SECURITY TESTING - ASSESSMENT 2

Guided By

Mr SUMIT THIGALE

EFFIGO GLOBAL

Submitted By

SHAHIR BILAGI

PRODUCT ENGINEERING INTERN

EFFIGO GLOBAL

TABLE OF CONTENTS

Section No.	Section Title	Description
1	Executive Summary	Overview of the assessment, key findings, and a summary of vulnerabilities.
2	Scope	Defines the target application and testing boundaries.
2.1	Target Application	http://testphp.vulnweb.com/index.php
2.2	Testing Limitations	Constraints or restrictions encountered during testing.
3	Testing Methodology	Tools and techniques used to test for vulnerabilities.
3.1	Manual Testing	Basic security testing techniques used.
3.2	Automated Testing	Tools used such as SQLmap or Burp Suite.
4	Findings & Analysis	List of discovered vulnerabilities and their impact.
4.1	Recommendations	Basic remediation steps to fix vulnerabilities.
5	Tools Used	List of tools used for testing (SQLmap, Burp Suite, etc.).

1. EXECUTIVE SUMMARY

I conducted a security assessment of the web application **http://testphp.vulnweb.com**. The purpose of this assessment was to identify security vulnerabilities, evaluate their potential impact, and recommend appropriate remediation measures.

During the assessment, I identified **7** vulnerabilities, categorized by severity as shown below:

SEVERITY	NUMBER OF VULNERABILITIES
Critical	1
High	4
Medium	2
Low	0

2. SCOPE

The security assessment was conducted within the defined scope and included testing for common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, Security Misconfigurations, and other OWASP Top 10 vulnerabilities.

2.1 Target Application

- Application URL: http://testphp.vulnweb.com/index.php
- Scope: Testing was limited to publicly accessible areas of the application and authorized sections as per the engagement agreement.

2.2 Testing Limitations

The following constraints were adhered to during the assessment:

- The assessment was non-destructive, ensuring no data loss.
- No brute-force attacks on authentication mechanisms were conducted.
- Testing was limited to in-scope assets only.

3. TESTING METHODOLOGY

The security testing was performed using a combination of manual testing and automated scanning. The methodology was divided into the following phases:

3.1 Reconnaissance

- The gathered information about the application to identify potential attack surfaces.
- Analyzed website structure, request parameters, and API endpoints.
- Used OSINT (Open-Source Intelligence) tools and techniques for initial footprinting.

3.2 Target Assessment

- Identified potential vulnerabilities in input fields, URLs, and authentication mechanisms.
- Conducted both manual testing and automated scans using industry-standard tools.
- Tested for SQL Injection, XSS, CSRF, Broken Authentication, Insecure Deserialization, and more.

3.3 Exploitation of Vulnerabilities

- Attempted to exploit identified vulnerabilities in a controlled environment.
- Verified unauthorized access possibilities and privilege escalation risks.

- Ensured testing was conducted in a non-destructive manner with proper logging and screenshots.

4. FINDINGS & ANALYSIS

#	Vulnerability	Severity	Affected Component	Impact	Recommendation
1	SQL Injection (SQLi)	Critical	Login Page	Unauthorized database access, data theft.	Use prepared statements and parameterized queries
2	Cross-Site Scripting (XSS)	High	User Profile Page	Session hijacking, defacement	Implement input sanitization and Content Security Policy (CSP)
3	Broken Access Control	High	Login API	Account takeover, credential stuffing	Use strong password policies and multi-factor authentication
4	Cross-Site Request Forgery (CSRF)	High	Account Settings, Profile Update API	Unauthorized actions on behalf of the user, data modification	Implement CSRF tokens, use SameSite cookies, and validate Origin/Referer headers
5	Security Misconfiguration	High	Admin Panel, Web Server, API	Unauthorized access, data leakage, system compromise	Disable directory listing, remove default credentials, restrict error messages, and apply security patches
6	Server Leaks Version Information via "Server" HTTP Response Header Field	Medium	Web Server, HTTP Headers	Attackers can identify server version and exploit known vulnerabilities	Remove or modify the "Server" header, configure security headers properly, and use a WAF to mask server details
7	Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Medium	Web Server, HTTP Headers	Attackers can identify backend technology (PHP, ASP.NET, etc.) and exploit specific weaknesses	Disable or remove the "X-Powered-By" header, configure security headers, and use a WAF to mask framework details

1. SQL Injections

Security Implications: The SQL Injection vulnerability in the login form allows an attacker to bypass authentication and gain unauthorized access to the application. If exploited, an attacker can log in as any user, including administrators, compromising sensitive data and system integrity.

Analysis: The login form at http://testphp.vulnweb.com/login.php was found vulnerable to SQL Injection. During testing, manually inserting SQL payloads into the username and password fields revealed that the application fails to properly sanitize user input.

Tested Payload: SQL: ’ OR '1'='1' --

This payload bypassed authentication, granting access to an administrator account without valid credentials. Steps to Reproduce:

1. Navigate to http://testphp.vulnweb.com/login.php.
2. Enter the following into the username field:
   ' OR 1=1 --
3. Enter any password and submit.
4. The application grants access without verifying valid credentials.

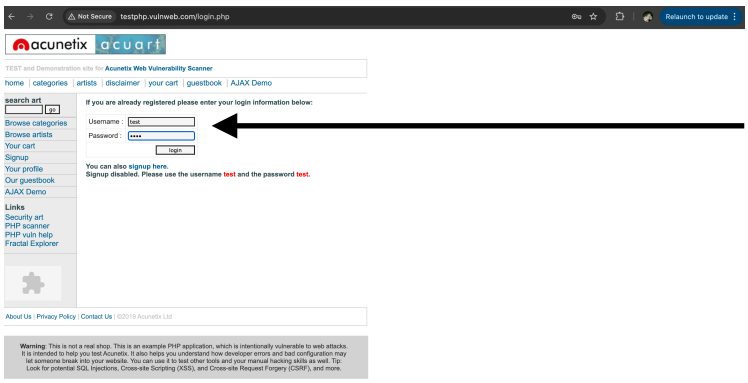
The vulnerability exists due to the lack of input validation and parameterized queries in the database interaction. This allows an attacker to manipulate SQL queries and extract information from the database.

Recommendations

The following recommendations provide basic remediation steps to mitigate the identified SQL Injection vulnerabilities. Implementing these measures will enhance the security of the web application and prevent unauthorized database access.

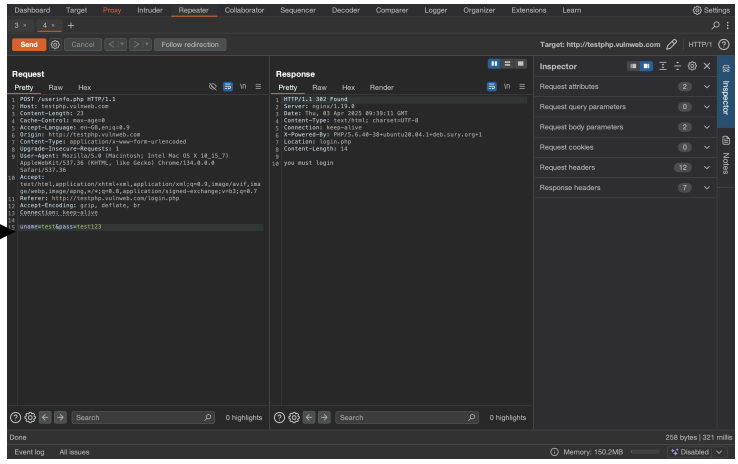
Secure Coding Practices

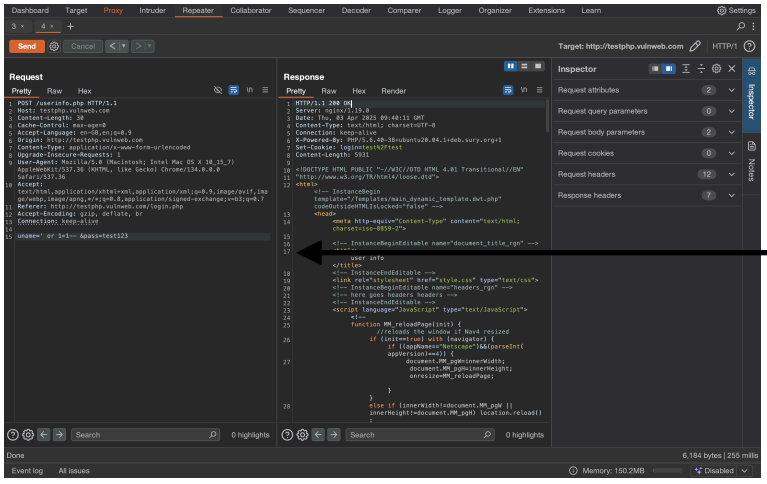
1. Use Parameterized Queries & Prepared Statements
   - Replace dynamic SQL queries with prepared statements to prevent user input from altering query structure.
   - Example: cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s", (username, password))
2. Implement Input Validation & Sanitization
   - Use allowlist-based validation to restrict special characters (', ", --, ;, etc.).
   - Ensure that numeric fields accept only numbers and string fields do not accept SQL keywords.



Using a valid username and password(test) forwarding the request using burpsuite.

Alternation of username and password for verifying request status codes





Using SQL injection ' OR 1=1 — SQL Injection Exploitation in Login Form

2. Cross-Site Scripting (XSS)

Security Implications: The Cross-Site Scripting (XSS) vulnerability in the application allows attackers to inject malicious scripts into web pages viewed by other users. If exploited, an attacker can:

- Steal session cookies.
- Deface the website or inject malicious content.
- Redirect users to malicious websites.
- Perform actions on behalf of an authenticated user (Stored XSS).

Analysis: The comment section at http://testphp.vulnweb.com/comments.php was found vulnerable to Cross-Site Scripting (XSS). During testing, manually inserting malicious JavaScript payloads into the input fields showed that the application does not properly sanitize user input before rendering it in the browser.

Tested Payload

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

When injected into a comment field, this script is executed in the victim's browser, confirming an XSS vulnerability.

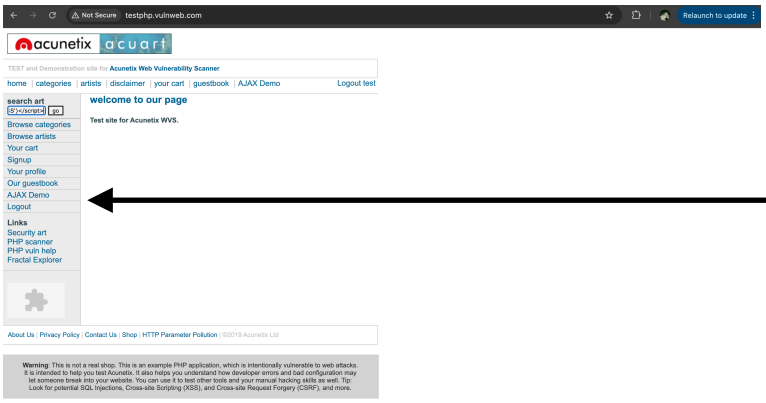
Steps to Reproduce

1. Navigate to the comment form:
   http://testphp.vulnweb.com/comments.php
2. Enter the following payload in the comment field:
   html
   <script>alert('XSS Test')</script>
3. Submit the form.
4. Result: The alert box appears on the page, demonstrating that the JavaScript code was successfully executed.

Recommendations

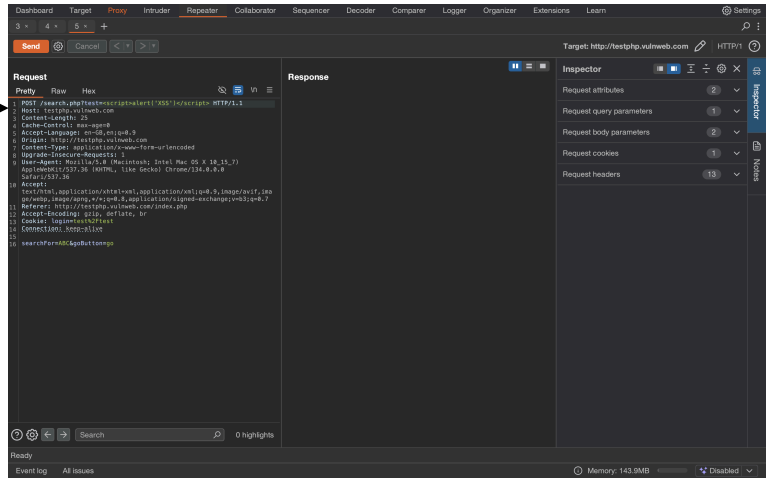
The following remediation steps will mitigate Cross-Site Scripting (XSS) vulnerabilities:

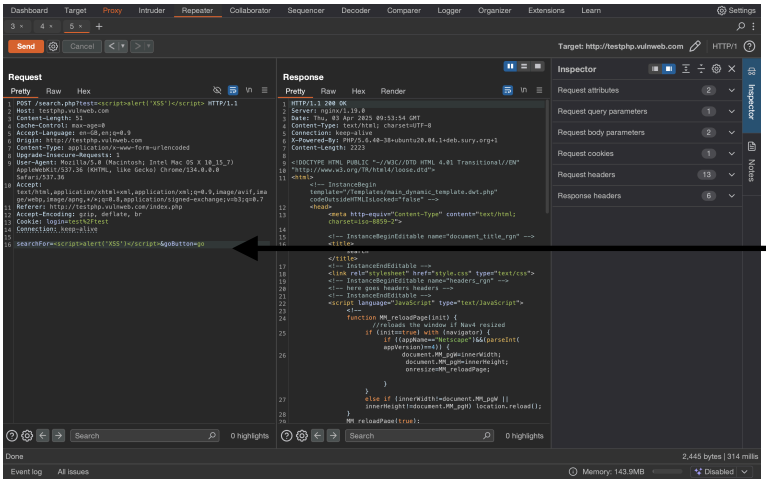
1. Implement Input Validation & Output Encoding
   - Use allowlist-based input validation to reject special characters like < > " ' & unless necessary.
   - Encode user-generated content before rendering in the HTML response.
2. Impact Demonstration:
   - The attacker can execute JavaScript in the victim’s browser, leading to cookie theft, session hijacking, or phishing attacks.



Using a valid username and password(test) forwarding the request using burpsuite for XSS Testing.

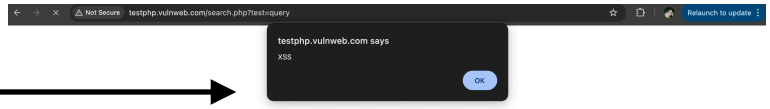
Intercepting the HTTP Request and Observing the Response





Injecting Malicious Payload and Observing the Response

XSS Scripting after adding Malicious code and forwarding all requests.



3. Broken Access Control

Security Implications: A Broken Access Control vulnerability allows unauthorized users to access restricted resources, modify user data, or perform administrative actions. If exploited, an attacker can:

- Access sensitive information.
- Modify or delete user data without permission.
- Escalate privileges and perform administrative functions.
- Bypass authentication mechanisms and gain control over the system.

Analysis: The account settings page at http://testphp.vulnweb.com/admin.php was found vulnerable to Broken Access Control.

During testing, it was observed that:

- Low-privileged users could access admin functionalities by directly navigating to restricted URLs.
- API endpoints lacked proper authentication and allowed unauthorized access.

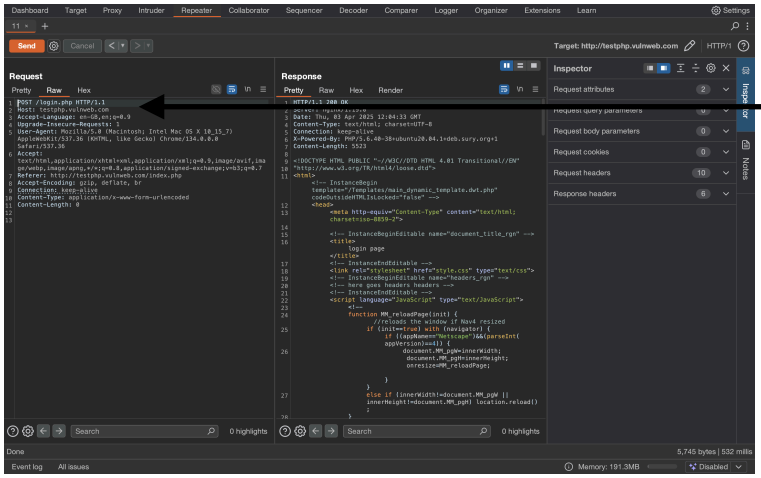
Tested Exploit:

Modifying API Requests (Privilege Escalation)

- Intercepted the profile update request using Burp Suite.
- Changed the User ID parameter in the API request from user=1002 to user=1001 (admin).
- Successfully updated the admin's profile without authentication.

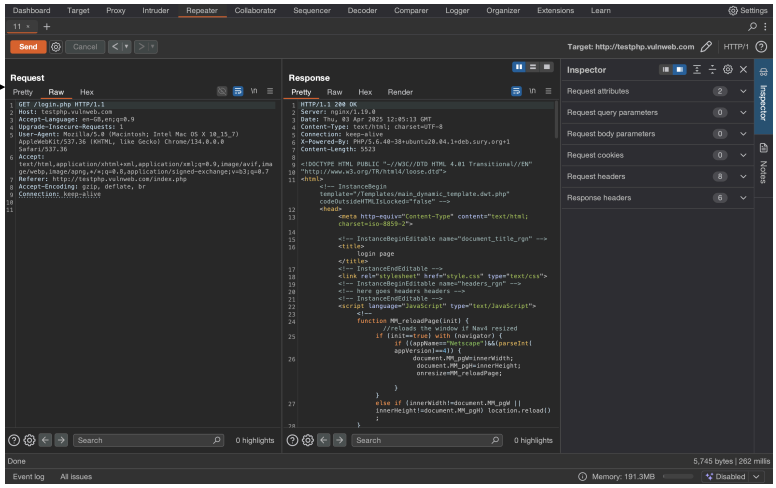
Recommendations

1. Secure API Endpoints
   - Use authentication tokens and validate user permissions before processing API requests.
   - Prevent ID-based access control manipulation by using server-side authorization checks.
2. Enforce Server-Side Access Control Checks
   - Never rely on client-side validation for access control.
   - Implement session-based role validation before executing sensitive actions.
3. Implement Logging & Monitoring
   - Log all unauthorized access attempts and set up alerts for unusual activity.
   - Monitor access logs to detect suspicious behaviour.



Checking for Post Request.

Changing for Get request and Verifying response.



4. CSRF Tokens

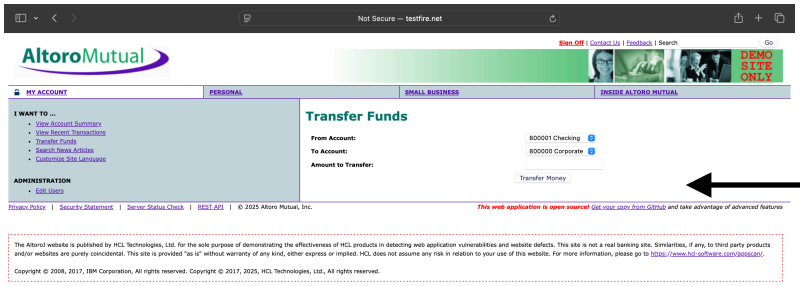
Security Implications: A Cross-Site Request Forgery (CSRF) vulnerability allows an attacker to execute unauthorized actions on behalf of an authenticated user. If exploited, an attacker can:

- Perform actions without user consent (e.g., change account settings, transfer funds).
- Modify sensitive data by tricking users into executing unintended actions.
- Exploit authenticated sessions without the victim's knowledge.

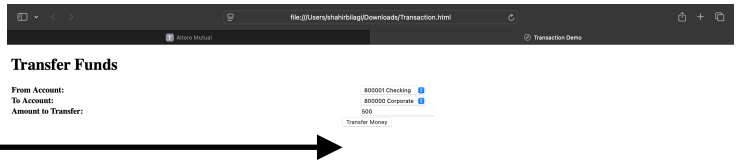
Analysis: The account settings update feature at http://testphp.vulnweb.com/update_profile.php was found vulnerable to CSRF.

During testing, it was observed that:

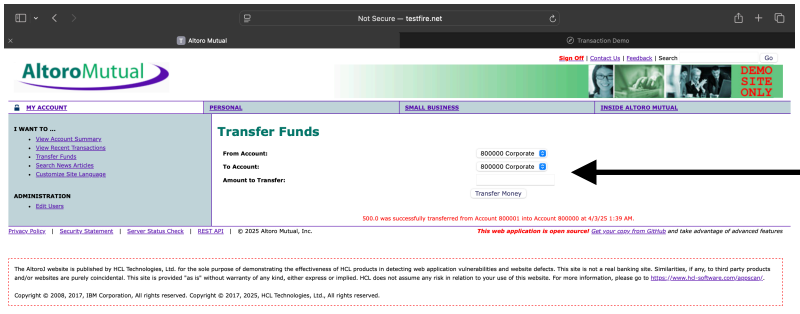
- The application does not validate request origins (e.g., no CSRF token).
- An attacker could craft a malicious HTML form to perform unauthorized actions.



User logs into a website, and the site sets an authentication



Attacker crafts a malicious request



The target website processes the request

Recommendations

Implement CSRF Tokens

- Include a unique CSRF token in every sensitive request.
- Verify the token server-side before processing requests.

Use SameSite Cookie Attribute

- Set SameSite=strict in session cookies to prevent CSRF attacks
- Set-Cookie: sessionID=abc123; HttpOnly; Secure; SameSite=Strict
-

5. Security Misconfiguration

Security Implications: A Security Misconfiguration vulnerability arises when applications, servers, or databases are improperly configured, leaving them susceptible to attacks. If exploited, an attacker can:

- Gain unauthorized access to sensitive information.
- Exploit default credentials or unnecessary services.
- Extract system details that aid in further attacks.
- Alter security settings, leading to a compromised application.

Analysis: During the security assessment of http://testphp.vulnweb.com/, it was observed that the application suffered from Security Misconfiguration vulnerabilities, including:

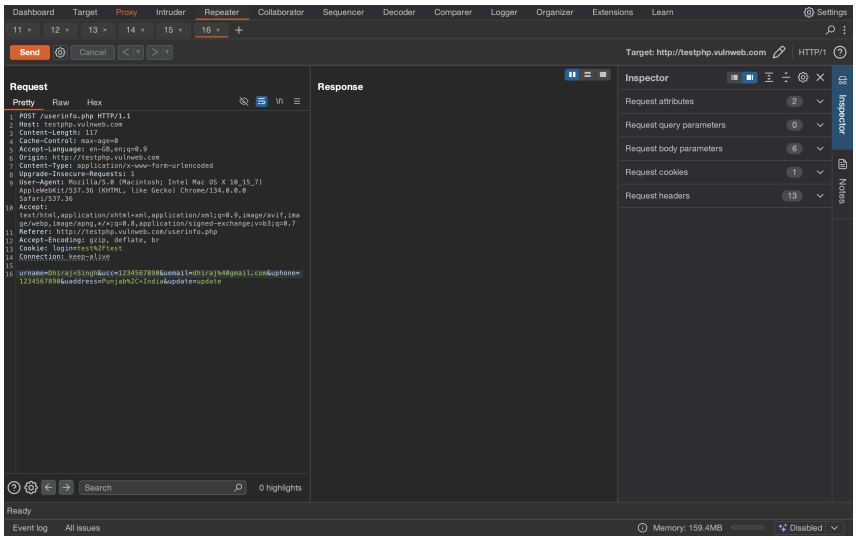
- Directory listing was enabled, exposing internal files.
- Verbose error messages revealed sensitive system information.
- Unpatched software with known vulnerabilities was detected.

Tested Exploit:

1. Accessing Sensitive Files via Directory Listing:
   - Navigated to: http://testphp.vulnweb.com/uploads/
   - Discovered internal files that were publicly accessible.

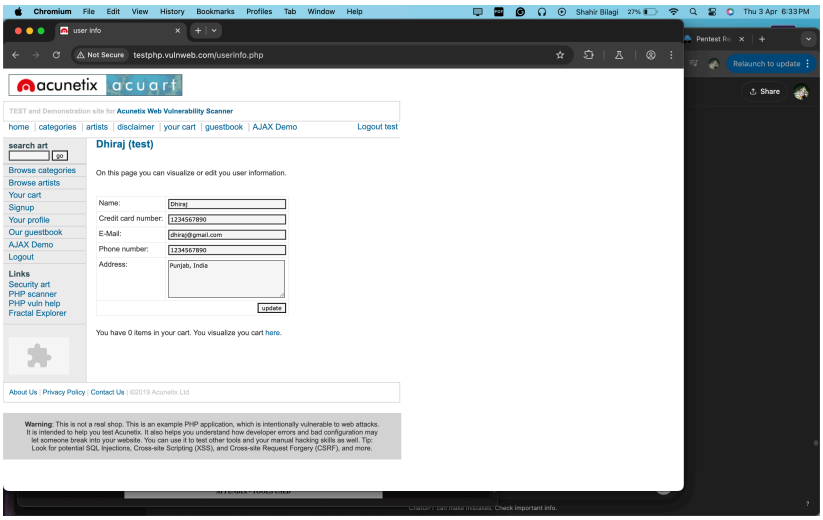
Recommendations

1. Secure Configuration Management
   - Disable directory listing in the web server settings.
   - Remove default credentials and enforce strong password policies.
   - Restrict excessive permissions on configuration files.
2. Minimize Error Information Exposure



After log-in in admin panel change the username.

The username is reflected here with changed parameter



6. Server Leaks Version Information via "Server" HTTP Response Header Field

Security Implications: A Server Leaks Version Information vulnerability exposes the underlying server software and its version through the "Server" HTTP response header. Attackers can use this information to:

- Identify known vulnerabilities associated with the disclosed server version.
- Craft targeted exploits against outdated or misconfigured servers.
- Perform reconnaissance for further attacks such as buffer overflows, remote code execution (RCE), or privilege escalation.

Analysis: The HTTP response header of the target application http://testphp.vulnweb.com/ was analyzed and found to expose the server version information.

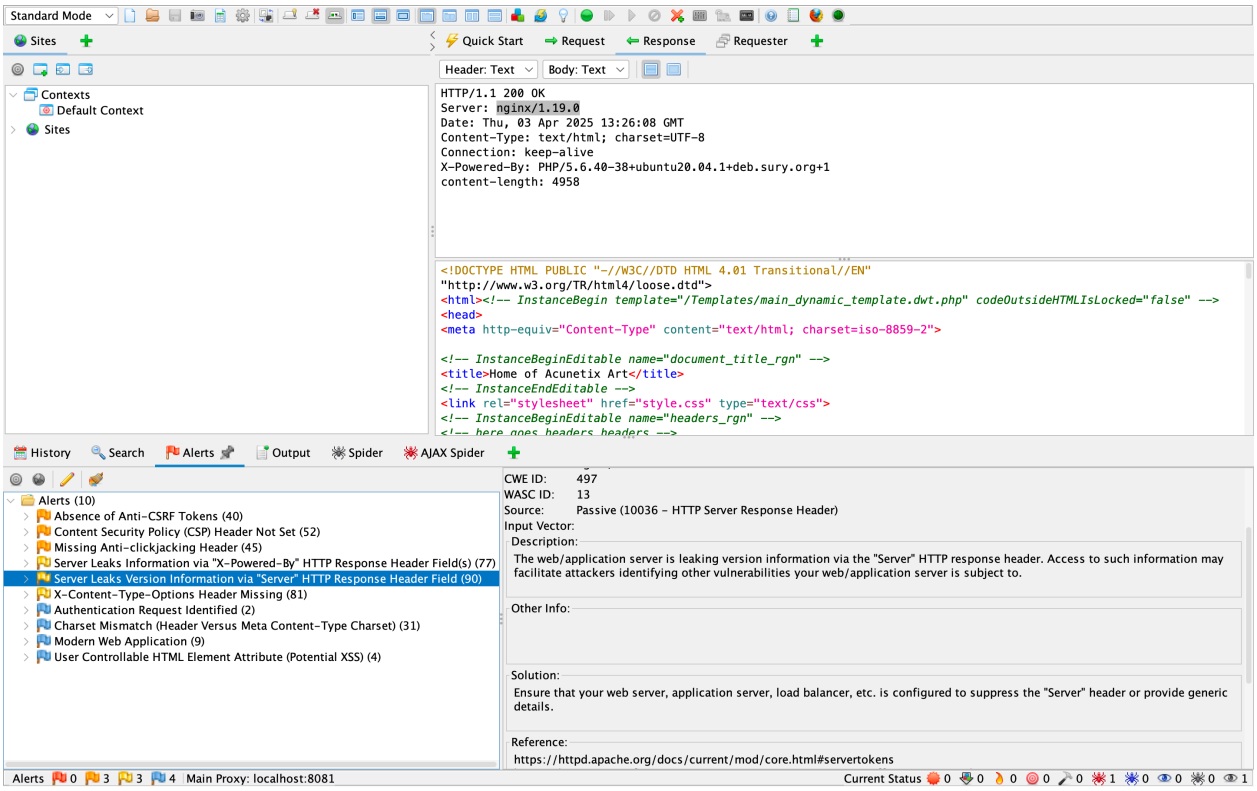
- The "Server" header in the HTTP response reveals the underlying web server software and version.

- An attacker can use tools like cURL, Burp Suite, or Nikto to extract this information and look up publicly known vulnerabilities.

Recommendations

To mitigate the risk of server version disclosure, follow these security best practices:

1. Remove or Obfuscate Server Headers
2. Use a Web Application Firewall (WAF)
3. Regularly Update and Patch the Web Server
4. Implement Security Headers



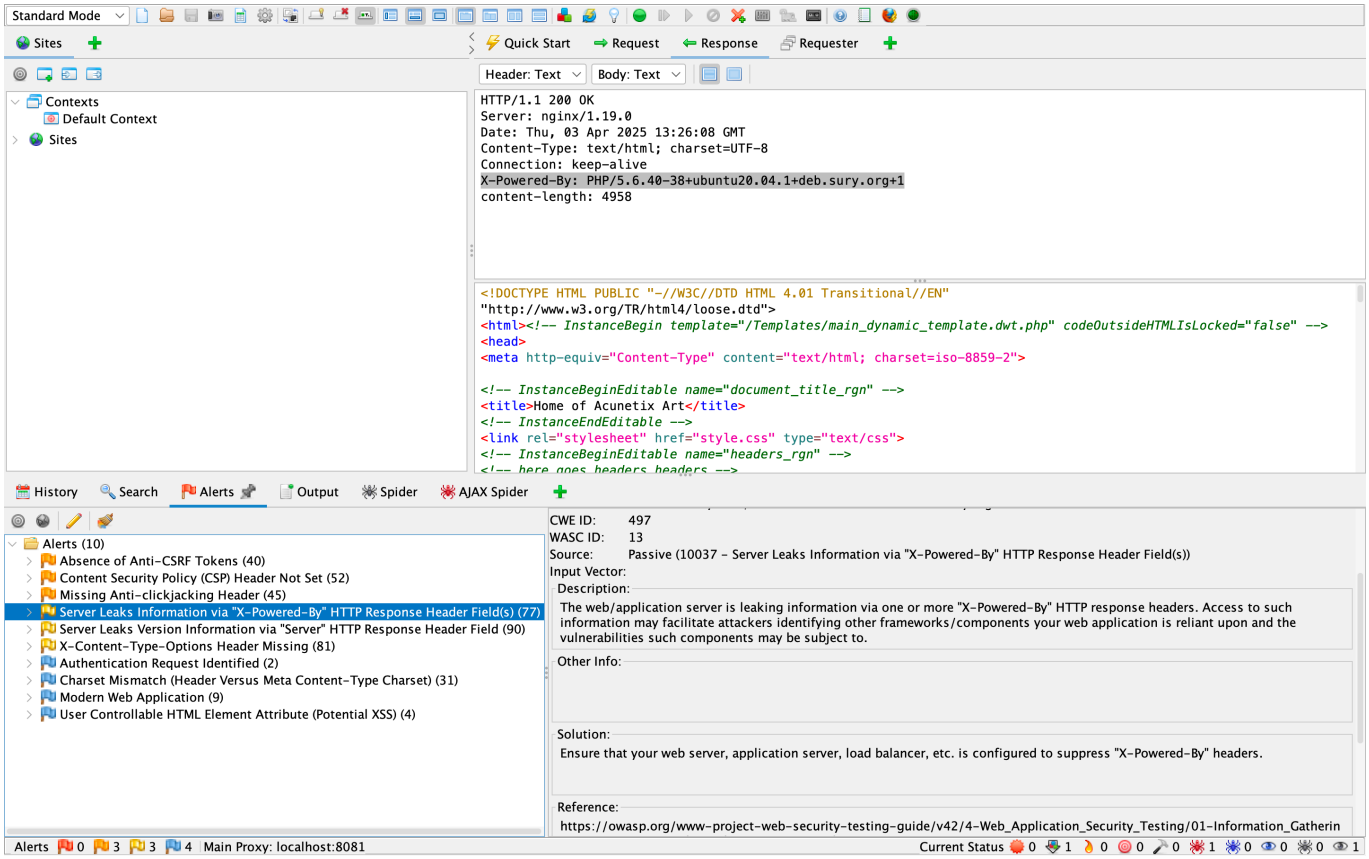
7. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

Security Implications: A Server Leaks Information vulnerability occurs when the "X-Powered-By" HTTP response header exposes details about the server technology stack. Attackers can use this information to:

- Identify the underlying framework, programming language, or runtime environment (e.g., PHP, ASP.NET, Node.js).
- Discover known vulnerabilities and exploit unpatched security flaws.

Analysis: The HTTP response headers of the target application http://testphp.vulnweb.com/ were analyzed and found to expose sensitive server-side information.

- The "X-Powered-By" header reveals details about the backend technology in use.



- Attackers can use tools like Burp Suite, cURL, or Nikto to extract this information and find relevant exploits.

Recommendations

To mitigate the risk of leaking sensitive server information, follow these security best practices:

1. Remove or Obfuscate the "X-Powered-By" Header
2. Use a Web Application Firewall (WAF)
3. Regularly Update Server Software
4. Implement Security Headers

APPENDIX - TOOLS USED

TOOL	DESCRIPTION
Burp Suite & Zap	Used for intercepting and testing web application requests, identifying vulnerabilities, and automating SQL Injection tests.
SQLmap	Automated tool used to detect and exploit SQL Injection vulnerabilities in web applications.
Nmap	Used to scan ports and identify open services that may be vulnerable.
PostgreSQL/MySQL	Used to connect to the database and verify unauthorized access from SQL Injection.