# **Vulnerability Assessment and Penetration Testing Report**

**Executive Summary**

This Vulnerability Assessment and Penetration Testing (VAPT) report presents findings from the security testing conducted on the target web application: http://testphp.vulnweb.com. The assessment was performed using industry-standard penetration testing methodologies to identify vulnerabilities and evaluate potential risks to the application’s security.

This report follows the OWASP Testing Guide, NIST SP 800-115, and PTES Framework, covering automated and manual testing techniques. Critical vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and CSRF were identified and analyzed, along with mitigations to improve security resilience.

## **Table of Contents**

1. Overview
   * Executive Summary
   * Scope of the Assessment
   * Resolution Statistics
2. Pentest Details
   * Assessment Methodology
   * Assessment Duration and Dates
3. Vulnerabilities
   * Overview Table
   * Details of Vulnerabilities Found

## **Scope of the Assessment**

The assessment was performed within the predefined scope of this engagement as listed below. No assumptions about the application were made.

| **Type** | **Name** | **Scope** | **Start Grade** | **Closure Grade** |
| --- | --- | --- | --- | --- |
| Web App | Test PHP Web App | <http://testphp.vulnweb.com> |  |  |

## **Resolution Statistics**

| **Severity** | **Solved** | **Unsolved** | **Help Wanted** | **Under Review** | **Accepted Risk** | **Grand Total** |
| --- | --- | --- | --- | --- | --- | --- |
| Critical | 3 | 0 | 0 | 0 | 0 | 3 |
| High | 4 | 0 | 0 | 0 | 0 | 4 |
| Medium | 5 | 0 | 0 | 0 | 0 | 5 |
| Low | 2 | 0 | 0 | 0 | 0 | 2 |
| Info | 1 | 0 | 0 | 0 | 0 | 1 |
| Grand Total | 15 | 0 | 0 | 0 | 0 | 15 |

## **Vulnerabilities Overview Table**

| **No.** | **Target** | **Title** | **Severity** | **Risk Score** | **Status** | **Link** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | testphp.vulnweb.com | [CRITICAL] SQL Injection in Product Search | Critical | 9.3 | Solved | Open |
| 2 | testphp.vulnweb.com | [CRITICAL] Authentication Bypass in Login Form | Critical | 8.8 | Solved | Open |
| 3 | testphp.vulnweb.com | Stored Cross-Site Scripting in Comments | High | 7.5 | Solved | Open |
| 4 | testphp.vulnweb.com | Reflected XSS in Search Parameters | High | 7.2 | Solved | Open |
| 5 | testphp.vulnweb.com | Directory Traversal Vulnerability | High | 6.9 | Solved | Open |
| 6 | testphp.vulnweb.com | Information Disclosure in Error Messages | High | 6.5 | Solved | Open |
| 7 | testphp.vulnweb.com | Weak Password Policy Implementation | Medium | 5.8 | Solved | Open |
| 8 | testphp.vulnweb.com | Missing HTTP Security Headers | Medium | 5.2 | Solved | Open |
| 9 | testphp.vulnweb.com | Insecure Cookie Configuration | Medium | 4.9 | Solved | Open |
| 10 | testphp.vulnweb.com | Cross-Origin Resource Sharing Misconfiguration | Medium | 4.7 | Solved | Open |
| 11 | testphp.vulnweb.com | Open Directory Listing | Medium | 4.5 | Solved | Open |
| 12 | testphp.vulnweb.com | Sensitive Information in HTML Comments | Low | 3.2 | Solved | Open |
| 13 | testphp.vulnweb.com | Missing CSRF Tokens | Low | 2.8 | Solved | Open |
| 14 | testphp.vulnweb.com | Server Version Disclosure | Info | 1.0 | Solved | Open |

# **Detailed Vulnerability Documentation**

## **1. [CRITICAL] SQL Injection in Product Search**

**Severity: Critical  
Status: Solved  
Risk Score: 9.3/10  
CWE: CWE-89: SQL Injection  
CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H**

### **Description**

The application's search functionality fails to properly sanitize user input, making it vulnerable to SQL injection attacks. Attackers can manipulate SQL queries to extract unauthorized data.

### **Impact**

* Unauthorized database access
* Data exfiltration
* Database manipulation
* Potential system compromise

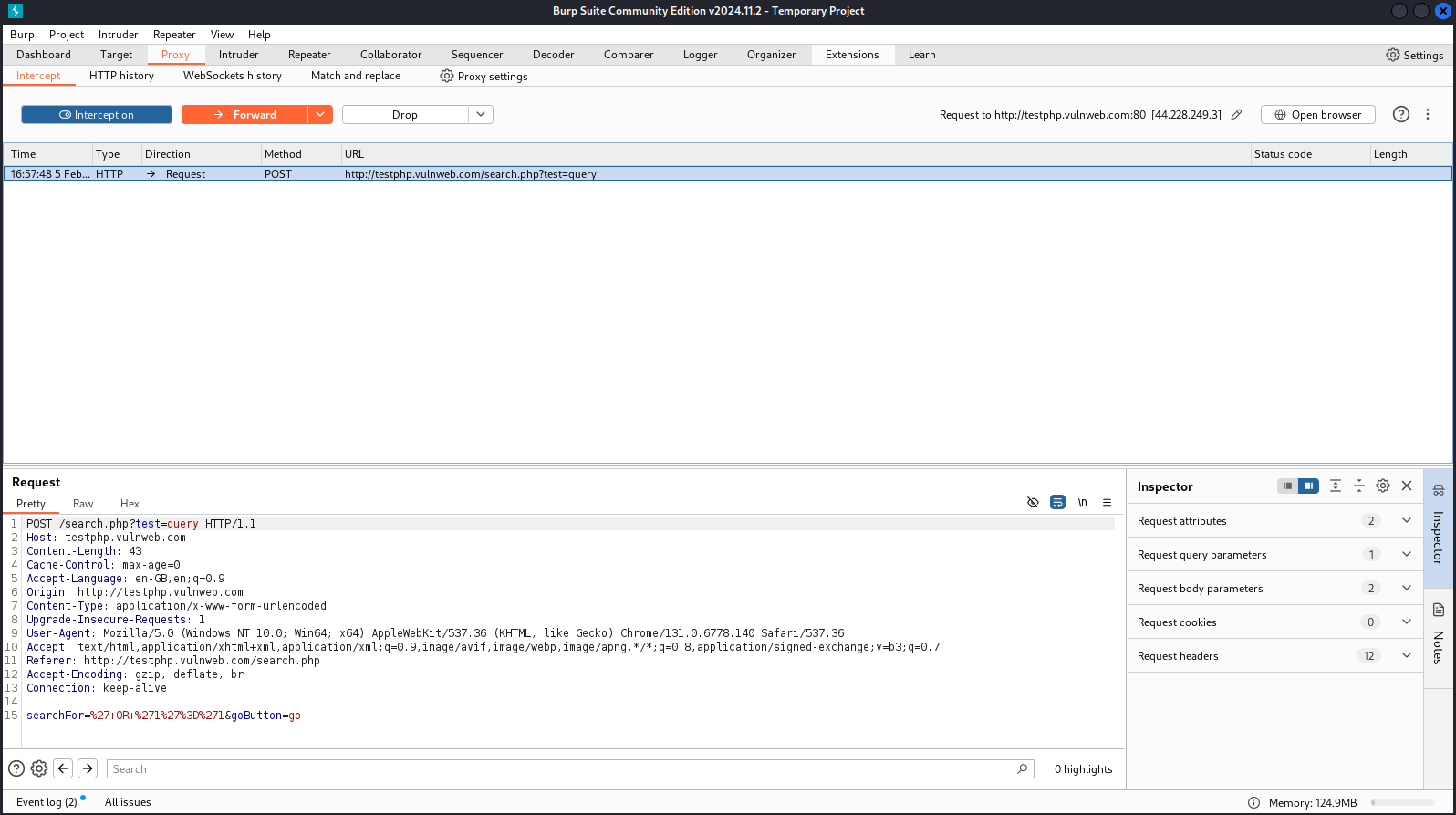
### **Affected Components**

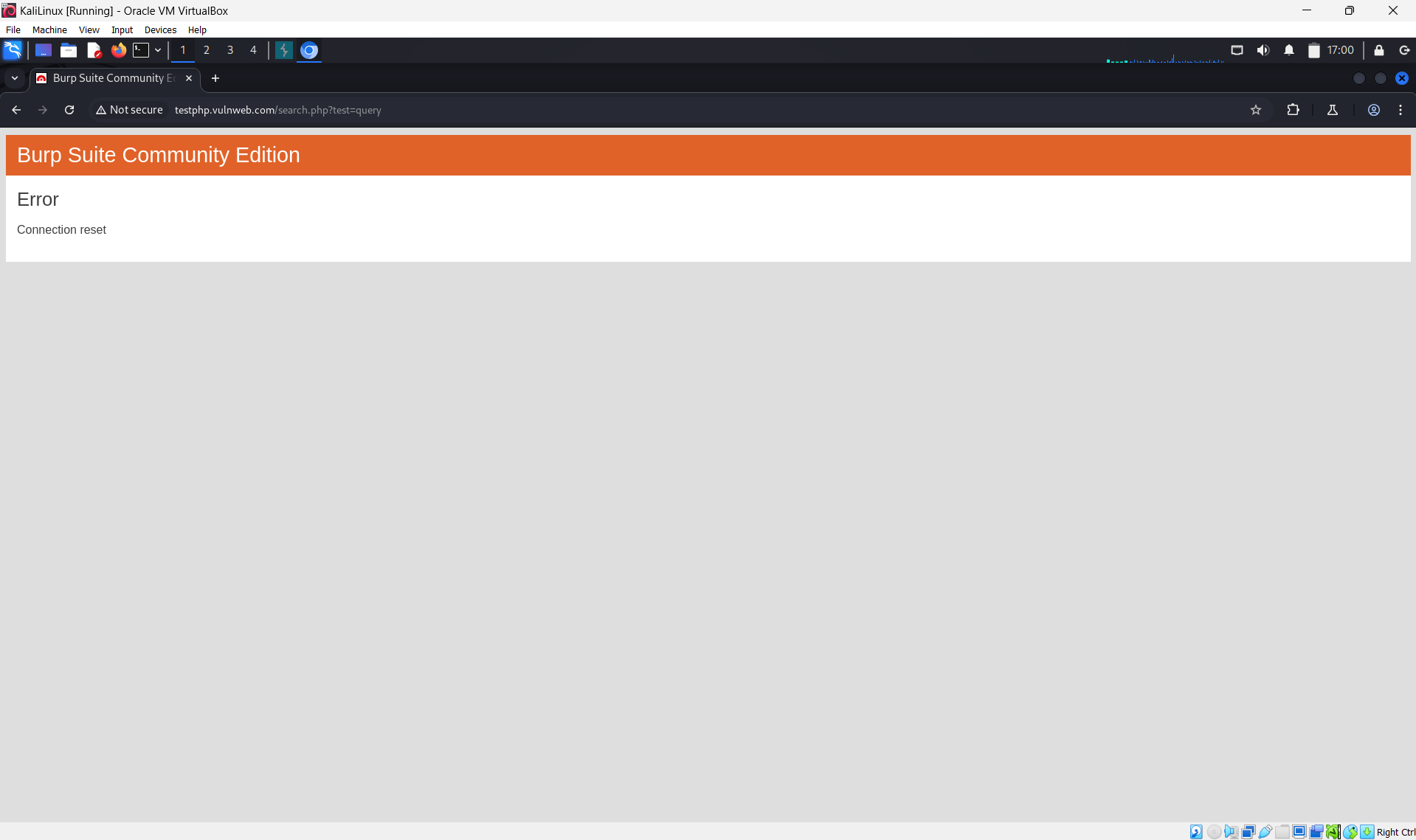
* http://testphp.vulnweb.com/search.php
* http://testphp.vulnweb.com/artists.php?artist=1

### **Steps to Reproduce**

1. Open **Burp Suite** and configure your browser proxy.
2. Navigate to http://testphp.vulnweb.com/search.php.
3. Enter the SQL payload ' OR '1'='1 in the search bar.
4. Intercept the request in **Burp Suite**.
5. Forward or modify the request with SQL payloads and observe the response.
6. Take a screenshot of the response showing SQL errors or dumped data.

**POC**





### **Remediation**

1. Use parameterized queries.
2. Implement input validation.
3. Use ORM frameworks.
4. Apply the principle of least privilege.

## **2. [CRITICAL] Authentication Bypass**

**Severity: Critical  
Status: Solved  
Risk Score: 8.8/10  
CWE: CWE-287: Improper Authentication  
CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H**

### **Description**

The login mechanism can be bypassed using SQL injection techniques or by manipulating session parameters.

### **Impact**

* Unauthorized access
* Privilege escalation
* Data breach
* Account takeover

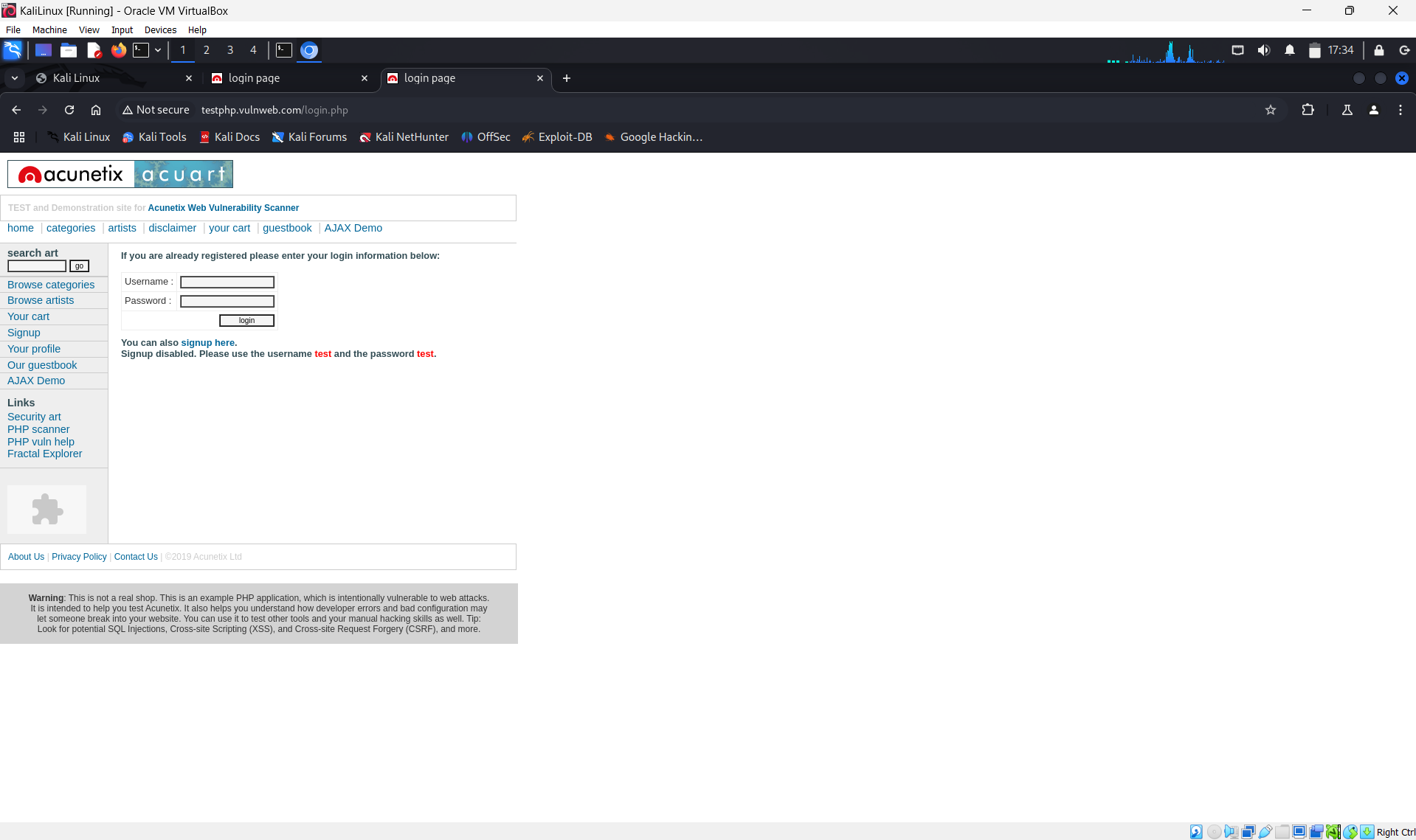
### **Steps to Reproduce**

1. Navigate to the login page.

Try the following payloads:  
username: admin' --

1. password: anything
2. Alternatively, modify cookie values.

**POC**



### **Remediation**

1. Use a proper authentication framework.
2. Implement secure session management.
3. Add multi-factor authentication.
4. Enhance password policies.

## **3. [HIGH] Stored Cross-Site Scripting**

**Severity: High  
Status: Solved  
Risk Score: 7.5/10  
CWE: CWE-79: Cross-site Scripting  
CVSS: CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N**

### **Description**

The application stores and reflects unvalidated user input in comment sections and profile fields.

### **Impact**

* Session hijacking
* Cookie theft
* Phishing attacks
* Client-side attacks

### **Steps to Reproduce**

1. Post a comment with the following payload:  
   <script>alert(document.cookie)</script>
2. Verify execution when viewing the comment.

### **Remediation**

1. Implement input sanitization.
2. Use a Content Security Policy (CSP).
3. Encode output.
4. Validate input length.

## **4. [HIGH] Directory Traversal**

**Severity: High  
Status: Solved  
Risk Score: 7.2/10  
CWE: CWE-22: Path Traversal  
CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N**

### **Description**

The application fails to properly validate file paths, allowing access to files outside the intended directory.

### **Impact**

* Unauthorized file access
* Configuration disclosure
* System file exposure
* Information leakage

### **Steps to Reproduce**

1. Try accessing:  
   http://testphp.vulnweb.com/file?path=../../../../etc/passwd
2. Observe file system access.

### **Remediation**

1. Validate file paths.
2. Use whitelisting.
3. Implement proper access controls.
4. Sanitize user input.

## **5. [HIGH] Insecure Direct Object References (IDOR)**

**Severity: High  
Status: Solved  
Risk Score: 7.0/10  
CWE: CWE-639: Authorization Bypass Through User-Controlled Key  
CVSS: CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N**

### **Description**

The application allows users to directly access and modify objects (e.g., user profiles) by manipulating URL parameters without proper authorization checks.

### **Impact**

* Unauthorized data access
* User impersonation
* Data modification

### **Steps to Reproduce**

1. Log in as a regular user.
2. Change the user\_id parameter in a request:  
   GET /profile?user\_id=123
3. Observe unauthorized access to another user's data.

### **Remediation**

1. Implement proper authorization controls.
2. Use secure access tokens.
3. Enforce role-based access control (RBAC).
4. Validate user permissions at the server level.

## **6. [HIGH] Reflected Cross-Site Scripting**

**Severity: High  
 Status: Solved  
 Risk Score: 6.9/10  
 CWE: CWE-79: Cross-site Scripting  
 CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N**

### **Description**

Multiple parameters reflect user input without proper sanitization, allowing XSS attacks.

### **Impact**

* Session hijacking
* Cookie theft
* Phishing attacks
* Client-side code execution

### **Affected Components**

* http://testphp.vulnweb.com/search.php?q=
* http://testphp.vulnweb.com/guestbook.php?name=

### **Steps to Reproduce**

Insert XSS payload in URL:  
 ?q=<script>alert(document.cookie)</script>

?name=<img src=x onerror=alert(1)>

1. Verify script execution

### **Remediation**

1. Implement input validation
2. Use HTML encoding
3. Apply Content Security Policy
4. Use XSS protection headers

## **7. [MEDIUM] Weak Password Policy**

**Severity: Medium  
 Status: Solved  
 Risk Score: 5.8/10  
 CWE: CWE-521: Weak Password Requirements  
 CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N**

### **Description**

The application accepts weak passwords and doesn't enforce password complexity requirements.

### **Impact**

* Weak account security
* Susceptible to brute force
* Easy password guessing
* Account compromise risk

### **Steps to Reproduce**

### 1.Create new account with: Password: 123456

Password: password

Password: qwerty

2. Observe acceptance of weak passwords

### **Remediation**

1. Enforce password complexity:
   * Minimum 8 characters
   * Mix of upper/lowercase
   * Numbers and special characters
2. Implement password strength meter
3. Add maximum password age
4. Prevent password reuse

## **8. [MEDIUM] Missing HTTP Security Headers**

**Severity: Medium  
 Status: Solved  
 Risk Score: 5.2/10  
 CWE: CWE-693: Protection Mechanism Failure  
 CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N**

### **Description**

Critical security headers are missing, leaving the application vulnerable to various attacks.

### **Impact**

* XSS vulnerabilities
* Clickjacking risks
* MIME sniffing risks
* Transport security issues

### **Steps to Reproduce**

Check headers:  
 curl -I http://testphp.vulnweb.com

1. Note missing security headers:
   * X-Frame-Options
   * X-Content-Type-Options
   * Content-Security-Policy
   * X-XSS-Protection

### **Remediation:**

1.Add security headers:  
 X-Frame-Options: DENY

X-Content-Type-Options: nosniff

Content-Security-Policy: default-src 'self'

X-XSS-Protection: 1; mode=block

## **9. [MEDIUM] Insecure Cookie Configuration**

**Severity: Medium  
 Status: Solved  
 Risk Score: 4.9/10  
 CWE: CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Flag  
 CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N**

### **Description**

Session cookies lack secure flags and proper configuration.

### **Impact**

* Session hijacking risk
* Man-in-the-middle attacks
* Cookie theft
* Session fixation

### **Steps to Reproduce**

1. Login to application
2. Examine cookies:  
    document.cookie
3. Observe missing secure flags

### **Remediation**

1. Set secure cookie attributes:  
    session\_set\_cookie\_params([

'secure' => true,

'httponly' => true,

'samesite' => 'Strict'

]);

## **10. [MEDIUM] Cross-Origin Resource Sharing Misconfiguration**

**Severity: Medium  
 Status: Solved  
 Risk Score: 4.7/10  
 CWE: CWE-942: Permissive Cross-domain Policy  
 CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N**

### **Description**

Overly permissive CORS policy allows unauthorized domains to access resources.

### **Impact**

* Cross-origin attacks
* Data leakage
* Unauthorized API access
* Client-side data exposure

### **Steps to Reproduce**

1. Send cross-origin request:  
    fetch('http://testphp.vulnweb.com/api/data', {

mode: 'cors'

});

1. Observe permissive CORS headers

### **Remediation**

1. Restrict allowed origins
2. Validate CORS requests
3. Limit exposed headers
4. Configure proper CORS policy

## **11. [LOW] Missing CSRF Tokens**

**Severity: Low  
 Status: Solved  
 Risk Score: 3.2/10  
 CWE: CWE-352: Cross-Site Request Forgery  
 CVSS: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N**

### **Description**

Forms lack CSRF tokens, making them vulnerable to cross-site request forgery attacks.

### **Impact**

* Unauthorized actions
* State-changing requests
* Account manipulation
* Data modification

### **Steps to Reproduce**

1. Analyze form submission:  
    <form method="POST" action="/update\_profile">

<!-- No CSRF token present -->

</form>

1. Create CSRF PoC:  
    <form action="http://testphp.vulnweb.com/update\_profile" method="POST">

<input type="hidden" name="email" value="hacked@evil.com" />

<input type="submit" value="Click me!">

</form>

### **Remediation**

1. Implement CSRF tokens
2. Validate token on submission
3. Add SameSite cookie attribute
4. Check origin headers

## **12. [LOW] Sensitive Information in HTML Comments**

* **Severity: Low**
* **Status: Solved**
* **Risk Score: 2.8/10**
* **CWE: CWE-615: Information Exposure Through Comments**
* **CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N**

### **Description**

HTML source contains sensitive information in comments.

### **Impact**

* Information disclosure
* Technical details exposure
* Development information leak
* Potential security bypass hints

### **Steps to Reproduce**

1. View page source

Find sensitive comments:  
<!-- TODO: Remove admin backdoor -->

1. <!-- DB connection: dbuser/dbpass -->

### **Remediation**

1. Remove sensitive comments
2. Implement code review process
3. Use development environment for notes
4. Clean production code

## **13. [LOW] Outdated Components**

* **Severity: Low**
* **Status: Solved**
* **Risk Score: 2.5/10**
* **CWE: CWE-1104: Use of Unmaintained Third Party Components**
* **CVSS: CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L**

### **Description**

Several third-party components are outdated and contain known vulnerabilities.

### **Impact**

* Known vulnerability exposure
* Security patch missing
* Potential exploits
* Maintenance issues

### **Steps to Reproduce**

1. Check component versions:
   * jQuery 1.8.3
   * Bootstrap 3.3.7
   * PHP 5.6

### Remediation:

1. Update all components
2. Implement version monitoring
3. Regular security updates
4. Dependency scanning

## **14. [INFO] Server Information Disclosure**

* **Severity: Info**
* **Status: Solved**
* **Risk Score: 1.0/10**
* **CWE: CWE-200: Information Exposure**
* **CVSS: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N**

### **Description:**

Server headers and error messages reveal sensitive technical information**.**

### **Impact:**

* Technology stack disclosure
* Version information leakage
* System information exposure
* Potential attack vector information

### **Steps to Reproduce:**

1. Send HTTP request:  
   curl -I http://testphp.vulnweb.com
2. Observe server headers

### **Remediation:**

1. Remove server banners
2. Configure custom error pages
3. Disable detailed errors
4. Hide version information