SOP For Vulnerabilities

Share To	**Land and Development (eDharti)**
Purpose:	**To mitigate the same**
Date:	**02-05-2025**
For Vulnerability	**All mentioned bugs in Report**
Reference No	**PTS-01-2024/LOD/001**

Vulnerabilities Reported:
#PT-LDOE-BUG ID: 01 – HTML Injection On Multiple Points
#PT-LDOE-BUG ID: 02 – XSS – Cross Site Scripting
#PT-LDOE-BUG ID: 03 – OTP Brute-Force
#PT-LDOE-BUG ID: 04 – Stored XSS In Description and Address
#PT-LDOE-BUG ID: 05 – Debug Mode Enabled
#PT-LDOE-BUG ID: 06 – SQL Query Disclosure On Stack Trace Error
#PT-LDOE-BUG ID: 07 – Session Hijacking
#PT-LDOE-BUG ID: 08 – ClickJacking
#PT-LDOE-BUG ID: 09 – Outdated Jquery Version
#PT-LDOE-BUG ID: 10 – Information Disclosure via X-Powered-By Header
#PT-LDOE-BUG ID: 11 – Server Version Disclosure (nginx)
#PT-LDOE-BUG ID: 12 – Strict-Transport-Security Header Not Set
#PT-LDOE-BUG ID: 13 – Cookie Without Secure Flag Set
#PT-LDOE-BUG ID: 14 – Cookie Without Httponly Flag Set
#PT-LDOE-BUG ID: 15 – CSP Header Is Missing
#PT-LDOE-BUG ID: 16 - X-Frame-Option Header Not Set
#PT-LDOE-BUG ID: 17 - HTTP Strict Transport Security Policy Header Not Set
#PT-LDOE-BUG ID: 18 - X-XSS-Protection Header Is Missing
#PT-LDOE-BUG ID: 19 - PHP Version Is Outdated

Bug ID: 01 – HTML Injection On Multiple Points

Description: HTML injection occurs when untrusted input is rendered as HTML, allowing attackers to inject malicious HTML code.
Solution:

- Sanitize and escape all user inputs before rendering them in HTML. Use PHP's htmlspecialchars() function to escape special characters.
- Validate input to ensure it conforms to expected formats (e.g., no HTML tags).
  Code-Level Fix:

```
// Sanitize and escape user input
$user_input = htmlspecialchars($_POST['input'], ENT_QUOTES, 'UTF-8');
echo $user_input; // Safe output
```

- Use a library like HTMLPurifier for advanced HTML sanitization if complex HTML input is allowed.
  Server-Level Fix:
- Ensure Nginx is not misconfigured to interpret user input as HTML (rare, but verify MIME types).
- Sanitize all user inputs using `htmlspecialchars()` or `filter_var()` in PHP.
- Escape output before rendering on the page.

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

Bug ID: 02 – XSS – Cross Site Scripting

Description: XSS occurs when malicious scripts are injected into web pages viewed by users.
Solution:

- Escape output using htmlspecialchars() for HTML contexts.
- Use json_encode() for JavaScript contexts.
- Implement Content Security Policy (CSP) to restrict script sources (see Bug ID: 15).
  Code-Level Fix:

```
// Escape output in HTML
echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
// Escape output in JavaScript
echo '<script>var data = ' . json_encode($user_data) . ';</script>';
```

Server-Level Fix:

- Add XSS protection headers (see Bug ID: 18).
- Sanitize input and encode output as above.
- Use Content Security Policy (CSP).
- Prefer server-side validation and escaping.

Consider a security library like HTMLPurifier

Bug ID: 03 – OTP Brute-Force

Description: Attackers can brute-force OTPs due to lack of rate limiting or lockout mechanisms.
Solution:

- Implement rate limiting for OTP submission attempts.
- Add account lockout after a set number of failed attempts.
- Use CAPTCHA to prevent automated submissions.

  Code-Level Fix:

```
session_start();
// Track OTP attempts
if (!isset($_SESSION['otp_attempts'])) {
    $_SESSION['otp_attempts'] = 0;
    $_SESSION['otp_lockout_time'] = 0;
}

// Check for lockout
if ($_SESSION['otp_lockout_time'] > time()) {
    die("Account locked. Try again later.");
}

// Validate OTP
if ($_POST['otp'] === $expected_otp) {
    $_SESSION['otp_attempts'] = 0; // Reset on success
} else {
    $_SESSION['otp_attempts']++;
    if ($_SESSION['otp_attempts'] >= 5) {
        $_SESSION['otp_lockout_time'] = time() + 900; // Lock for 15
minutes
        die("Too many attempts. Account locked.");
    }
    die("Invalid OTP.");
}
```

Server-Level Fix:

- Use Nginx rate limiting to restrict OTP endpoint requests.

```
limit_req_zone $binary_remote_addr zone=otp_limit:10m rate=5r/m;
server {
    location /otp_endpoint {
        limit_req zone=otp_limit burst=10;
    }
}
```

- Implement rate-limiting and account lockout after a number of failed attempts.
- Use CAPTCHA after multiple failures.
- Log all OTP attempts with IP for anomaly detection.

Bug ID: 04 – Stored XSS In Description and Address

Description: Stored XSS occurs when malicious scripts are stored in the database and served to users.
Solution:

- Sanitize inputs before storing them in the database using htmlspecialchars() or HTMLPurifier.
- Escape outputs when displaying stored data.
  Code-Level Fix:

```
// Sanitize input before storing
$description = htmlspecialchars($_POST['description'], ENT_QUOTES, 'UTF-
8');
$address = htmlspecialchars($_POST['address'], ENT_QUOTES, 'UTF-8');
// Store in database
$stmt = $pdo->prepare("INSERT INTO table (description, address) VALUES (?,
?)");
$stmt->execute([$description, $address]);

// Escape output when displaying
echo htmlspecialchars($row['description'], ENT_QUOTES, 'UTF-8');
```

Server-Level Fix:

- Same as Bug ID: 02 (CSP and XSS headers).
- Sanitize inputs before storing.
- Escape all dynamic output with `htmlspecialchars()` when rendering.
- Prevent rich text entry unless sanitized via a library like HTMLPurifier.

Bug ID: 05 – Debug Mode Enabled

Description: Debug mode exposes sensitive information like stack traces.
Solution:

- Disable PHP debug mode in production.
- Turn off display_errors and log errors to a file instead.
  Code-Level Fix:
  Edit php.ini or add to your PHP code:

```
ini_set('display_errors', '0');
ini_set('display_startup_errors', '0');
error_reporting(E_ALL);
ini_set('log_errors', '1');
ini_set('error_log', '/path/to/error.log');
```

Server-Level Fix:

- Ensure Nginx does not expose error details. Use custom error pages:

```
error_page 500 502 503 504 /custom_error.html;
```

- Ensure `display_errors = Off` in `php.ini`.
- Set `error_reporting(0)` in production.
- Remove any `phpinfo()` calls or development tools.

Bug ID: 06 – SQL Query Disclosure On Stack Trace Error

Description: Stack traces reveal SQL queries, aiding attackers in crafting exploits.
Solution:

- Disable debug mode (see Bug ID: 05).
- Use prepared statements to prevent SQL injection.
- Handle errors gracefully without exposing stack traces.
  Code-Level Fix:

```
try {
    $stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
    $stmt->execute([$_GET['id']]);
} catch (PDOException $e) {
    // Log error without exposing details
    error_log($e->getMessage());
    http_response_code(500);
    echo "An error occurred.";
    exit;
}
```

- Never display stack traces or SQL errors to users.
- Use generic error messages and log technical details server-side.
- Wrap database queries in try-catch blocks.

Bug ID: 07 – Session Hijacking

Description: Weak session management allows attackers to steal session cookies.
Solution:

- Use secure, HTTP-only cookies.
- Regenerate session IDs on login.
- Implement session expiration.
  Code-Level Fix:

```
session_start([
    'cookie_secure' => true,
    'cookie_httponly' => true,
    'cookie_samesite' => 'Strict',
]);
session_regenerate_id(true); // Regenerate session ID on login
// Set session timeout (e.g., 30 minutes)
if (isset($_SESSION['last_activity']) && (time() -
$_SESSION['last_activity'] > 1800)) {
    session_unset();
    session_destroy();
}
$_SESSION['last_activity'] = time();
```

Server-Level Fix:

- Enforce HTTPS (see Bug ID: 12).
- Enable `session.cookie_httponly = 1` and `session.cookie_secure = 1` in `php.ini`.
- Regenerate session ID on login: `session_regenerate_id(true);`
- Bind sessions to IP/user-agent if feasible.

Bug ID: 08 – ClickJacking

Description: Lack of frame protection allows attackers to embed your site in an iframe.
Solution:

- Set the X-Frame-Options header to prevent framing.
  Server-Level Fix:

```
server {
    add_header X-Frame-Options "DENY" always;
}
```

Add this to your Nginx config:

```
add_header X-Frame-Options "DENY";
```

Bug ID: 09 – Outdated jQuery Version

Description: Using an outdated jQuery version exposes the app to known vulnerabilities.
Solution:

- Update jQuery to the latest stable version (e.g., 3.7.1 as of 2025).
  Code-Level Fix:
  Update your HTML:

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
```

- Audit other JavaScript libraries for outdated versions.
- Upgrade to the latest jQuery version.

Bug ID: 10 – Information Disclosure via X-Powered-By Header

Description: The X-Powered-By header reveals PHP version information.
Solution:

- Disable the X-Powered-By header.
  Code-Level Fix:
  Edit php.ini:

```
expose_php = Off
```

Server-Level Fix:

- Remove the header in Nginx:

```
server {
    fastcgi_hide_header X-Powered-By;
}
```

In PHP config (`php.ini`):

```
expose_php = Off
```

Or in Nginx:

```
fastcgi_hide_header X-Powered-By;
```

Bug ID: 11 – Server Version Disclosure (nginx)

Description: Nginx exposes its version in HTTP headers, aiding attackers.
Solution:

- Hide the server version.
  Server-Level Fix:
  Edit nginx.conf:

```
http {
    server_tokens off;
}
```

In `nginx.conf`:

```
server_tokens off;
```

Bug ID: 12, 17 – Strict-Transport-Security Header Not Set

Description: Missing HSTS header allows connections over HTTP, risking man-in-the-middle attacks.
Solution:

- Enable HSTS to enforce HTTPS.
  Server-Level Fix:

```
server {
    add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains" always;

}
```

Add to Nginx config (HTTPS block):

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
```

Bug ID: 13 – Cookie Without Secure Flag Set

Description: Cookies without the Secure flag can be sent over HTTP.
Solution:

- Set the Secure flag for cookies.
  Code-Level Fix:

```
session_start([
    'cookie_secure' => true,
]);
setcookie('name', 'value', ['secure' => true]);


and also alternate as:
```

In PHP

```
setcookie("name", "value", [
  'secure' => true,
  'httponly' => true,
  'samesite' => 'Strict'
]);
```

Set session cookies in `php.ini`:

```
session.cookie_secure = 1
```

Bug ID: 14 – Cookie Without HttpOnly Flag Set

Description: Cookies without HttpOnly flag are accessible via JavaScript, risking XSS attacks.
Solution:

- Set the HttpOnly flag for cookies.
  Code-Level Fix:

```
session_start([
    'cookie_httponly' => true,
]);
setcookie('name', 'value', ['httponly' => true]);
```

Same as above, ensure:

```
session.cookie_httponly = 1
```

Bug ID: 15 – CSP Header Is Missing

Description: Missing Content Security Policy allows unsafe script execution.
Solution:

- Implement a strict CSP header.
  Server-Level Fix:

```
server {
    add_header Content-Security-Policy "default-src 'self'; script-src
'self' https://code.jquery.com; style-src 'self'; img-src 'self'; connect-
src 'self'; frame-ancestors 'none';" always;
}
```

- Adjust the CSP based on your app's needs (e.g., allow specific external scripts).
- Add to Nginx:
- `add_header Content-Security-Policy "default-src 'self'; script-src 'self'; object-src 'none';";`
-

Bug ID: 16 – X-Frame-Options Header Not Set

Description: Missing X-Frame-Options header allows clickjacking (same as Bug ID: 08).
Solution:

- Same fix as Bug ID: 08.

Same as Clickjacking fix:

```
add_header X-Frame-Options "DENY";
```

Bug ID: 17

- Same fix as Bug ID: 12.

Bug ID: 18 – X-XSS-Protection Header Is Missing

Description: Missing X-XSS-Protection header disables browser XSS filtering (legacy but useful).
Solution:

- Enable the header.
  Server-Level Fix:

```
server {
    add_header X-XSS-Protection "1; mode=block" always;
}
```

Add to Nginx:

```
add_header X-XSS-Protection "1; mode=block";
```

Bug ID: 19 – PHP Version Is Outdated

Description: An outdated PHP version has known vulnerabilities.
Solution:

- Upgrade to the latest stable PHP version (e.g., PHP 8.3 or 8.4 as of 2025).
- Update dependencies and test the application for compatibility.
  Server-Level Fix:
- Update PHP on the server:

```
sudo apt update
sudo apt install php8.3 php8.3-fpm php8.3-mysql php8.3-curl
```

- Update Nginx to use the new PHP version:

```
location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.3-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}
```

- Update to the latest stable PHP version (`apt upgrade php`).
- Remove unused/old PHP packages.

Consolidated Nginx Configuration

Here’s a consolidated Nginx configuration to address server-level fixes:

```
http {
    server_tokens off; # Hide Nginx version
    limit_req_zone $binary_remote_addr zone=otp_limit:10m rate=5r/m;

    server {
        listen 443 ssl;
        ssl_certificate /path/to/cert.crt;
        ssl_certificate_key /path/to/cert.key;

        # Security headers
        add_header X-Frame-Options "DENY" always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header Strict-Transport-Security "max-age=31536000;
includeSubDomains" always;
        add_header Content-Security-Policy "default-src 'self'; script-src
'self' https://code.jquery.com; style-src 'self'; img-src 'self'; connect-
src 'self'; frame-ancestors 'none';" always;
        fastcgi_hide_header X-Powered-By;

        # Rate limiting for OTP
        location /otp_endpoint {
            limit_req zone=otp_limit burst=10;
        }

        # PHP processing
        location ~ \.php$ {
            fastcgi_pass unix:/var/run/php/php8.3-fpm.sock;
            fastcgi_index index.php;
            include fastcgi_params;
        }

        # Custom error pages
        error_page 500 502 503 504 /custom_error.html;
    }
}
```

General Recommendations

1. **Input Validation**: Always validate and sanitize user inputs using allowlists for expected formats.
2. **Regular Updates**: Keep PHP, Nginx, and all libraries (e.g., jQuery) up to date.
3. **Security Audits**: Regularly audit your application using tools like OWASP ZAP or Burp Suite.
4. **Backups**: Maintain regular backups of your application and database.
5. **Logging**: Log security events (e.g., failed OTP attempts) for monito

END OF REPORT