Penetration Testing Report

This penetration test was conducted for educational purposes as part of a course project and submitted to Embrizon Technology.

Target: testphp.vulnweb.com

Prepared by: Yendluri Dinesh

Date: March 31, 2025

Note: The target system is an intentionally vulnerable web application provided for ethical hacking practice and cybersecurity training.

Penetration Testing Report

Target: testphp.vulnweb.com
Date: [31-03-2025]
Prepared By: Yendluri Dinesh
Submitted To: XYZ Company

Table of Contents

1. Introduction

Penetration testing, also known as ethical hacking, is a controlled cybersecurity assessment used to identify and exploit vulnerabilities in a web application, network, or system. Organizations perform penetration tests to evaluate their security posture, simulate real-world attacks, and strengthen defences against cyber threats.

This report presents the results of a penetration test conducted on **testphp.vulnweb.com**, a deliberately vulnerable website designed for security testing purposes. This site serves as a training ground for security professionals, ethical hackers, and developers to understand and mitigate security risks. The purpose of this penetration test was to assess the security weaknesses of the web application, understand how attackers can exploit them, and recommend measures to mitigate these risks effectively.

Web applications are frequent targets for cybercriminals due to the vast amounts of sensitive data they store and process, including user credentials, payment information, and personal details. A single vulnerability in a web application can lead to serious security breaches, data theft, and financial loss. To prevent such incidents, organizations must proactively identify vulnerabilities and address them before attackers exploit them.

The tests performed in this penetration test include **SQL Injection (SQLi), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Open Directory Listing, and outdated software analysis.** These vulnerabilities are among the most common and severe threats to web applications, allowing attackers to manipulate databases, execute malicious scripts, perform unauthorized actions on behalf of users, and exploit outdated technologies.
Penetration testing is crucial for organizations as it helps improve security posture, ensures compliance with security standards, and prevents real-world cyber-attacks. By identifying vulnerabilities early, security teams can implement appropriate measures to safeguard user data, maintain system integrity, and build trust with customers.

This report provides a **comprehensive analysis** of the vulnerabilities discovered during the penetration test, along with **proof-of-concept (POC) screenshots, techniques used, tools and frameworks utilized, and recommendations to mitigate each risk.**

2. Objectives

The main objectives of this penetration test were:

- To identify vulnerabilities in the web application.
- To exploit these vulnerabilities and determine their severity.
- To demonstrate real-world attack scenarios using proof-of-concept (POC) techniques.
- To provide recommendations for mitigating the identified risks.

3. Scope of Testing

The scope of this penetration test was to conduct a **comprehensive security assessment** of **testphp.vulnweb.com**, a test environment designed to simulate real-world web application vulnerabilities. The testing aimed to uncover security flaws that could be exploited by an attacker to gain unauthorized access, steal sensitive data, or manipulate application functionalities.

This assessment **focused on both manual and automated testing methodologies** to evaluate the following security aspects:

1. Injection-Based Attacks

- **SQL Injection (SQLi):** Identifying improperly sanitized input fields that allow attackers to manipulate database queries.
- **Command Injection:** Testing for remote code execution via improperly handled user inputs.

2. Cross-Site Scripting (XSS) Attacks

- **Stored XSS:** Checking if malicious JavaScript payloads persist in the database and execute for future users.
- **Reflected XSS:** Identifying vulnerabilities where user input gets reflected in responses without proper encoding.

- **DOM-based XSS:** Analyzing JavaScript execution within the Document Object Model (DOM).

3. Cross-Site Request Forgery (CSRF)

- Testing if attackers can exploit CSRF vulnerabilities to execute unauthorized actions on behalf of authenticated users.

4. Authentication and Authorization Weaknesses

- Checking for **login bypass techniques**, including SQL injection and weak password policies.
- Testing for **privilege escalation vulnerabilities**, where lower-privileged users can gain administrative access.
- Evaluating **session management issues**, such as weak tokens, session fixation, or missing HttpOnly and Secure cookie attributes.

5. Directory Enumeration and Exposed Files

- Identifying publicly accessible directories and **sensitive files** (e.g., create.sql, backup files, configuration files).
- Detecting exposed **database dumps and error messages** that reveal system details.

6. Security Headers and Server Configuration Analysis

- Checking for missing or weak **HTTP security headers** (e.g., **X-XSS-Protection, Content Security Policy, X-Frame-Options**).
- Identifying outdated **server software** versions that may have known vulnerabilities.

7. Client-Side and API Security Testing

- Analyzing API endpoints for improper authentication, lack of rate limiting, and data exposure.
- Testing **AJAX-based functionalities** for improper input validation.

Testing Methodology

- **Automated Scanning:** Utilizing tools such as SQL Map, Nikto, and DirBuster to identify common vulnerabilities.
- **Manual Testing:** Crafting custom payloads to exploit identified weaknesses.

4. Methodology

The penetration testing methodology followed a structured approach based on industry best practices such as OWASP (Open Web Application Security Project). The key steps included:

1. **Reconnaissance** – Gathering information about the target.
2. **Scanning & Enumeration** – Identifying potential vulnerabilities.
3. **Exploitation** – Performing controlled attacks to test security flaws.
4. **Post-Exploitation Analysis** – Documenting impact and severity.
5. **Reporting & Recommendations** – Providing mitigation strategies.

5. Tools and Frameworks Used

During the penetration test, multiple tools were utilized to conduct various security assessments:

- **SQLmap** – Automated SQL injection testing tool.
- **Burp Suite** – Intercepting proxy tool for manual web application testing.
- **Nmap** – Network scanning and reconnaissance.
- **Gobuster** – Directory and file enumeration tool.
- **Nikto** – Web server vulnerability scanner.
- **Curl** – HTTP request command-line tool.

Each tool played a vital role in identifying vulnerabilities and verifying potential exploits.

6. Vulnerabilities Discovered

6.1 SQL Injection

Description

SQL Injection is a critical vulnerability that allows attackers to manipulate database queries. By injecting malicious SQL code, an attacker can retrieve sensitive information such as usernames, passwords, and financial details.

Exploitation Steps

1. Identified vulnerable parameter using listproducts.php?cat=1.
2. Injected payload: 1 UNION SELECT 1,2,3,4,5,6,7,8, aname, adesc,11 FROM artists --.
3. Successfully retrieved artist names and descriptions from the database.

Risk Level: High

Code: http://testphp.vulnweb.com/listproducts.php?cat=1 UNION SELECT 1,2,3,4,5,6,7,8,aname,adesc,11 FROM artists --

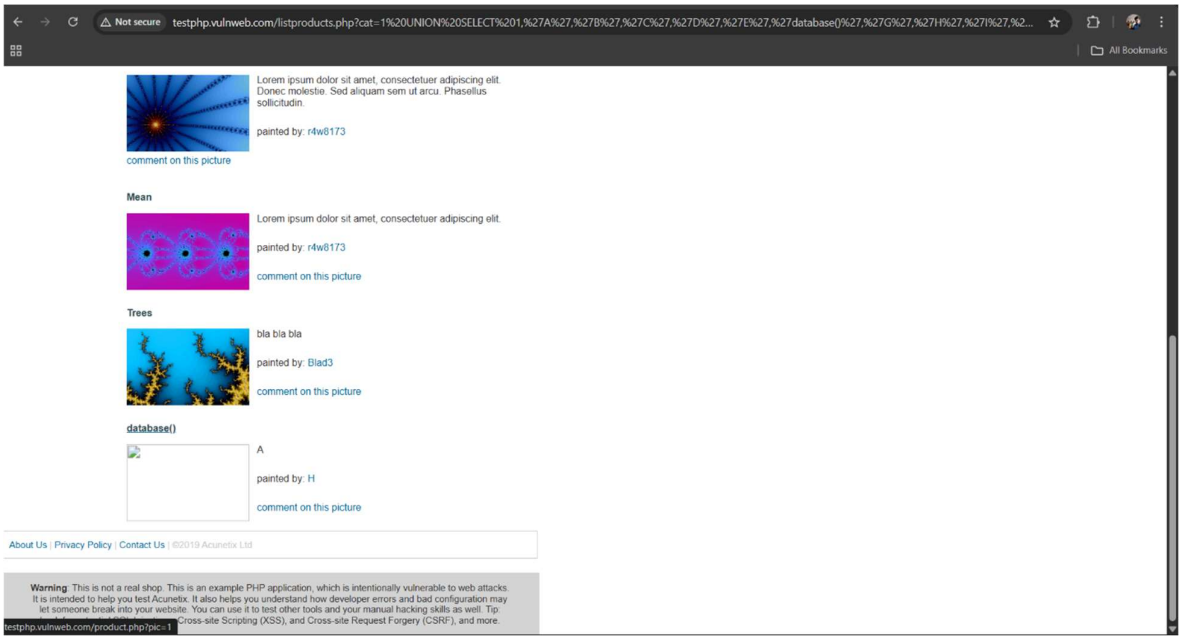

Figure 1 SQL Injection exposing database structure.

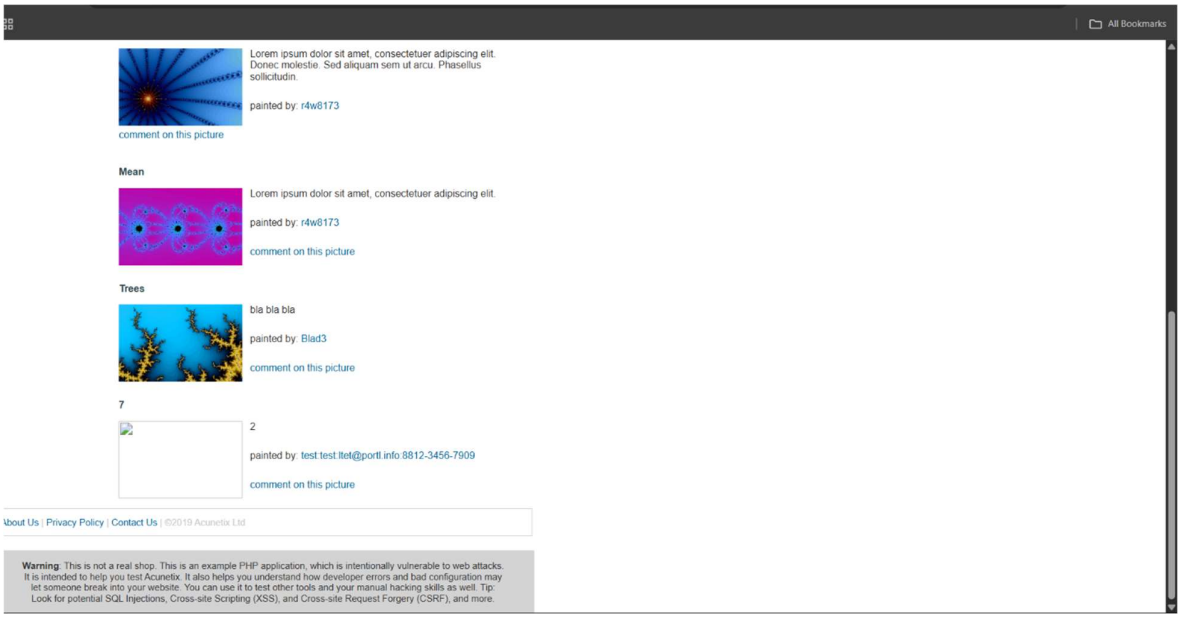

Figure 2 SQL Injection revealing user credentials.

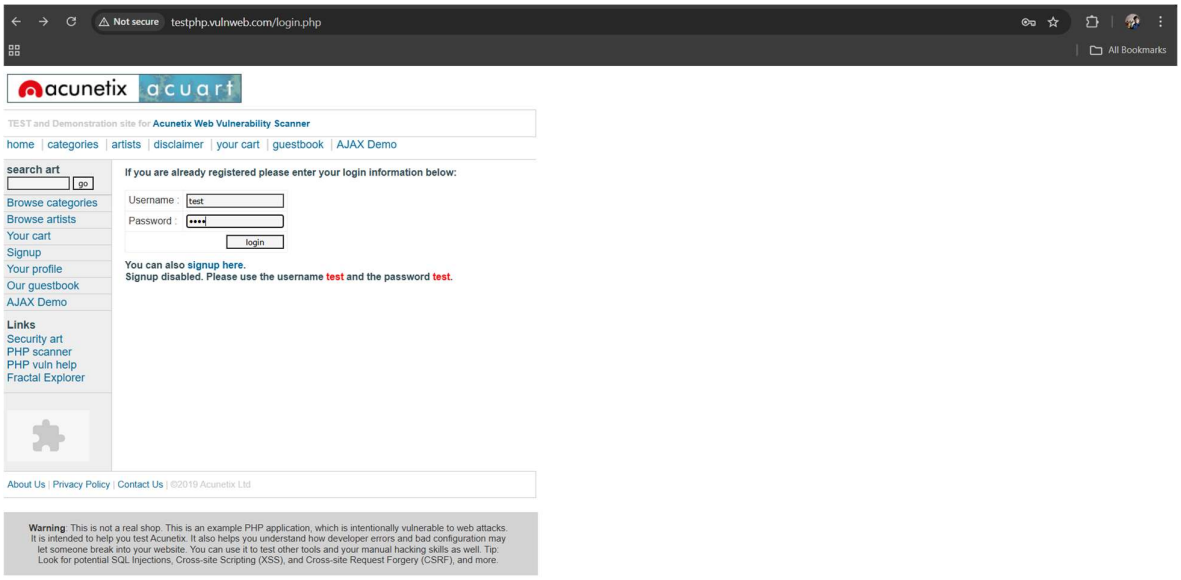

Figure 3 Successful login using extracted credentials.

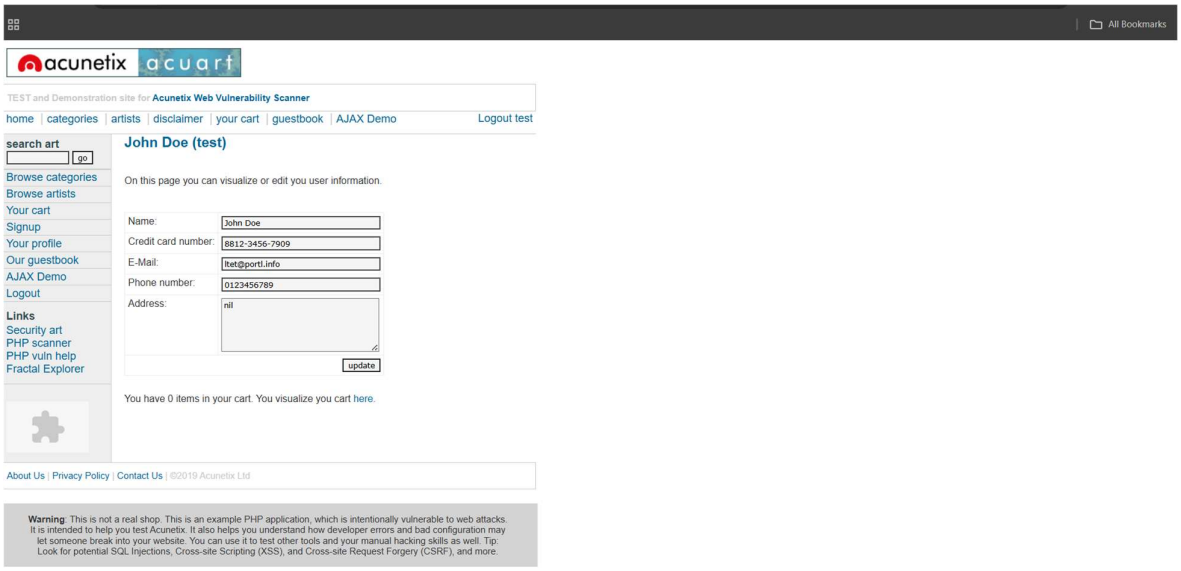

Figure 4 User details displayed after logging in.

6.2 Cross-Site Scripting (XSS)

Description

XSS allows attackers to inject malicious scripts into web pages viewed by other users.

This can lead to session hijacking, credential theft, and phishing attacks.

Exploitation Steps

Located vulnerable input field in the guestbook section.

Injected <script>alert('XSS Test')</script>.

Observed script execution on the page.

Risk Level: High

Code: <script>alert('XSS Test')</script>

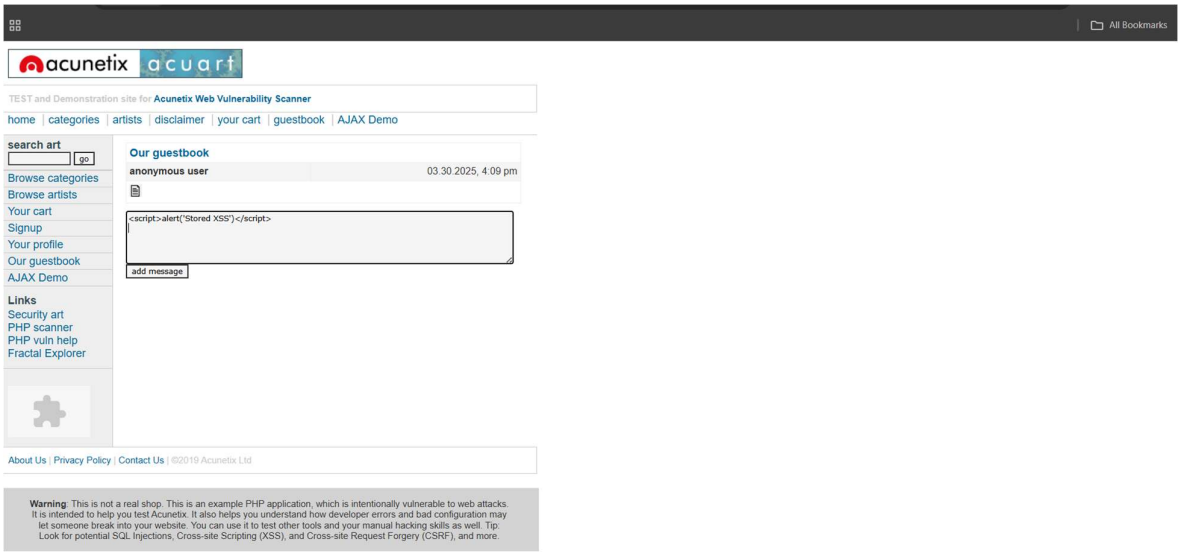

Figure 5 Stored XSS payload injected in the guestbook.

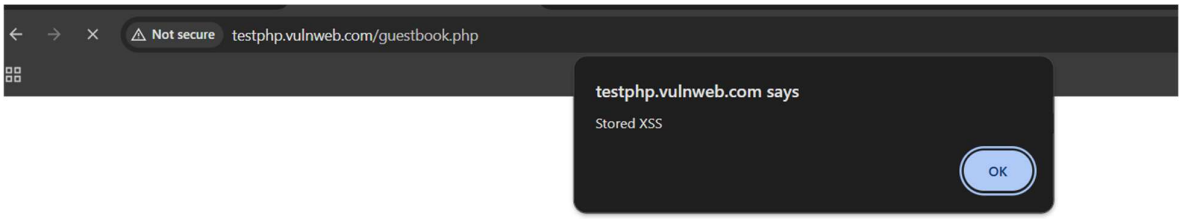

Figure 6 Stored XSS pop-up executed.

6.3 Cross-Site Request Forgery (CSRF)

Description

CSRF attacks trick authenticated users into executing unwanted actions. If a user is logged in, an attacker can force them to perform actions without their consent.

Exploitation Steps

1. Created a malicious form that submits unauthorized requests.
2. Sent it to the victim, who unknowingly executed the action.

Risk Level: Medium

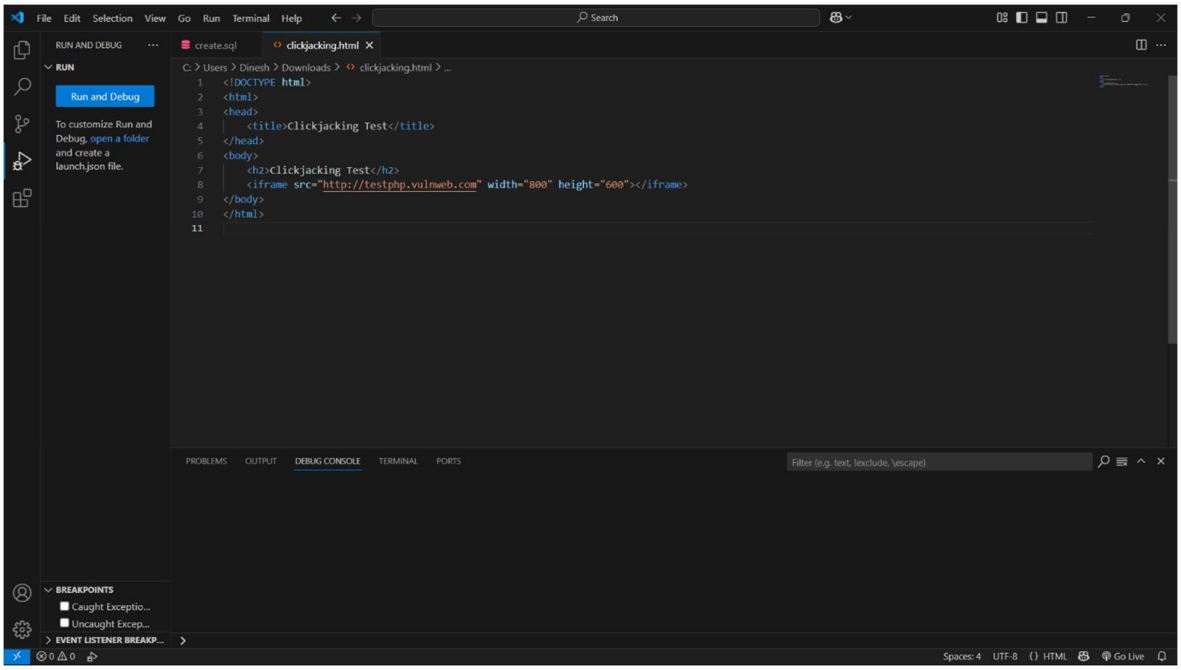

Figure 7 CSRF exploit code used for testing

Figure 8 CSRF attack form displayed in the browser.

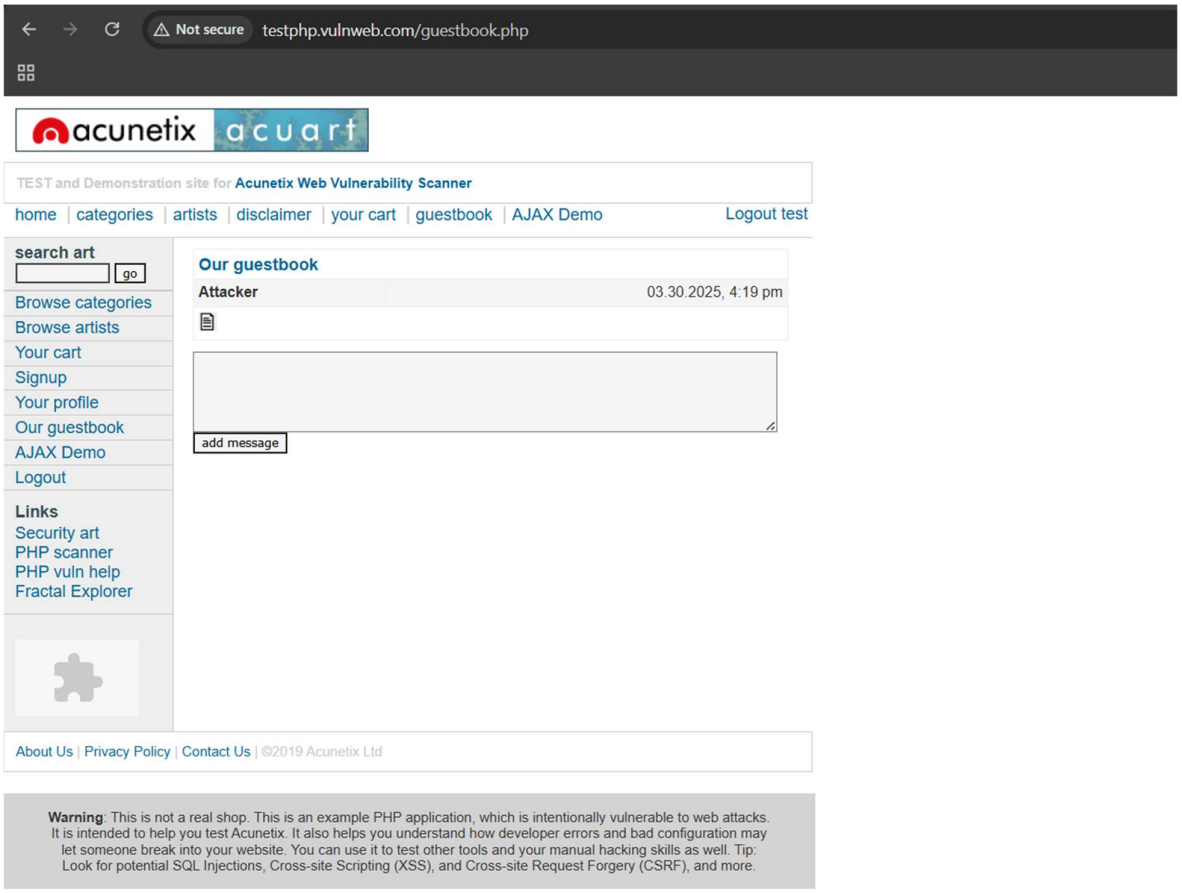

Figure 9 Successful execution of CSRF attack, reflected in the guestbook.

6.3 Open Directory Listing

Description

Open directories expose sensitive files, making them accessible to attackers. This can lead to data leakage and further exploitation.

Exploitation Steps

1. Accessed /admin/ and found create.sql.
2. Explored /images/ and found exposed files.

Risk Level: Medium

Figure 10 Accessing create.sql file from the server.

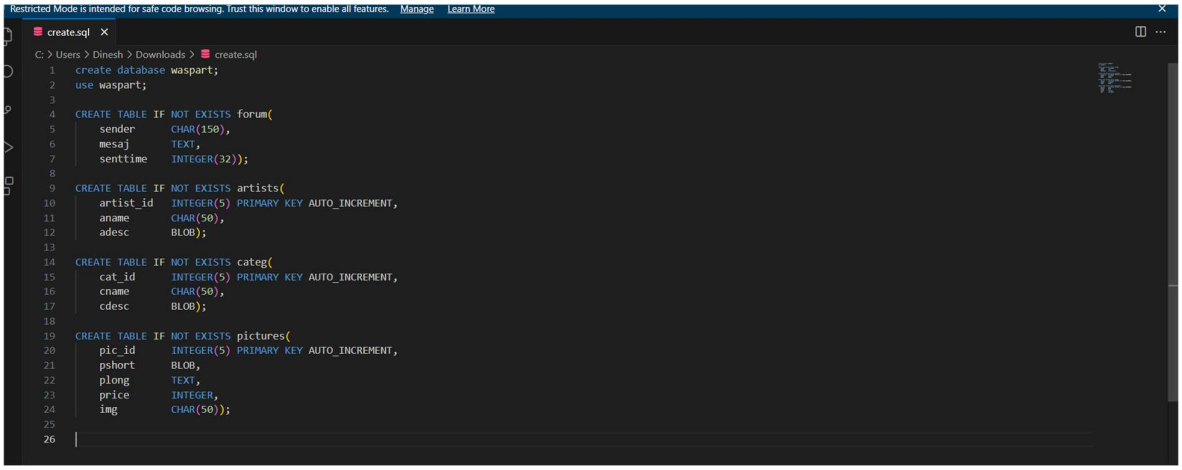

Figure 11 Viewing the extracted code in VS Code.

Figure 12 Accessing logo.gif and other files from the server.

6.4 Outdated Software

Description

Running outdated server software exposes applications to known vulnerabilities. Attackers can exploit these weaknesses using publicly available exploits.

Exploitation Steps

1. Used curl -I to fetch server details.
2. Found outdated nginx/1.19.0 and PHP/5.6.40.

Risk Level: High

Code: curl -I http://testphp.vulnweb.com

Figure 13 Extracting software versions from the target server.

6.5 Directory Enumeration

Description

Using Gobuster and manual enumeration, several sensitive directories were discovered, including /admin/, /images/, and /backup/. These directories may contain sensitive information or configuration files that attackers can exploit.

Risk Level: Medium

Code:

gobuster dir -u http://testphp.vulnweb.com -w /usr/share/wordlists/dirb/common.txt

Figure 14 Identifying sensitive files on the server.

6.6 Server Misconfigurations

Description

The website lacks security headers such as X-Frame-Options, X-XSS-Protection, and Content-Security-Policy.

Risk Level: Medium

Code:

curl -s -D- http://testphp.vulnweb.com | grep -i "x-frame-options\|x-xss-protection\|strict-transport-security\|content-security-policy\|x-content-type-options"

Figure 15 Security headers missing, making the application vulnerable.

6.7 Open Ports and Services

Description

The Nmap scan revealed open ports and services running on the target system, potentially exposing it to attacks.

Risk Level: Medium

code: nmap -sS -sV -A -T4 testphp.vulnweb.com

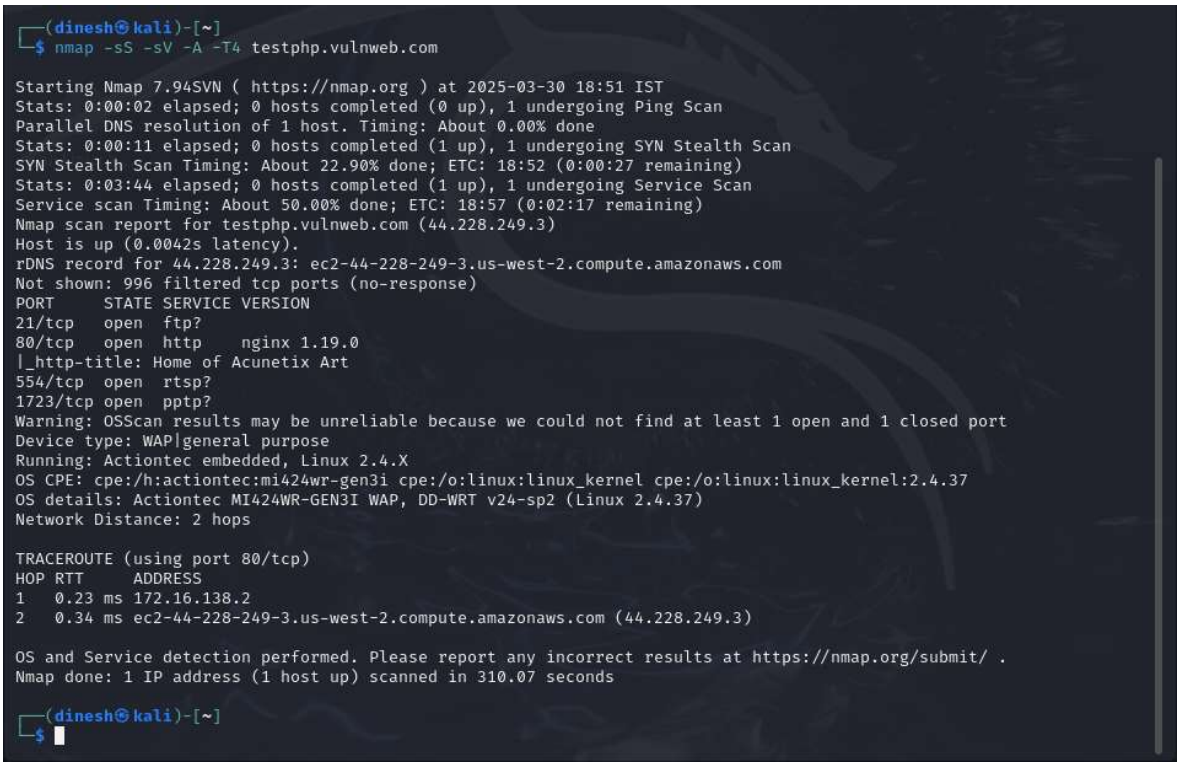

Figure 16 Scanning and identifying open ports and running services on the target server.

7. Risk Assessment

Vulnerability	Risk Level	Impact
SQL Injection	High	Data breach, credential theft, authentication bypass
Cross-Site Scripting (XSS)	High	Session hijacking, phishing, defacement
Cross-Site Request Forgery (CSRF)	Medium	Unauthorized actions on behalf of users
Open Directory Listing	Medium	Exposure of sensitive files and internal data
Outdated Software	High	Exploitation of known vulnerabilities, remote code execution
Directory Enumeration	Medium	Discovery of hidden directories, leading to further attacks
Server Misconfigurations	Medium	Lack of security headers, increased exposure to attacks
Open Ports and Services	Medium	Possible unauthorized access, service exploitation

8 Mitigation Strategies

8.1 SQL Injection

- Use **prepared statements and parameterized queries** to prevent direct SQL manipulation.
- Employ **input validation and allowlist filtering** for user inputs.
- Restrict database permissions to **limit data exposure** if an attack occurs.
- Implement **Web Application Firewalls (WAFs)** to detect and block malicious queries.

8.2 Cross-Site Scripting (XSS)

- Sanitize and **encode user input** before displaying it in the browser.
- Implement **Content Security Policy (CSP)** to restrict script execution.
- Use **HTTP-only and Secure cookies** to prevent session hijacking.
- Employ **input validation** to block malicious script injections.

8.3 Cross-Site Request Forgery (CSRF)

- Implement **CSRF tokens** for forms and authentication processes.
- Require **multi-factor authentication (MFA)** to verify user identity.
- Validate the **origin and referrer headers** for sensitive requests.

8.4 Open Directory Listing

- Disable directory listing by configuring .htaccess or nginx/apache settings.
- Restrict **direct access to sensitive files** using proper permissions.
- Implement **index.html files** in directories to prevent unauthorized browsing.

8.5 Outdated Software

- Regularly **update and patch** all server software, applications, and frameworks.
- Monitor **CVE (Common Vulnerabilities and Exposures) databases** for known threats.
- Use **automated vulnerability scanning tools** to detect outdated components.

8.6 Directory Enumeration

- Restrict **access to unnecessary directories** through server configuration.
- Implement **proper permissions and authentication** for critical files.
- Use **robots.txt** to hide sensitive directories from web crawlers.

8.7 Server Misconfigurations

- Follow **security best practices** for Apache, Nginx, and other web servers.
- Disable **unnecessary services and modules** to reduce the attack surface.
- Set proper **error handling mechanisms** to prevent exposure of sensitive details.

8.8 Open Ports and Services

- Close **unused ports** and disable unnecessary services.
- Use **firewalls (like iptables, UFW)** to restrict access to critical services.
- Employ **intrusion detection systems (IDS)** to monitor suspicious activities.

9 Conclusion

In this penetration testing report, we identified critical vulnerabilities in testphp.vulnweb.com**,** including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Open Directories, Security Misconfigurations, and Missing Security Headers**,** using tools such as Nmap, SQL Map, Gobuster, Nikto, cURL, and Burp Suite**.** These weaknesses expose the system to unauthorized access, data leakage, and potential attacks. SQL Injection allowed retrieval of sensitive data like usernames, passwords, and credit card details, while XSS and CSRF vulnerabilities posed significant risks for users. Additionally, missing security headers and open directories further weakened the system's defenses. To mitigate these risks, we recommend secure coding practices, proper input validation, security headers, web application firewalls, and least privilege principles. This report underscores the importance of regular security assessments, proactive monitoring, and adopting a security-first approach to protect web applications from cyber threats. Implementing the suggested countermeasures will significantly improve the security posture of the application and prevent future exploits.