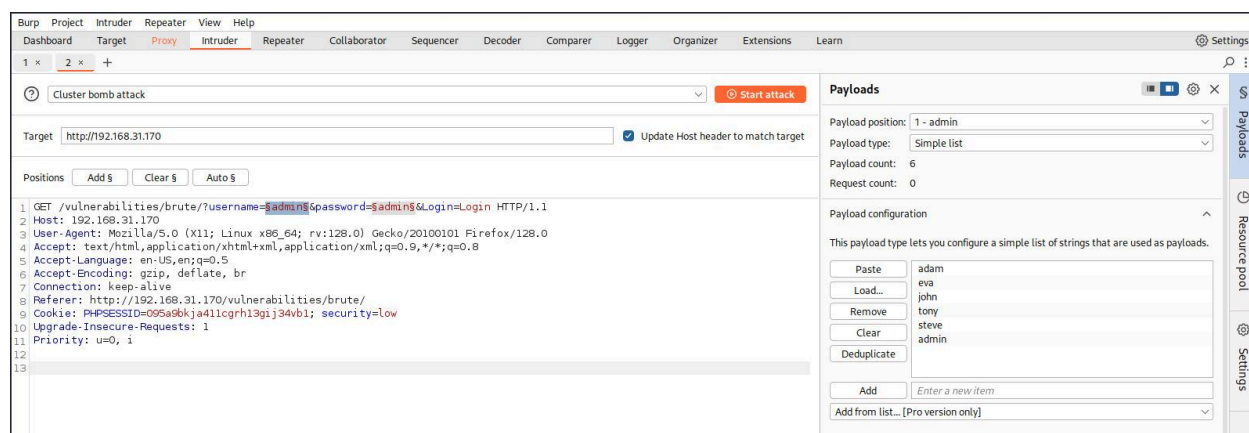
**Vulnerability: Brute Force**

**Login**

Username:

Password:

Intercepting the website request to view the structure of the website in the Burp Suite. After send the request to the intruder I have set the payloads for username and password with the technique of cluster bomb attack to brute force both simultaneously



Trying all the combinations of username and password to identify the valid credentials of the admin via brute forcing as we can see by sorting the result bellow the length of one login is larger than any other login attempt

2. Intruder attack of http://192.168.31.170								
Results Positions								
Capture filter: Capturing all items								
View filter: Showing all items								
Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
24	admin	password	200	15			5044	
0			200	8			5000	
3	john	tryhackme	200	6			5000	
6	admin	tryhackme	200	12			5000	
8	eva	jesus123	200	7			5000	
10	tony	jesus123	200	9			5000	
12	admin	jesus123	200	6			5000	
14	eva	password2031	200	8			5000	
16	tony	password2031	200	8			5000	
18	admin	password2031	200	11			5000	
20	eva	password	200	9			5000	
21	john	password	200	11			5000	
22	tony	password	200	9			5000	
23	steve	password	200	9			5000	
25	admin	admin	200	10			5000	
26	eva	admin	200	12			5000	
27	john	admin	200	10			5000	
28	tony	admin	200	11			5000	
29	steve	admin	200	7			5000	
30	admin	admin	200	7			5000	
31	adam	bruteforce	200	6			5000	
32	eva	bruteforce	200	18			5000	
33	john	bruteforce	200	6			5000	
34	tony	bruteforce	200	25			5000	
35	steve	bruteforce	200	9			5000	
36	admin	bruteforce	200	5			5000	
1	adam	tryhackme	200	6			4999	
2	eva	tryhackme	200	5			4999	
4	tony	tryhackme	200	5			4999	
5	steve	tryhackme	200	4			4999	
7	adam	jesus123	200	6			4999	
9	john	jesus123	200	8			4999	



As we can render the result bellow shows that we have gained the admin privilege access control for that website

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
24	admin	password	200	15			5044	
0			200	8			5000	
3	john	tryhackme	200	6			5000	
6	admin	tryhackme	200	12			5000	
8	eva	jesus123	200	7			5000	
10	tony	jesus123	200	9			5000	
12	admin	jesus123	200	6			5000	
14	eva	password2031	200	8			5000	
16	tony	password2031	200	8			5000	

Request Response

Pretty Raw Hex Render

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The left sidebar has a menu with options: Home, Instructions, Setup, Brute Force (highlighted), Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The main content area is titled 'Vulnerability: Brute Force' and contains a 'Login' form with fields for 'Username:' and 'Password:', a 'Login' button, and a message: 'Welcome to the password protected area admin'. Below the form, there is a 'More info' section with three links related to OWASP and security focus.

## 4. Cross-site Request Forgery

**Location:** Entire application

CSRF is an attack that forces a user to execute unwanted actions on a web application in which they are authenticated. By exploiting the trust a site has in the user's browser, attackers can perform state-changing actions (like changing passwords, updating profiles, or initiating financial transactions) without the user's consent.

**POC:**

The screenshot shows the DVWA interface for the 'Vulnerability: Cross Site Request Forgery (CSRF)' section. The main content area is titled 'Change your admin password:' and contains a form with two input fields: 'New password:' and 'Confirm new password:'. Both fields have masked characters (dots). There is a 'Change' button at the bottom of the form. The left sidebar menu is visible on the left side of the page.

To change the password in any account without requesting to forget the password. View the website source page and look for the form contains the password changing features, There is a form field presented in the web site source page that contains the option to change the new password

```

<div class="body_padded">
  <h1>Vulnerability: Cross Site Request Forgery (CSRF)</h1>

  <div class="vulnerable_code_area">

    <h3>Change your admin password:</h3>
    <br>
    <form action="#" method="GET">      New password:<br>
    <input type="password" AUTOCOMPLETE="off" name="password_new"><br>
    Confirm new password:<br>
    <input type="password" AUTOCOMPLETE="off" name="password_conf">
    <br>
    <input type="submit" value="Change" name="Change">
    </form>

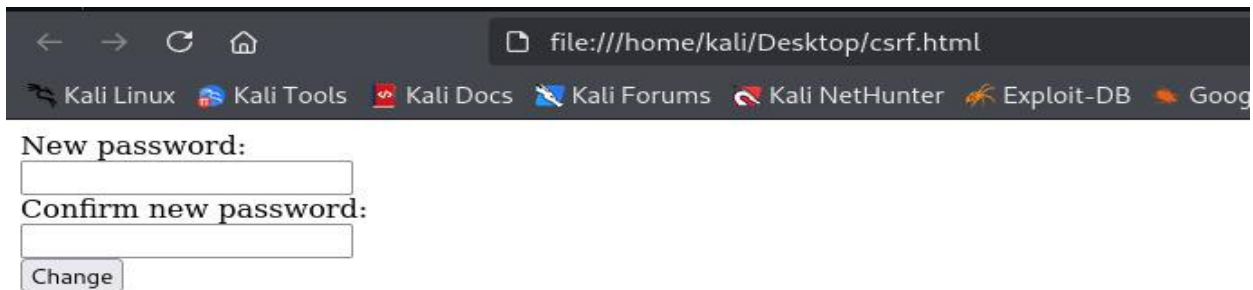
    <pre> Password Changed </pre>

  </div>

```

copy the form field and create a separate file as csrf.html and paste the form in that file save and open as html file

As we can see below the html file shows a separate option for changing the password of the user website login



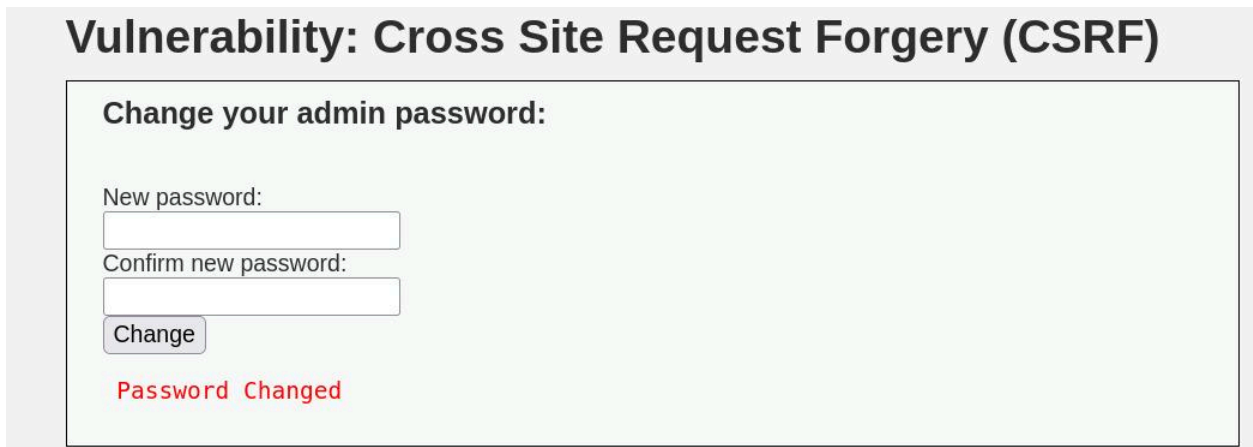
file:///home/kali/Desktop/csrf.html

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google

New password:

Confirm new password:

Change



## Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Change

Password Changed

As we can see the result that without requesting the forget password option we can easily change the password of the user account

# Analysis, Impact & Remediation

## 1. SQLi Injection:

### Analysis:

- **Expected Behavior:** The login form should validate credentials against the database and reject incorrect inputs
- **Observed Behavior:** The application accepted the input `1'OR '1'='1`, bypassed the password check, and logged the attacker in as the `admin` user.

### Impact:

An attacker can bypass authentication, exfiltrate database contents, modify data, or potentially achieve Remote Code Execution (RCE) depending on the database and configuration.

### Recommendation:

- Use **parameterized queries** or **prepared statements**.
- Employ **input validation** and **output encoding**.
- Implement a **Web Application Firewall (WAF)** to detect and block SQLi attempts.
- Conduct regular code reviews and security testing.

## 2. Insecure Direct Object Reference (IDOR):

### Analysis:

- **Expected Behavior:** User A should not be able to access User B's profile.
- **Observed Behavior:** The server returned sensitive data for User B without validating ownership or access rights.

### Impact:

- Unauthorized access to personal data of other users (PII exposure)
- Potential for account takeover or social engineeringViolation of data protection regulations (e.g., GDPR)

### Recommendation:

- Enforce **object-level authorization** on all endpoints.
- Never trust client-side identifiers for access control decisions.
- Use **session-based ownership checks** on the server-side.
- Regularly test API endpoints for broken access control issues.

### 3. Brute Force Attack

#### Analysis:

Attempt	Payload	Status Code	Response size	Note
1	admin123	200	500	Invalid
2	password	302	512	Login

#### Impact:

- Unauthorized access to user/admin accounts.
- Potential full control over the application if privileged accounts are compromised.
- Risk of data theft, defacement, or further privilege escalation.

#### Recommendation:

- Implement **account lockout** after a defined number of failed attempts.
- Add **rate-limiting** and **CAPTCHA** on login endpoints.
- Use **multi-factor authentication (MFA)** for all users, especially admins.
- Monitor and alert on failed login attempts from a single IP.

### 4. Cross-Site Request Forgery (CSRF):

#### Analysis:

- No CSRF token was required.
- No re-authentication or origin header validation was enforced.
- The server processed the malicious request successfully and updated the user's email.

#### Impact:

- Unauthorized state changes performed without user interaction.
- Account takeover risks (e.g., email or password changes).
- Violation of user trust and potential legal/regulatory consequences (e.g., GDPR).

#### Recommendation:

- Implement **anti-CSRF tokens** (e.g., synchronizer tokens or double submit cookies).
- Validate **Origin** and **Referer** headers for state-changing requests.
- Use **SameSite cookies** set to Strict or Lax.
- Prompt re-authentication for sensitive actions (like email/password change).

# Conclusion

The penetration test conducted on DVWA (Damn Vulnerable Web Application) successfully identified several critical and high-risk vulnerabilities that can be exploited by attackers to compromise the confidentiality, integrity, and availability of the application and its data. The following key vulnerabilities were discovered:

1. **SQL Injection (SQLi)** – Enabled unauthorized database access and authentication bypass through unsanitized input.
2. **Insecure Direct Object Reference (IDOR)** – Allowed unauthorized access to other users' sensitive data by manipulating object identifiers in the request.
3. **Brute Force Attack** – Revealed weak authentication controls due to the absence of rate-limiting, account lockouts, and CAPTCHA mechanisms.
4. **Cross-Site Request Forgery (CSRF)** – Permitted unauthorized state changes (e.g., email updates) via forged requests due to missing CSRF protection tokens.

These vulnerabilities highlight a lack of secure coding practices and insufficient implementation of access control and session management mechanisms.

While DVWA is intentionally vulnerable for educational purposes, this exercise emphasizes the importance of secure development practices, proper input validation, and comprehensive access control mechanisms in real-world applications.

## Recommendations Summary

To mitigate the identified vulnerabilities, it is strongly recommended to:

- Sanitize and parameterize all user inputs.
- Implement access control checks at both the object and functional levels.
- Introduce account lockout and rate-limiting mechanisms.
- Use CSRF tokens and SameSite cookie attributes for all state-changing operations.

## Final Note

This penetration test was conducted in a controlled lab environment for learning and awareness purposes. In a production environment, such vulnerabilities can lead to serious breaches. Organizations are encouraged to follow secure coding standards, conduct regular security testing, and promote a strong security culture among development and operations teams.