BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

Name	Sujal Sandeep Dingankar
UID no.	2024301005
Date	15-10-2025
Lab	7
Github Link	https://github.com/SUJALGPM/DVWA-WebSecurity

Executive Summary :-

This report documents the successful identification, exploitation, and mitigation of common web application vulnerabilities using the Damn Vulnerable Web Application (DVWA). Key vulnerabilities including SQL Injection, Cross-Site Scripting (Reflected and Stored), Command Injection, File Upload, CSRF, and others were successfully exploited under a low-security setting to understand their root causes and impact. Subsequently, the application's high-security configurations were enabled to demonstrate the effectiveness of modern mitigation techniques such as prepared statements, output encoding, and the use of anti-CSRF tokens. The experiment highlights the critical importance of secure coding practices, validating all user input, and implementing a defense-in-depth security posture for any web application.

Setup Notes :-

The experiment was conducted in a controlled virtual environment to ensure safety and prevent any impact on the host system or network.

- Virtualization Software: Oracle VM VirtualBox
- Operating System: Ubuntu Server 22.04.5 LTS
- Web Stack: LAMP (Linux, Apache2, MariaDB, PHP)
- Application: DVWA (Damn Vulnerable Web Application) cloned from the official GitHub repository.
- Network Configuration: The VM was configured with a Bridged Network Adapter to be accessible from the host machine's browser. The IP address was masked for this report (e.g., 192.168.1.XXX).

Key Setup Commands :

1. Use a VM (VirtualBox / VMware / a cloud VM). DO NOT run this on your host machine or a public server — DVWA is intentionally vulnerable.

2.This guide assumes an Ubuntu/Debian VM with internet access and you have a user with sudo privileges.

1.Update system & install required packages (LAMP components + extras)

Commands:-
sudo apt update
sudo apt upgrade -y

Install Apache, MariaDB, PHP and required PHP extensions, git and unzip
sudo apt install -y apache2 mariadb-server php php-mysqli php-xml php-gd php-mbstring git unzip curl

```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo apt install -y apache2 mariadb-server php php-mysqli php-xml php-gd php-
mbstring git unzip curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'php8.1-mysql' instead of 'php-mysqli'
apache2 is already the newest version (2.4.52-1ubuntu4.16).
unzip is already the newest version (6.0-26ubuntu3.2).
The following packages were automatically installed and are no longer required:
  libaio1 libevent-core-2.1-7 libevent-pthreads-2.1-7 libmecab2 libprotobuf-lite23 linux-headers-6.8.0-79-generic
  linux-hwe-6.8-headers-6.8.0-79 linux-hwe-6.8-tools-6.8.0-79 linux-image-6.8.0-79-generic linux-modules-6.8.0-79-generic
  linux-modules-extra-6.8.0-79-generic linux-tools-6.8.0-79-generic mecab-ipadic mecab-ipadic-utf8 mecab-utils
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  git-man libapache2-mod-php8.1 liberror-perl libonig5 php-common php8.1 php8.1-cli php8.1-common php8.1-gd php8.1-mbstring php8.1-opcache
  php8.1-readline php8.1-xml
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn php-pear
The following NEW packages will be installed:
  curl git git-man libapache2-mod-php8.1 liberror-perl libonig5 mariadb-server php php-common php-gd php-mbstring php-xml php8.1 php8.1-
cli
  php8.1-common php8.1-gd php8.1-mbstring php8.1-mysql php8.1-opcache php8.1-readline php8.1-xml
0 upgraded, 21 newly installed, 0 to remove and 0 not upgraded.
Need to get 194 kB/10.4 MB of archives.
After this operation, 45.7 MB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.21 [194 kB]
Fetched 194 kB in 2s (117 kB/s)
Selecting previously unselected package curl.
(Reading database ... 251063 files and directories currently installed.)
Preparing to unpack .../00-curl_7.81.0-1ubuntu1.21_amd64.deb ...
Unpacking curl (7.81.0-1ubuntu1.21) ...
Selecting previously unselected package liberror-perl.
```

```
Creating config file /etc/php/8.1/mods-available/readline.ini with new version
Setting up curl (7.81.0-1ubuntu1.21) ...
Setting up php8.1-opcache (8.1.2-1ubuntu2.22) ...

Creating config file /etc/php/8.1/mods-available/opcache.ini with new version
Setting up libonig5:amd64 (6.9.7.1-2build1) ...
Setting up php-xml (2:8.1+92ubuntu1) ...
Setting up php8.1-mbstring (8.1.2-1ubuntu2.22) ...

Creating config file /etc/php/8.1/mods-available/mbstring.ini with new version
Setting up php-mbstring (2:8.1+92ubuntu1) ...
Setting up php8.1-cli (8.1.2-1ubuntu2.22) ...
update-alternatives: using /usr/bin/php8.1 to provide /usr/bin/php (php) in auto mode
update-alternatives: using /usr/bin/phar8.1 to provide /usr/bin/phar (phar) in auto mode
update-alternatives: using /usr/bin/phar.phar8.1 to provide /usr/bin/phar.phar (phar.phar) in auto mode

Creating config file /etc/php/8.1/cli/php.ini with new version
Setting up git (1:2.34.1-1ubuntu1.15) ...
Setting up libapache2-mod-php8.1 (8.1.2-1ubuntu2.22) ...

Creating config file /etc/php/8.1/apache2/php.ini with new version
Module mpm_event disabled.
Enabling module mpm_prefork.
apache2_switch_mpm Switch to prefork
apache2_invoke: Enable module php8.1
Setting up php8.1 (8.1.2-1ubuntu2.22) ...
Setting up php (2:8.1+92ubuntu1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.11) ...
Processing triggers for php8.1-cli (8.1.2-1ubuntu2.22) ...
Processing triggers for libapache2-mod-php8.1 (8.1.2-1ubuntu2.22) ...
```

2. Enable & start services

Commands:-
sudo systemctl enable --now apache2
sudo systemctl enable --now mariadb
sudo systemctl status apache2 --no-pager
sudo systemctl status mariadb --no-pager

3. Secure MariaDB

Commands:-
sudo mysql_secure_installation

```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo systemctl status mariadb --no-pager
● mariadb.service - MariaDB 10.6.22 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-10-01 10:36:21 IST; 13min ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
   Main PID: 8725 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 8 (limit: 124729)
     Memory: 66.9M
        CPU: 507ms
     CGroup: /system.slice/mariadb.service
             └─8725 /usr/sbin/mariadbd

Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC mariadbd[8725]: 2025-10-01 10:36:21 0 [Note] /usr/sbin/mariadbd: ready for connections
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC mariadbd[8725]: Version: '10.6.22-MariaDB-0ubuntu0.22.04.1'  socket: '/run/mysqld/m…u
22.04
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC systemd[1]: Started MariaDB 10.6.22 database server.
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8741]: Upgrading MySQL tables if necessary.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: Looking for 'mariadb' as: /usr/bin/mariadb
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: Looking for 'mariadb-check' as: /usr/bin/mariadb-check
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: This installation of MariaDB is already upgraded to 1…
riaDB.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: There is no need to run mysql_upgrade again.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: You can use --force if you still want to run
mysql_upgrade
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8754]: Checking for insecure root accounts.
Hint: Some lines were ellipsized, use -l to show in full.
```


```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo systemctl status mariadb --no-pager
● mariadb.service - MariaDB 10.6.22 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-10-01 10:36:21 IST; 13min ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
   Main PID: 8725 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 8 (limit: 124729)
     Memory: 66.9M
        CPU: 507ms
     CGroup: /system.slice/mariadb.service
             └─8725 /usr/sbin/mariadbd

Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC mariadbd[8725]: 2025-10-01 10:36:21 0 [Note] /usr/sbin/mariadbd: ready for connections
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC mariadbd[8725]: Version: '10.6.22-MariaDB-0ubuntu0.22.04.1'  socket: '/run/mysqld/m…u
22.04
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC systemd[1]: Started MariaDB 10.6.22 database server.
Oct 01 10:36:21 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8741]: Upgrading MySQL tables if necessary.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: Looking for 'mariadb' as: /usr/bin/mariadb
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: Looking for 'mariadb-check' as: /usr/bin/mariadb-check
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: This installation of MariaDB is already upgraded to 1…
riaDB.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: There is no need to run mysql_upgrade again.
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8744]: You can use --force if you still want to run
mysql_upgrade
Oct 01 10:36:22 students-HP-280-G3-SFF-Business-PC /etc/mysql/debian-start[8754]: Checking for insecure root accounts.
Hint: Some lines were ellipsized, use -l to show in full.
```

```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] y
Enabled successfully!
Reloading privilege tables..
 ... Success!


You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
 ... Success!
```

4. Download DVWA into web root

Commands:-
cd /tmp
git clone https://github.com/digininja/DVWA.git
sudo mv DVWA /var/www/html/dvwa
ls -la /var/www/html/dvwa

5. Set ownership & permissions

Commands:-
sudo chown -R www-data:www-data /var/www/html/dvwa
sudo chmod -R 755 /var/www/html/dvwa
sudo cp /var/www/html/dvwa/config/config.inc.php.dist var/www/html/dvwa/config/config.inc.php

```
Cloning into 'DVWA'...
remote: Enumerating objects: 5588, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 5588 (delta 47), reused 32 (delta 32), pack-reused 5529 (from 3)
Receiving objects: 100% (5588/5588), 2.65 MiB | 25.17 MiB/s, done.
Resolving deltas: 100% (2772/2772), done.
```

6.Create DVWA database & user (MariaDB)

Commands:-

Method A — using sudo mysql (works if root uses socket auth):

sudo mysql -e "CREATE DATABASE IF NOT EXISTS dvwa;"
sudo mysql -e "CREATE USER IF NOT EXISTS 'dvwauser'@'localhost' IDENTIFIED BY
sudo mysql -e "GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost';"
sudo mysql -e "FLUSH PRIVILEGES;"

Method B — using mysql -u root -p (if you set root password):

mysql -u root -p
then at mysql> prompt:
CREATE DATABASE dvwa;
CREATE USER 'dvwauser'@'localhost' IDENTIFIED BY 'dvwapass';
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost';
FLUSH PRIVILEGES;
EXIT;

```
Disallow root login remotely? [Y/n] y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo mysql <<SQL
CREATE DATABASE IF NOT EXISTS dvwa;
CREATE USER IF NOT EXISTS 'dvwauser'@'localhost' IDENTIFIED BY 'dvwapass';
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost';
FLUSH PRIVILEGES;
EXIT;
SQL
-------------
EXIT
-------------
```

```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo mysql <<'SQL'
CREATE DATABASE IF NOT EXISTS dvwa;
CREATE USER IF NOT EXISTS 'dvwauser'@'localhost' IDENTIFIED BY 'dvwapass';
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwauser'@'localhost';
FLUSH PRIVILEGES;
SQL
```

```
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo mysql -e "SHOW DATABASES LIKE 'dvwa';"
+----------------+
| Database (dvwa) |
+----------------+
| dvwa           |
+----------------+
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ sudo mysql -e "SELECT user,host FROM mysql.user WHERE user='dvwauser';"
+----------+-----------+
| User     | Host      |
+----------+-----------+
| dvwauser | localhost |
+----------+-----------+
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ mysql -u dvwauser -p -D dvwa
# when prompted, enter: dvwapass
Enter password:
ERROR 1045 (28000): Access denied for user 'dvwauser'@'localhost' (using password: YES)
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ # when prompted, enter: dvwapass
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ mysql -u dvwauser -p -D dvwa
Enter password:
ERROR 1045 (28000): Access denied for user 'dvwauser'@'localhost' (using password: YES)
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$ mysql -u dvwauser -p -D dvwa
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 47
Server version: 10.6.22-MariaDB-0ubuntu0.22.04.1 Ubuntu 22.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [dvwa]> exit
Bye
students@students-HP-280-G3-SFF-Business-PC:~/2024301005$
```

7. Edit DVWA config to match DB credentials

Commands:-

sudo nano /var/www/html/dvwa/config/config.inc.php

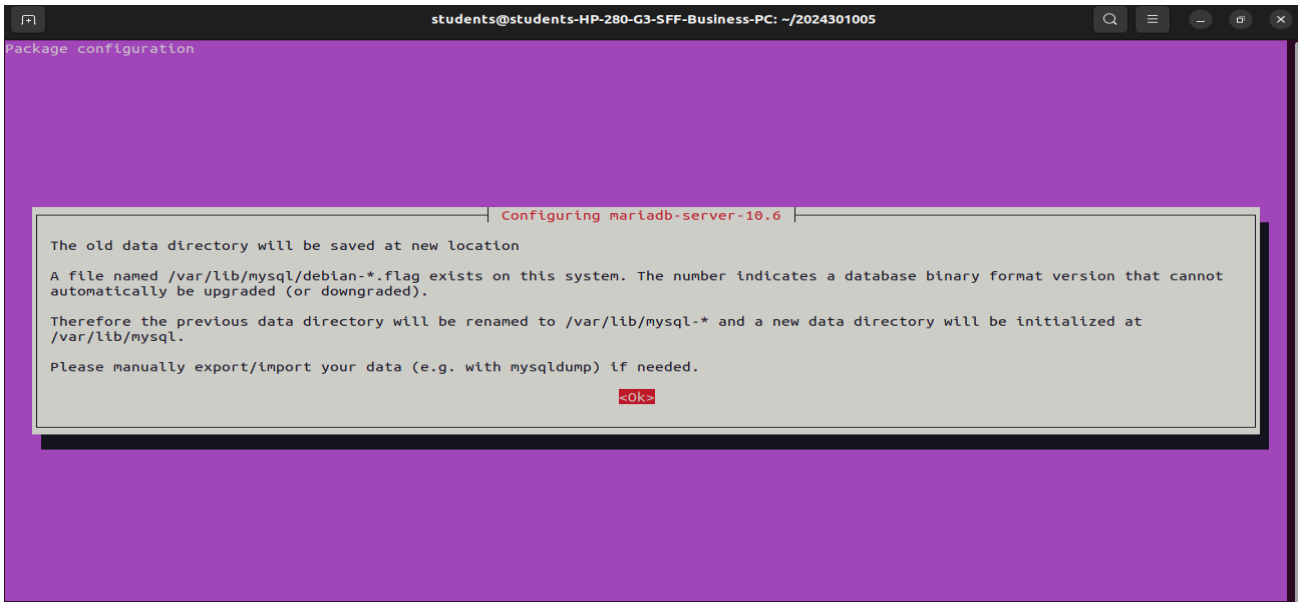
Find and set these values (exact lines may vary slightly):

$_DVWA['db_server'] = 'localhost';

$_DVWA['db_database'] = 'dvwa';

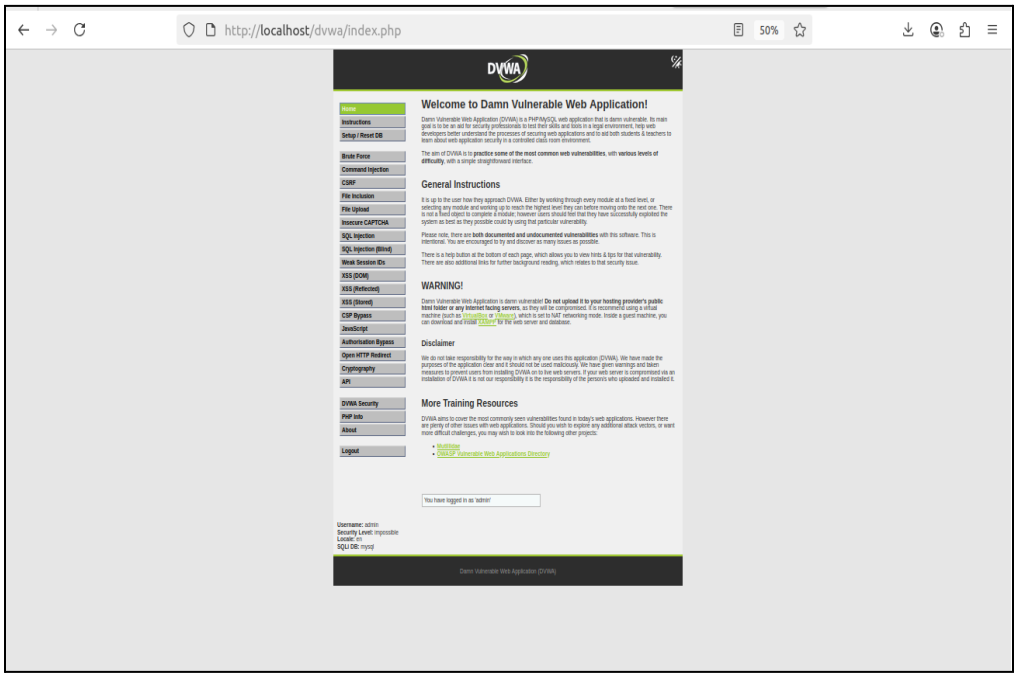
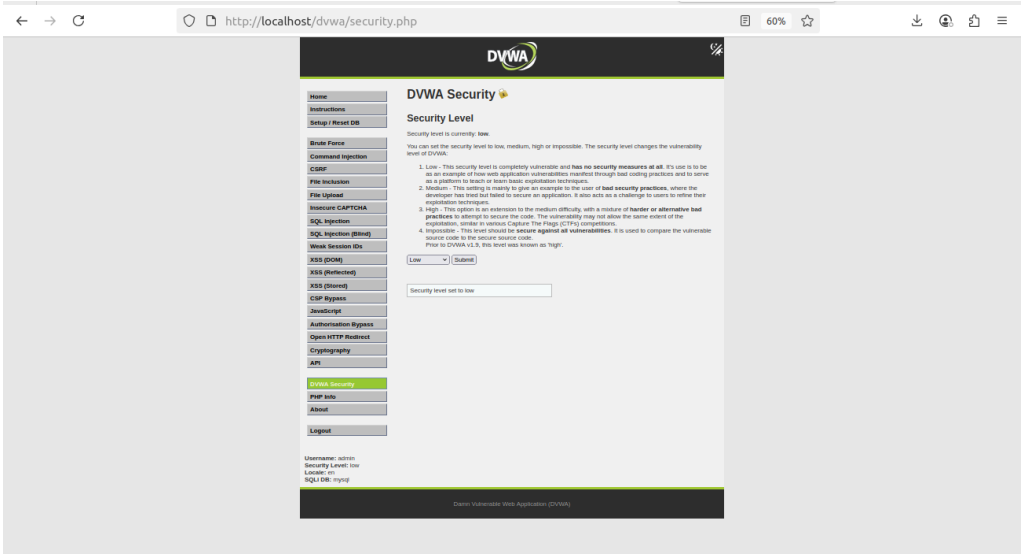
$_DVWA['db_user'] = 'dvwauser';

$_DVWA['db_password'] = 'dvwapass';



Part A: Setup & Baseline :-

This section confirms the successful deployment and initial configuration of DVWA.

Item	Item Evidence
DVWA Setup Page	
DVWA Login Page	
DVWA Security Level	

Security Level Explanation

The DVWA security setting changes how the application code handles user input to simulate different levels of protection:

● Low: Implements no security measures, making it trivial to demonstrate basic exploits.

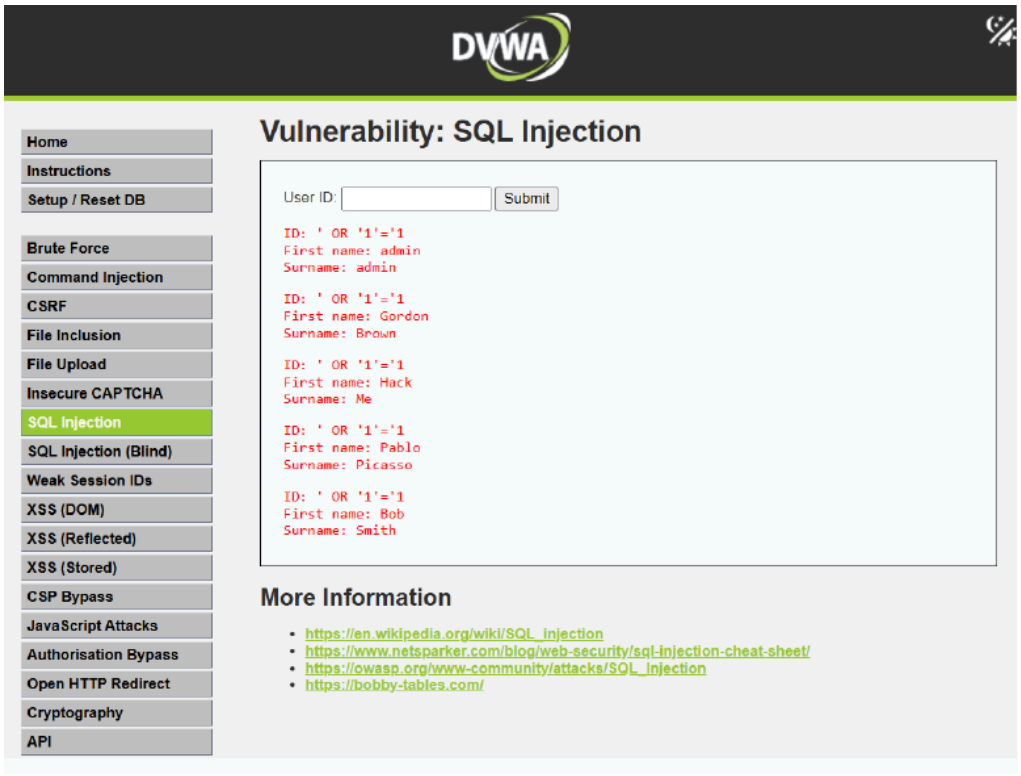
● Medium: Introduces basic, often flawed, security filters (e.g., blacklisting keywords) to teach bypass techniques.

● High: Implements stronger, modern defenses (e.g., prepared statements, CSRF tokens) that are much harder to exploit.

Part B & C: Basic Vulnerability Exploitation (Low Security)

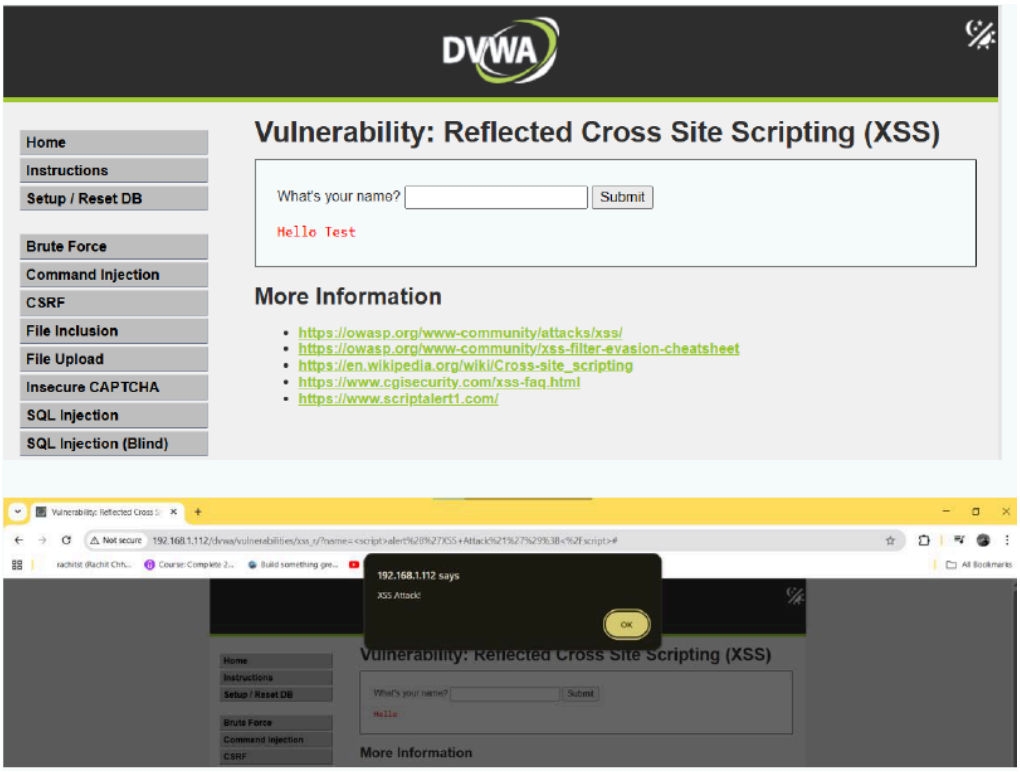
The following vulnerabilities were identified and exploited with the security level set to Low.

SQL Injection (SQLi)

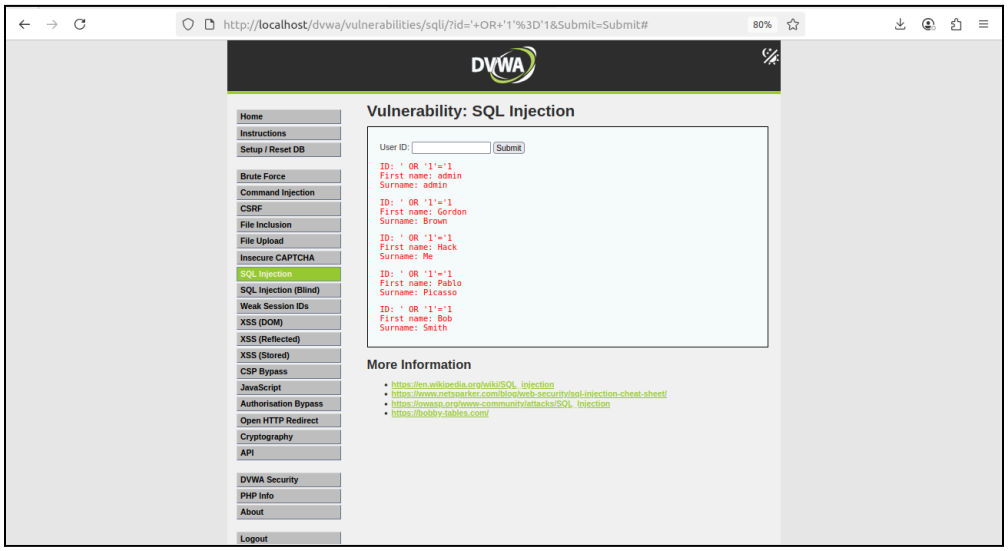
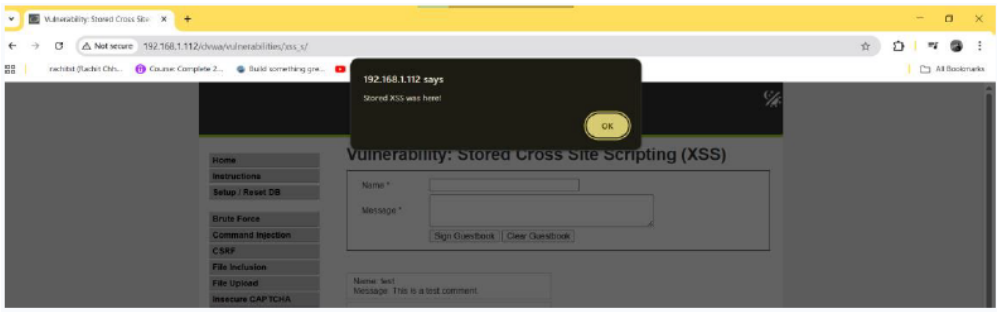
Aspect	Details
Risk Rating	High
Exploitation Steps	1. Navigate to the "SQL Injection" page. 2. In the "User ID" input box, enter the payload: ' OR '1'='1 3. Click "Submit". The application will dump the user details for all users in the database.
Evidence	
Root Cause Analysis	The application directly concatenates the user's input into the SQL query without sanitization.

	This allows the input to be interpreted as part of the SQL command, altering the query's logic to bypass the WHERE clause and return all records.
Proposed Fix	Implement Parameterized Queries (Prepared Statements). This practice separates the SQL code from the user-supplied data, ensuring the input is always treated as data and never as an executable command.

Reflected Cross-Site Scripting (XSS)

Aspect	Details
Risk Rating	Medium
Exploitation Steps	1. Navigate to the "XSS (Reflected)" page. 2. In the name input box, enter the payload: <script>alert('XSS Attack!');</script> 3. Click "Submit". The browser executes the script, displaying a pop-up alert box
Evidence	
Root Cause Analysis	The application takes user input and reflects it directly back onto the webpage without proper output encoding. The browser misinterprets the malicious script as legitimate code and executes it.
Proposed Fix	Implement context-aware Output Encoding. Before rendering user input in HTML, convert special characters (e.g., < > " ') into their corresponding HTML entities (e.g., < > " ').

Stored Cross-Site Scripting (XSS)

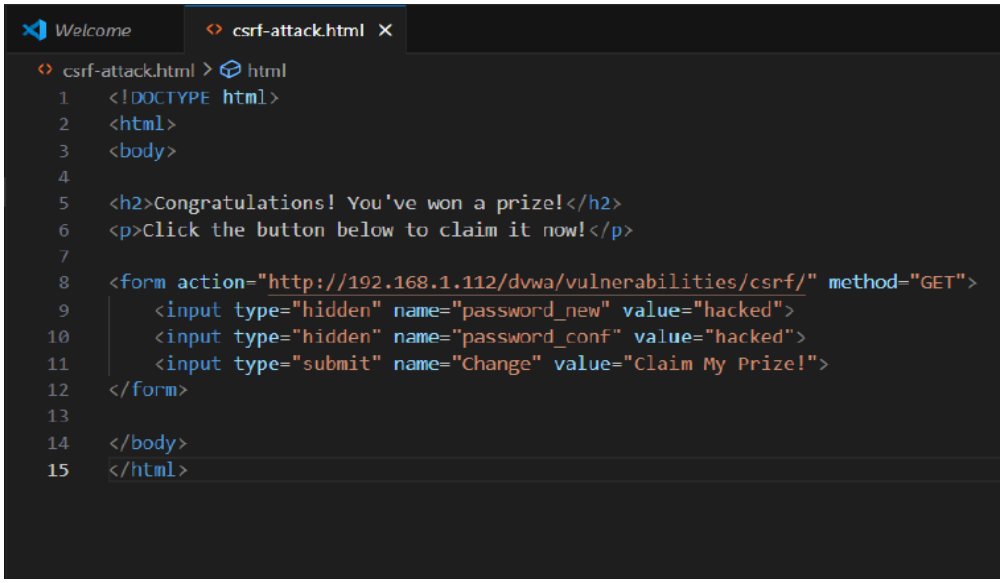
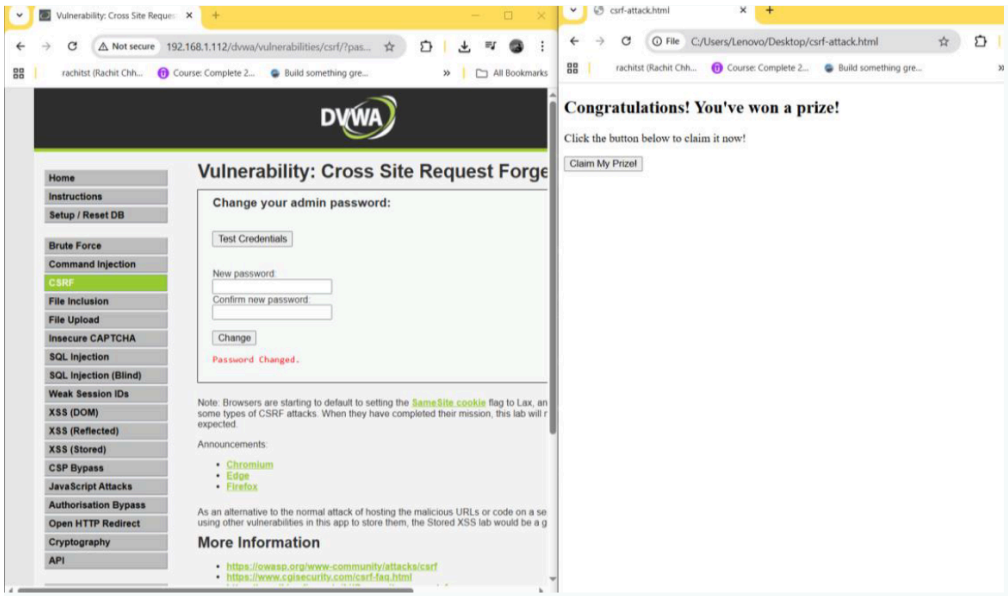
Aspect	Details
Risk Rating	High
Exploitation Steps	1. Navigate to the "XSS (Stored)" page. 2. In the "Message" input box, enter the payload: <script>alert('Stored XSS was here!');</script> 3. Click "Sign Guestbook". The malicious script is saved to the database. 4. The page reloads, and the script executes for the current user and for any future visitor to the page.
Evidence	 
Root Cause Analysis	The application stores unsanitized user input in the database. When this stored data is retrieved and displayed to other users, it is rendered without output encoding, causing the malicious script to execute in their browsers.
Proposed Fix	A combination of Input Validation (to strip dangerous tags before storing) and strict Output Encoding (when displaying the data) is required for a robust defense.

Brute Force :

Aspect	Details
Risk Rating	Medium
Exploitation Steps	1. Log out of DVWA to access the login page. 2. Enter the username admin and a series of incorrect passwords (e.g., 123, password123, test). 3. Observe that the application allows unlimited, rapid login attempts without any penalty, delay, or lockout. This behavior is vulnerable to automated attacks.
Evidence	
Root Cause Analysis	The login mechanism lacks essential security controls like rate-limiting or account lockout. It does not track failed login attempts, allowing an attacker to make an infinite number of password guesses.
Proposed Fix	Implement Account Lockout policies (e.g., lock account for 15 minutes after 5 failed attempts), introduce Progressive Delays between failed attempts, and use a CAPTCHA to deter automated bots.

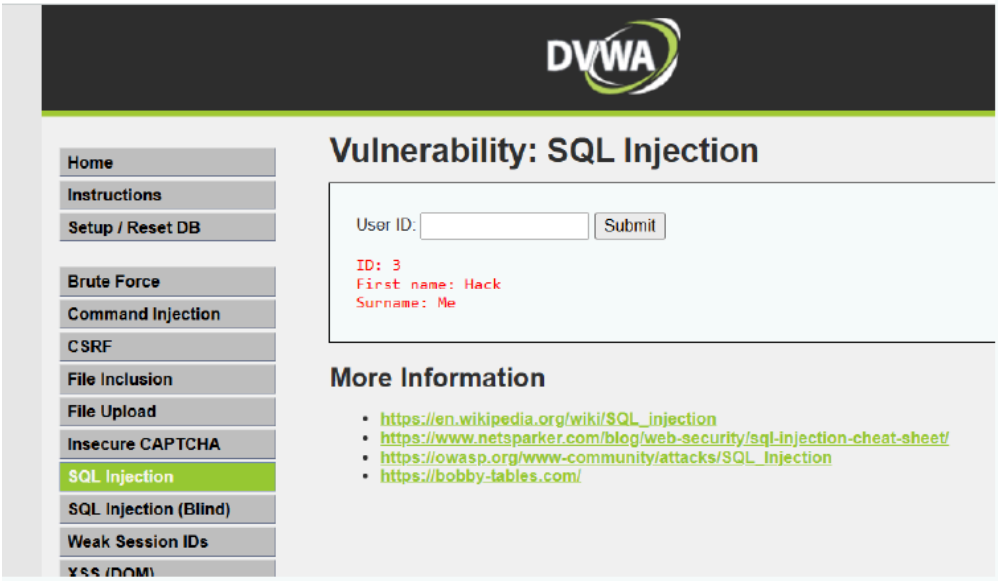
Cross-Site Request Forgery (CSRF)

Aspect	Details
Risk Rating	Medium
Exploitation Steps	1. Create a malicious HTML page (csrf-attack.html) containing a hidden form that targets the DVWA password change function. 2. With a valid session in DVWA, open the malicious page in another browser tab. 3. The victim clicks the "Claim My Prize!" button, which unknowingly submits the password change request to DVWA. 4. The password for the admin account is successfully changed to "hacked".

Evidence	 
Root Cause Analysis	The application fails to verify the origin and intent of the request. It processes the state-changing action (password change) based solely on the user's active session cookie, without requiring a unique, secret token to confirm the request came from the legitimate application form.
Proposed Fix	Implement Anti-CSRF Tokens. A unique, unpredictable token should be embedded in every state-changing form. The server must validate this token upon submission to ensure the request is legitimate

Insecure Direct Object References (IDOR)

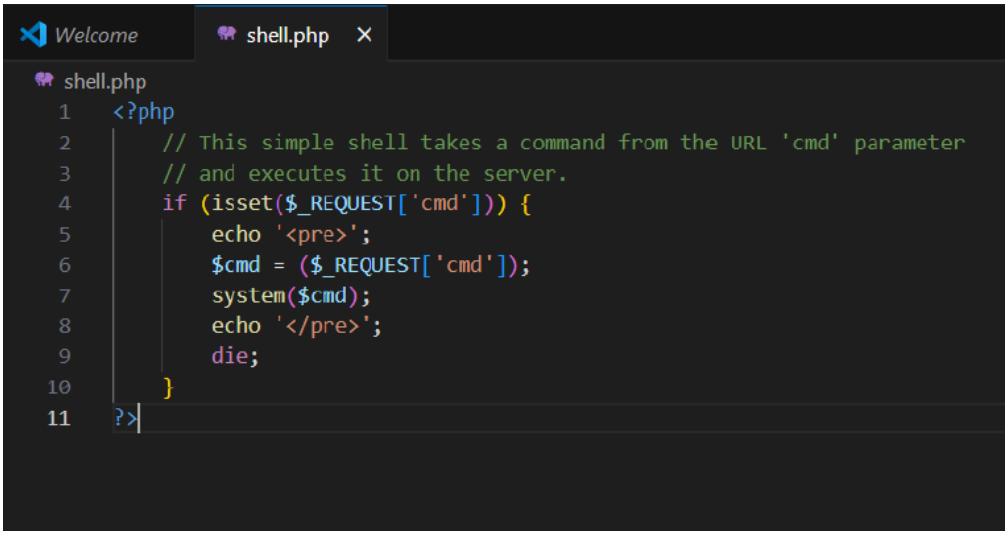
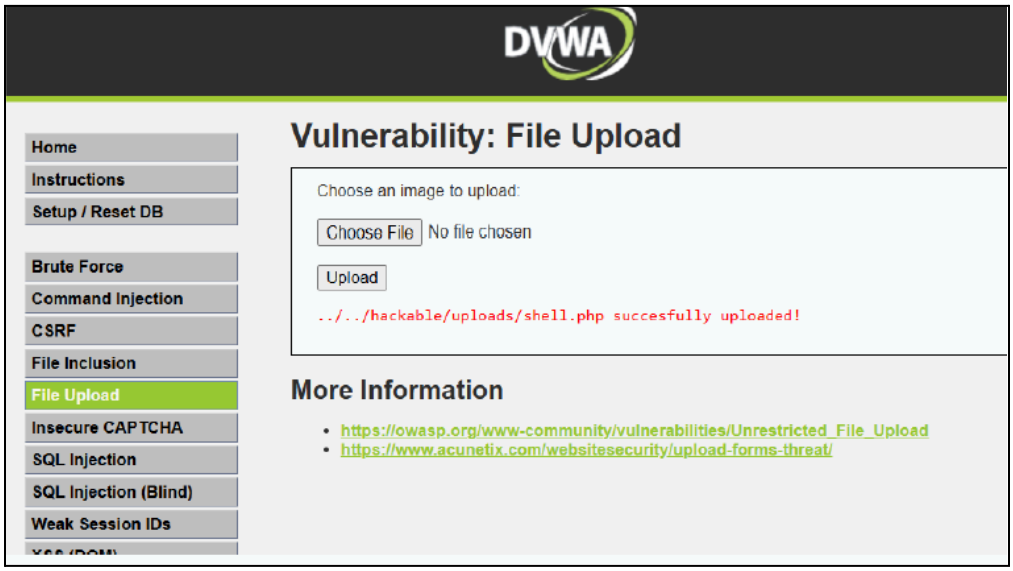
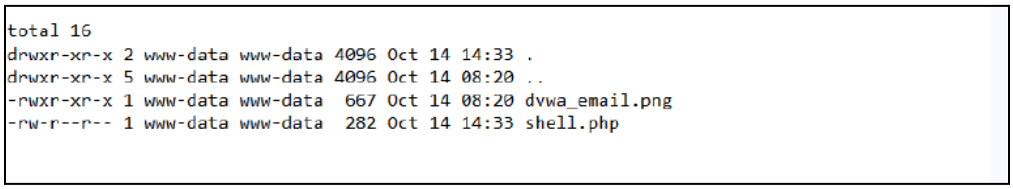
Aspect	Details
Risk Rating	Medium
Exploitation Steps	1. Navigate to the "SQL Injection" page and submit User ID 1. 2. Observe the URL, which contains ?id=1.

	3. Manually modify the URL in the browser's address bar, changing the id parameter to 2, then 3, etc. 4. The application displays the information for other users, proving a lack of authorization checks.
Evidence	
Root Cause Analysis	The application retrieves data based solely on the user-supplied object identifier (the id parameter). It fails to perform an authorization check to verify that the currently logged-in user has the permission to view the requested object.
Proposed Fix	Implement strict, server-side Access Control Checks. For every request, the application must verify that the authenticated user's session is authorized to access the specific resource ID being requested

Part D: File and Functionality Exploitation (Low Security)

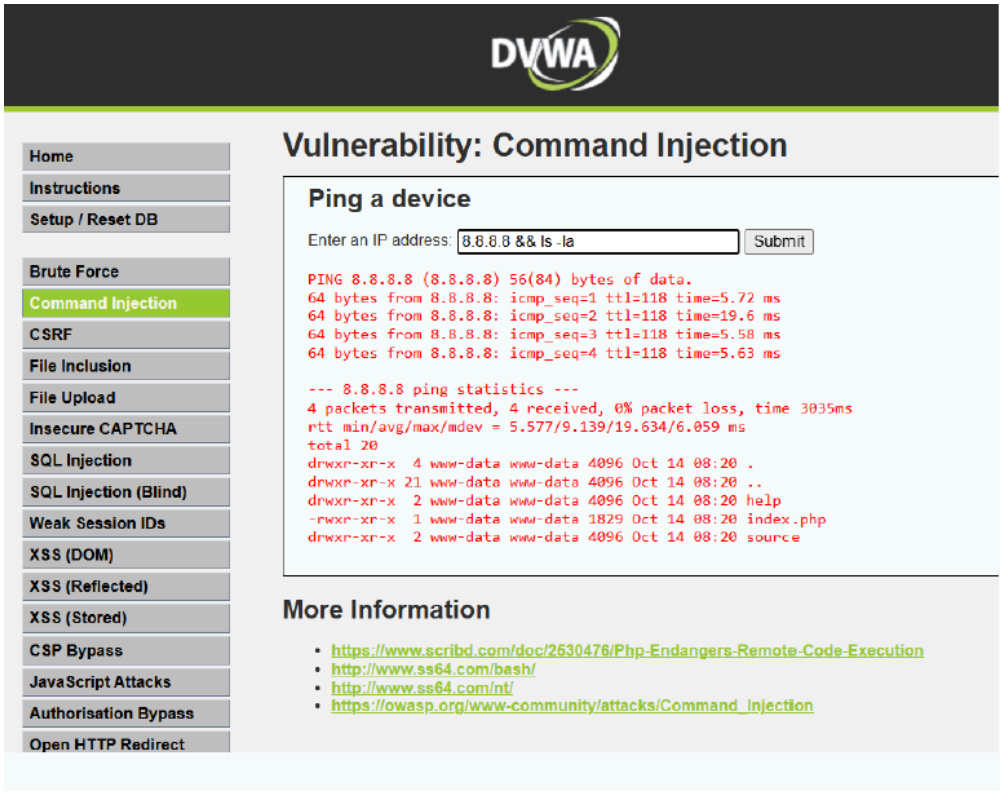
File Upload Vulnerability

Aspect	Details
Risk Rating	High
Exploitation Steps	1. Create a simple PHP web shell and save it as shell.php. 2. Navigate to the "File Upload" page. 3. Upload the shell.php file. The application accepts it without validation. 4. Access the uploaded shell via its URL (.../hackable/uploads/shell.php). 5. Execute OS commands by passing them in a cmd URL parameter (e.g., ?cmd=whoami).

Evidence	  
Root Cause Analysis	The application has no server-side validation to check the file's extension, content type, or contents. It allows executable files (.php) to be uploaded to a web-accessible directory, leading to Remote Code Execution.
Proposed Fix	Implement a multi-layered defense: whitelist safe file extensions, validate the file's MIME type on the server, rename uploaded files to a random string, and store them outside the web root directory.

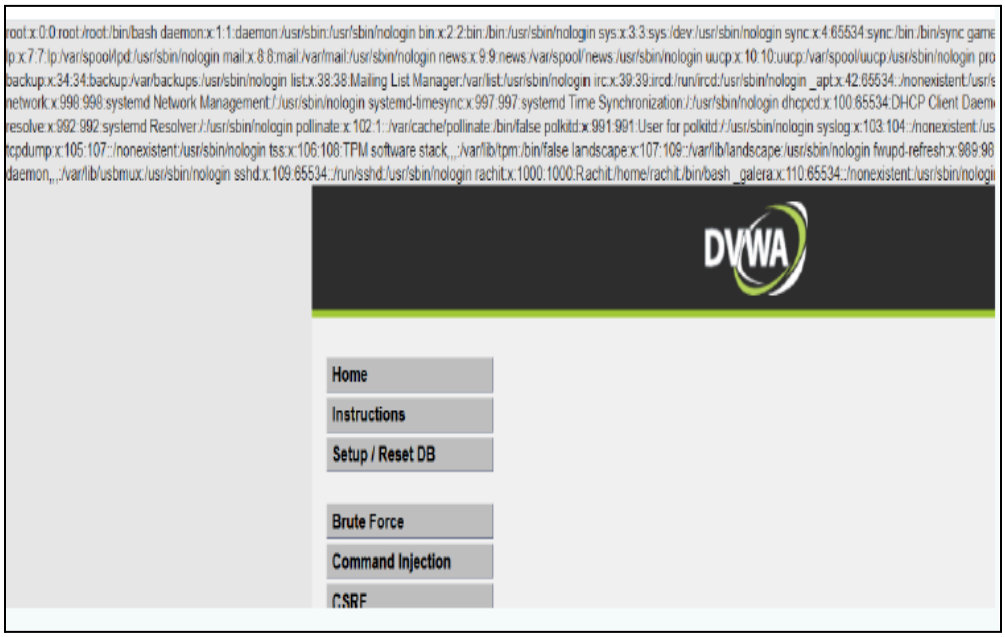
Command Injection :

Aspect	Details
Risk Rating	High

Exploitation Steps	1. Navigate to the "Command Injection" page. 2. In the IP address input box, enter the payload: 8.8.8.8 && ls –la 3. Click "Submit". The application executes both the ping command and the injected ls -la command, displaying the output of both on the page.
Evidence	
Root Cause Analysis	The application takes user input and passes it directly to a system shell command without sanitizing shell metacharacters like &, `
Proposed Fix	The best practice is to avoid calling system commands with user input. If unavoidable, input must be strictly sanitized using a whitelist of allowed characters, and parameters should be passed safely to system calls.

File Inclusion :

Aspect	Details
Risk Rating	High
Exploitation Steps	1. Navigate to the "File Inclusion" page. 2. Observe the page parameter in the URL (?page=include.php). 3. Manipulate the page parameter with a directory traversal payload to read a sensitive system file: ../../../../../../etc/passwd 4. The application includes and displays the contents of the /etc/passwd file.

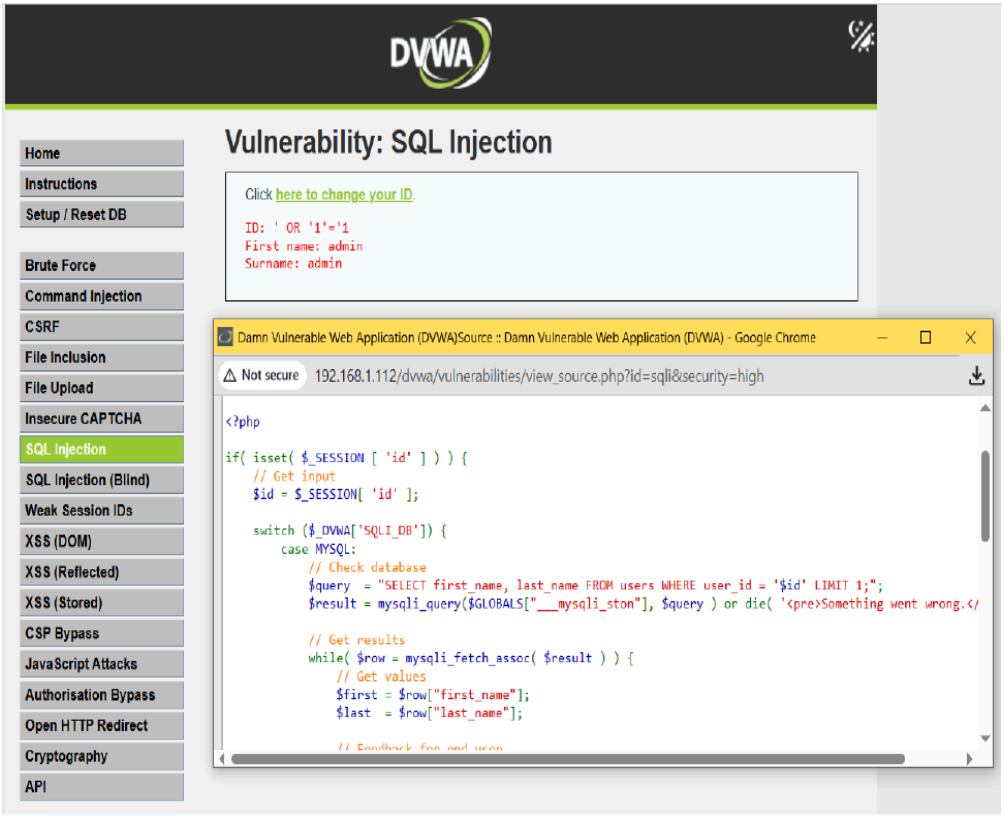
Evidence	
Root Cause Analysis	The application uses user-supplied input directly in a file inclusion function without proper validation. It fails to sanitize or restrict the input, allowing an attacker to use directory traversal sequences (../) to access arbitrary files on the server.
Proposed Fix	Use a whitelist approach. Never accept full filenames or paths from the user. Instead, map clean, user-friendly input (e.g., ?page=about) to a hardcoded, safe file path on the server.

Part E: Defense & Remediation

The following tests were conducted with the DVWA security level set to High to demonstrate the effectiveness of implemented security controls.

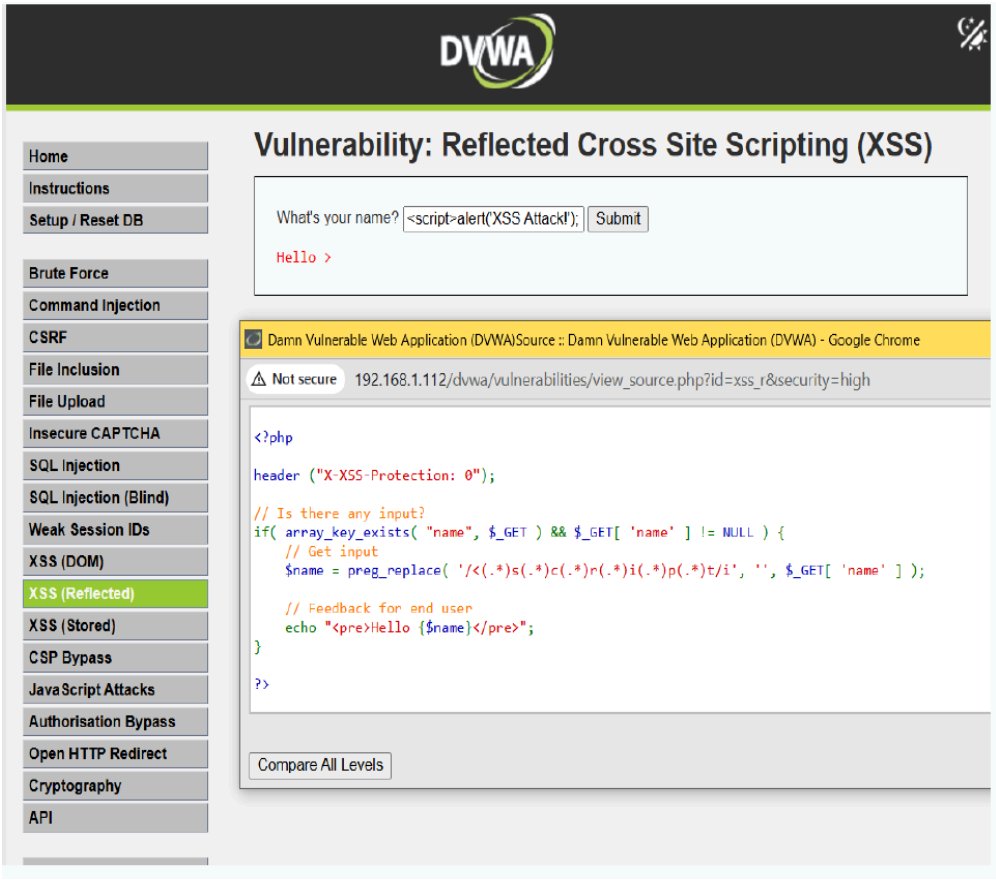
SQL Injection Mitigation

Aspect	Details
Attack Attempt	The "SQL Injection" page was visited. On High security, the input field is removed entirely. Result Failed.
Result	The attack vector is eliminated as the page no longer accepts user input for the query. It securely retrieves data based on the logged-in user's session.

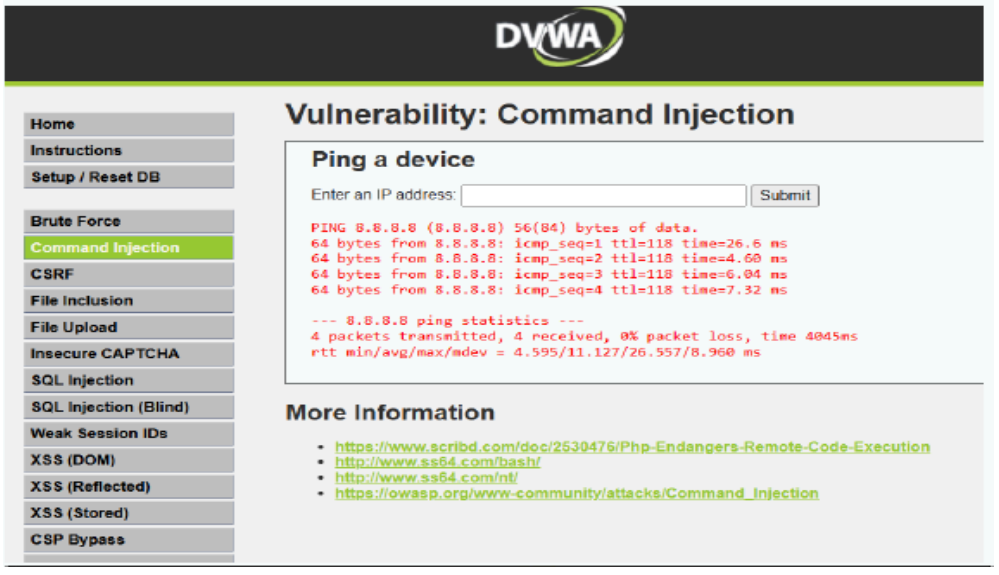
Evidence	

Reflected XSS Mitigation

Aspect	Details
Attack Attempt	The payload <script>alert('XSS Attack!');</script> was submitted on the "XSS (Reflected)" page.
Result	Failed. The script was not executed. Instead, it was rendered as harmless text on the page due to proper output encoding.

Evidence	

Command Injection Mitigation

Aspect	Details
Attack Attempt	The payload 8.8.8.8 && ls -la was submitted on the "Command Injection" page.
Result	Failed. The application rejected the input as invalid because it contained non-IP address characters. The malicious command was not executed.
Evidence	

Section 2: Implementation of Custom Fixes

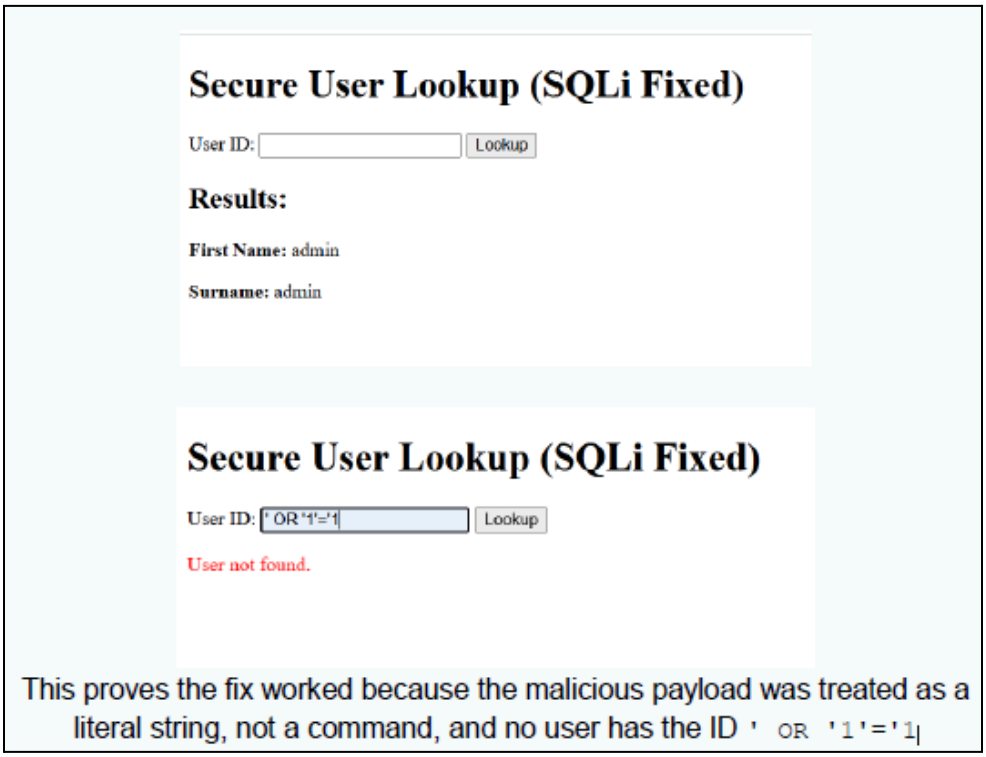
To further demonstrate an understanding of remediation, custom PHP scripts were created and deployed to the server to fix three key vulnerabilities from scratch.

SQL Injection Mitigation (Prepared Statements)

Remediation Script

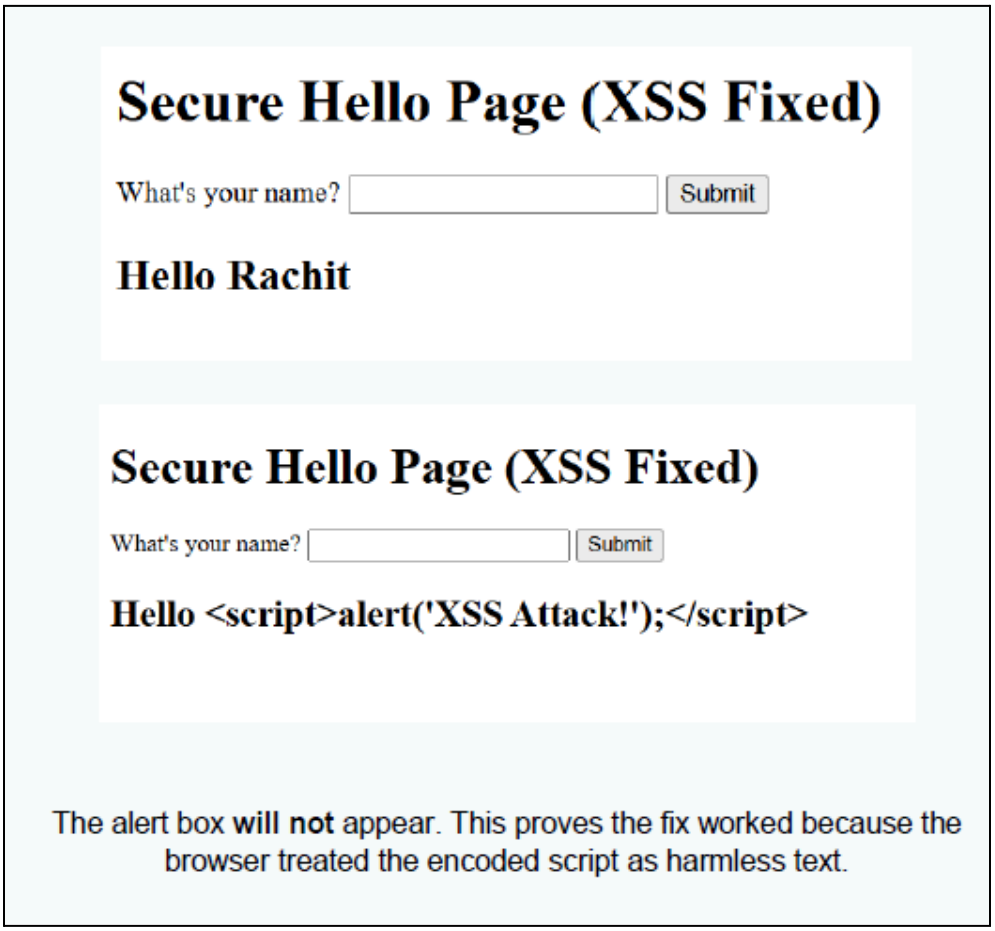
```php
<?php
// fix_sqli.php - Secure user lookup with Prepared Statements

// Database credentials from DVWA's config
$db_server = '127.0.0.1';
$db_user = 'dvwauser';
$db_password = 'dvwapass';
$db_database = 'dvwa';

// Establish a connection
$conn = new mysqli($db_server, $db_user, $db_password,
$db_database);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$user_id = '';
$first_name = '';
$surname = '';
$error_message = '';
if (isset($_GET['id']) && $_GET['id'] != '') {
    $user_id = $_GET['id'];

    // 1. Prepare the statement with a placeholder (?)
    $stmt = $conn->prepare("SELECT first_name, last_name FROM
users WHERE user_id = ?");
    // 2. Bind the user input to the placeholder
    // 's' means the input is treated as a string
    $stmt->bind_param("s", $user_id);
    // 3. Execute the safe query
    $stmt->execute();
    $result = $stmt->get_result();
    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $first_name = $row['first_name'];
        $surname = $row['last_name'];
    } else {
```

	<pre>
```
        $error_message = "User not found.";
    }
    $stmt->close();
}
$conn->close();
?>

<!DOCTYPE html>
<html>
<head>
    <title>Secure User Lookup</title>
</head>
<body>
    <h1>Secure User Lookup (SQLi Fixed)</h1>
    <form method="GET" action="">
        <label for="id">User ID:</label>
        <input type="text" id="id" name="id">
        <input type="submit" value="Lookup">
    </form>

    <?php if ($first_name): ?>
        <h2>Results:</h2>
        <p><strong>First Name:</strong> <?php echo
htmlspecialchars($first_name); ?></p>
        <p><strong>Surname:</strong> <?php echo
htmlspecialchars($surname); ?></p>
    <?php endif; ?>

    <?php if ($error_message): ?>
        <p style="color: red;"><?php echo
htmlspecialchars($error_message); ?></p>
    <?php endif; ?>
</body>
</html>
```
</pre> |
| **Explanation of Fix** | The code uses a prepared statement ($conn->prepare(...)). The user input is never mixed with the SQL query itself. Instead, it is sent to the database separately using bind_param(), ensuring it is always treated as data, not as a command, thus neutralizing the SQL injection attack. |

Evidence	 This proves the fix worked because the malicious payload was treated as a literal string, not a command, and no user has the ID `' OR '1'='1`

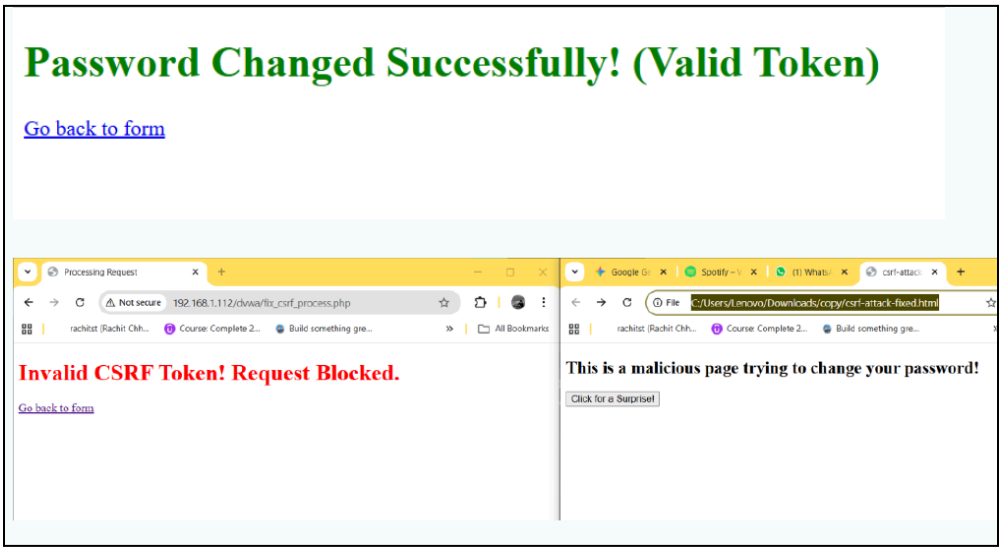
Reflected XSS Mitigation (Output Encoding)

Remediation Script	```php <?php // fix_xss.php - Secure output encoding to prevent Reflected XSS $name = ''; if (isset($_GET['name'])) { $name = $_GET['name']; } ?> <!DOCTYPE html> <html> <head> <title>Secure Hello Page</title> </head> <body> Secure Hello Page (XSS Fixed) <form method="GET" action=""> <label for="name">What's your name?</label> <input type="text" id="name" name="name"> <input type="submit" value="Submit"> </form> <?php if ($name !== ''): ?> ```

	 `` <?php // Use htmlspecialchars() to encode output. // This converts < into < and > into > echo "Hello " . htmlspecialchars($name, ENT_QUOTES, 'UTF-8'); ?> <?php endif; ?> </body> </html> ```
Explanation of Fix	The vulnerability is mitigated by processing all user-supplied output through the htmlspecialchars() function. This function converts characters that have special meaning in HTML (like < and >) into their safe entity equivalents (< and >). This ensures the browser displays the input as plain text rather than executing it as a script.
Evidence	

CSRF Mitigation (Anti-CSRF Tokens)

Remediation Script	<?php // fix_csrf_form.php - A form protected with an Anti-CSRF token session_start(); if (empty($_SESSION['csrf_token'])) { $_SESSION['csrf_token'] = bin2hex(random_bytes(32)); }$token = $_SESSION['csrf_token']; ?> <!DOCTYPE html> <html><head> <title>Secure Password Change</title></head> <body> <h1>Change Your Password (CSRF Protected)</h1> <form method="POST" action="fix_csrf_process.php"> <label for="password">New Password:</label> <input type="password" id="password" name="password_new"> <input type="hidden" name="csrf_token" value="<?php echo htmlspecialchars($token); ?>"> <input type="submit" value="Change Password"> </form></body></html><?php // fix_csrf_process.php - Validates the Anti-CSRF token session_start(); $message = ''; $color = 'red'; // Check if the submitted token matches the one in the session if (isset($_POST['csrf_token'], $_SESSION['csrf_token']) && hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) { $message = "Password Changed Successfully! (Valid Token)"; $color = 'green'; } else { $message = "Invalid CSRF Token! Request Blocked."; $color = 'red'; } unset($_SESSION['csrf_token']); ?> <!DOCTYPE html> <html><head> <title>Processing Request</title></head><body> <h1 style="color: <?php echo $color; ?>;"><?php echo htmlspecialchars($message); ?></h1> Go back to form </body> </html>
Explanation of Fix	This fix prevents CSRF by implementing the Synchronizer Token Pattern. The server requires a secret, unique, and unpredictable token with every state-changing request. An attacker's malicious page cannot guess or access this token, so any forged request submitted from it will be invalid. The hash_equals() function provides a secure way to compare the tokens, protecting against timing attacks

Evidence	

Lessons Learned & Recommended Hardening Checklist

Lessons Learned

The primary lesson from this experiment is that all user-supplied input must be treated as untrusted and potentially malicious. A defense-in-depth strategy is essential, as relying on a single security control is often insufficient. Secure coding is not about a single technique but a mindset of anticipating adversarial actions at every step. Key principles demonstrated include the importance of server-side validation, separating data from commands, implementing strong authorization checks, and ensuring the integrity of user requests.

Recommended LAMP Hardening Checklist

- **Input Validation:**
  - [ ] Use whitelisting over blacklisting for all user input.
  - [ ] Enforce strict data types, character sets, and length limits.
- **Database Security:**
  - [ ] Use Parameterized Queries (Prepared Statements) for all database access to prevent SQLi.
  - [ ] Apply the Principle of Least Privilege: ensure the web application's database user has only **the** minimum required permissions.
- **Output Handling:**
  - [ ] Implement context-aware output encoding for all user-supplied data displayed in HTML, JS, and CSS to prevent XSS.
- **Authentication & Session Management:**
  - [ ] Enforce strong password policies.
  - [ ] Implement account lockout and rate-limiting on login forms to prevent brute-forcing.
  - [ ] Use anti-CSRF tokens for all state-changing actions.
- **Access Control:**
  - [ ] Perform server-side authorization checks for every request to prevent IDOR.

● **File Handling:**
○ [ ] Whitelist allowed file extensions and MIME types for uploads.
○ [ ] Rename all uploaded files to a random string.
○ [ ] Store uploaded files outside of the web root directory.
● **System Interaction:**
○ [ ] Avoid passing user input to system shell commands. Use language-native functions where possible.
○ [ ] If shell commands are necessary, strictly sanitize all input.

Conclusion :

During this experiment, I worked on identifying and fixing typical web application flaws using DVWA, including session fixation, SQL injection, insecure direct object references, and reflected XSS. The activity showed how weak validation or session handling can lead to attacks. Implementing input checks, parameterized queries, secure sessions, and output encoding proved effective in improving the overall security of web applications.