

REPORT

Vulnerability Assessment and Penetration Testing

Sakshi Chandesurye

5th February, 2025



TABLE OF CONTENTS

1. OVERVIEW

- 1.1 Executive Summary
- 1.2 Background
- 1.3 Objectives
- 1.4 Scope of Assessment
- 1.5 Tools Used
- 1.6 Summary of Findings

2. VULNERABILITY DETAILS

- 2.1 Cross-Site Scripting
- 2.2 SQL Injection
- 2.3 SQL Injection - MySQL
- 2.4 SQL Injection - Oracle Time Based
- 2.5 SQL Injection - SQLite
- 2.6 Absence of Anti-CSRF Tokens
- 2.7 Content Security Policy (CSP) Header Not Set
- 2.8 Directory Browsing



2.9 Missing Anti-clickjacking Header

2.10 Server Leaks Information via "X-Powered-By" HTTP Response Header

2.11 Server Leaks Version Information via "Server" HTTP Response Header

2.12 X-Content-Type-Options Header Missing

2.13 XSLT Injection

2.14 Modern Web Application Identified

2.15 Authentication Request Identified

2.16 Charset Mismatch (Header Versus Meta Content-Type Charset)

2.17 GET for POST

2.18 Information Disclosure - Suspicious Comments

2.19 User Agent Fuzzer

2.20 User Controllable HTML Element Attribute

3. APPENDIX



1. OVERVIEW

1.1 Executive Summary

A white-box penetration test was conducted for the web application, AcuArt, a demonstration site for Acunetix Web Vulnerability Scanner. The assessment was limited to specific procedures and analysis based on the information available during the testing period from 3 February 2025 to 5 February 2025. Changes in circumstances after this date could affect the findings outlined in this report.

The purpose of this Web Security Assessment was to identify exploitable vulnerabilities and insufficiently configured security controls to determine the likelihood that users with varying levels of knowledge (e.g., informed and uninformed insiders and outsiders) could exploit weaknesses in the application as those cataloged in the OWASP Top 10, OWASP ASVS, OWASP Testing Guide, and Penetration Testing Execution Standard.

The assessment also included a review of security controls and requirements listed in the OWASP Application Security Verification Standard (ASVS). This review does not, nor is it intended to, identify all potential vulnerabilities.

Automated and manual testing methodologies were used during the assessment. Automated tools like OWASP ZAP, BurpSuite facilitated initial scans, but the majority of the assessment involved manual analysis.

The results summarized in this document are based upon a collection of methodologies and tests interacting at a single point in time with technology that is continually changing. Any projection to the future of the findings contained in this document is subject to the risk that they may no longer portray the system or environment accurately due to changes.

1.2 Background

The primary objective of this assessment was to perform a comprehensive penetration test on the AcuArt application to identify and exploit potential vulnerabilities. The focus was on manual and automated testing methods to detect vulnerabilities and provide recommendations for remediation.

The assessment was conducted to identify the security vulnerabilities within the web application and propose solutions to enhance its security posture.

1.3 Objectives

The objective of the tests performed was to:

- Identify possible vulnerabilities and gaps in the AcuArt application.
- Assess the severity of the identified vulnerabilities and determine the associated risk.
- Suggest recommendations to remediate the identified vulnerabilities and gaps.

This assessment report contains:

- Technical details of the vulnerabilities discovered with evidence of exploits.
- Risk mitigation recommendations that need to be implemented to ensure that the system is secure from the risks arising due to the discovered vulnerabilities.


1.4 Scope of Assessment

Scope of Assessment

The security assessment was conducted using automated and semi-automated dynamic testing methodologies through Burp Suite and OWASP ZAP to identify potential security vulnerabilities.

The scope included, but was not limited to, the following areas:

1. **Information Gathering:** Enumeration of application endpoints, crawling the application, and reconnaissance to understand its structure and functionalities.
2. **Configuration Analysis:** Automated checks for security misconfigurations in server headers and application configurations.

- 
3. **Input Validation and Injection Testing:** Automated scanning for input-related vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

URLs

<http://testphp.vulnweb.com>

Environment Details

Application Name	AcuArt
Environment	Test Site
Accessibility	Internet
Authentication Method	Login

1.5 Tools Used

The following tools are used for the testing.

- Burp Suite
- OWASP ZAP
- Nmap

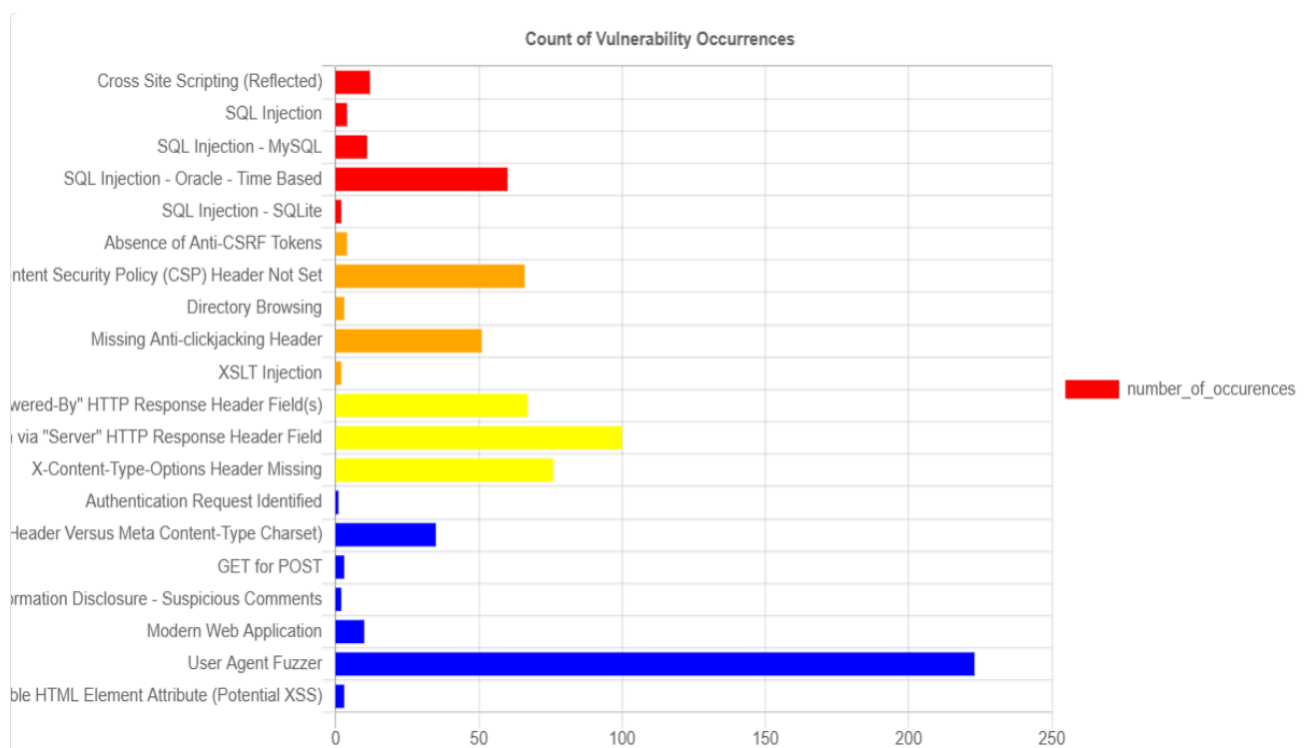
1.6 Summary Of Findings

The test identified a total of 20 security vulnerabilities across the target application, categorized by risk levels: 5 High, 5 Medium, 3 Low, and 7 Informational. Key findings included cross-site scripting, multiple SQL injection vulnerabilities, absence of anti-CSRF tokens, missing security headers, and information disclosure issues. The assessment highlights critical areas requiring immediate attention to enhance the security posture of the application. Recommended remediation measures focus on input validation, secure header implementation, and robust user session management to mitigate identified risks.

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	5
Low	3
Informational	7

Frequency Of Vulnerability



- The most critical vulnerabilities include SQL Injection (multiple types) and Cross-Site Scripting (Reflected), totaling 89 instances. Medium-risk issues, such as missing security headers and directory browsing, account for significant occurrences, with 66 instances of CSP header misconfigurations.
- Low-risk issues, like server information leaks, occur frequently, with 100 instances related to version disclosures. Informational findings, including User Agent Fuzzer (223 instances) and charset mismatches (35 instances), indicate areas for best practice improvements.

2. VULNERABILITY DETAILS

2.1 Cross-Site Scripting

Parameter	Description
Severity	High
Impact	High
Risk Score	8.9
Affected URL	http://testphp.vulnweb.com/hpp/params.php?p=<script>alert(1);</script>&pp=12
Threat	<p>The impact of a successful CSRF attack can be severe, leading to unauthorized actions :</p> <ol style="list-style-type: none">1. Account Hijacking (Cookie Theft)2. Browser Redirection3. Displaying Fraudulent Content4. Potential System Compromise

Cross-Site Scripting (XSS) is a vulnerability that occurs when an application includes attacker-supplied code in a user's browser instance without proper validation. This can lead to severe consequences, as the malicious code executes within the security context of the hosting website.

Reflected XSS occurs when the malicious payload is delivered through a URL and executed immediately upon visiting the page, without persistence in the application backend. In this instance, the attack vector was embedded within the query string of the affected URL.

Resolution Measures

To mitigate the risk of XSS attacks and protect the application:

1. Output Encoding:

- Use encoding libraries such as OWASP ESAPI Encoding module or Microsoft's Anti-XSS library to properly encode output.

2. Input Validation:

- Implement strict input validation for all user-supplied data, particularly in query parameters.

3. Content Security Policy (CSP):

- Enforce CSP headers to limit browser script execution and prevent the injection of unauthorized scripts.

4. Use Framework Security Features:

- Leverage secure web frameworks that provide built-in XSS protection, such as Apache Wicket.

5. Testing:

- Regularly perform security testing to identify and remediate XSS vulnerabilities promptly.

Evidence:

The screenshot displays the ZAP (Zed Attack Proxy) interface during a security scan. The top pane shows the 'Sites' tree on the left and the 'Request' and 'Response' details on the right. The 'Response' pane shows an HTTP 200 OK status with a content-type of text/html. The body of the response contains a message about a user being introduced to the database, with a red box highlighting a successful XSS payload: `<script>alert(1)</script>`. The bottom pane shows the 'Alerts' list, which includes a 'Cross Site Scripting (Reflected)' alert for the URL `http://testphp.vulnweb.com/secured/newuser.php`. The alert details show the attack payload and the evidence of the alert being triggered. The status bar at the bottom indicates 'Current Scans' with various icons and a 'Main Proxy: localhost:8081'.

2.2 SQL Injection

Parameter	Description
Severity	High
Impact	High
Risk Score	8.7
Affected URL	http://testphp.vulnweb.com/artists.php?artist=4-2
Threat	<p>Impact of SQL Injection:</p> <ol style="list-style-type: none">1. Data Breach: Unauthorized access to sensitive data such as user information.2. Data Manipulation: Insertion, deletion, or modification of database entries.3. Authentication Bypass: Circumvention of authentication mechanisms.4. Privilege Escalation: Elevation of attacker privileges within the system.

SQL Injection is a critical vulnerability that occurs when untrusted user input is improperly sanitized, allowing attackers to inject malicious SQL statements. This can lead to unauthorized data access, data modification, or even complete database compromise.

Resolution Measures

To mitigate SQL Injection vulnerabilities, implement the following security best practices:

1. **Input Validation:**
 - Never trust client-side input, even if client-side validation exists.
 - Perform strict input validation on the server side.
2. **Parameterized Queries:**
 - Use **PreparedStatement** or **CallableStatement** in JDBC.
 - For ASP applications, use ADO Command Objects with strong type checking and parameterized queries.
 - Avoid dynamic SQL query construction using string concatenation.
3. **Stored Procedures:**
 - Use stored procedures instead of dynamic SQL queries.

- Avoid using **exec**, **exec immediate**, or equivalent dynamic execution functions.

4. Escape Input Data:

- Apply strict character escaping for data inputs from clients.
- Implement an allow list or deny list for character validation in user inputs.

5. Least Privilege Principle:

- Use the least privileged database user possible for application interactions.
- Avoid using privileged accounts like **sa** or **db-owner**.

6. Secure Database Configuration:

- Limit the database access permissions required for application operations.

7. Regular Security Testing:

- Conduct regular penetration testing and static code analysis to detect and remediate SQL injection vulnerabilities.

Evidence:

The screenshot displays the Burp Suite interface. The top pane shows the 'Sites' tab with a tree view of the target site 'http://testphp.vulnweb.com'. The middle pane shows the 'Response' tab for a GET request to 'http://testphp.vulnweb.com/artists.php?artist=4-2'. The response body contains a JavaScript function `MM_reloadPage` which is highlighted with a red box. The bottom pane shows the 'Alerts' tab with a list of detected issues. Two SQL Injection alerts are highlighted with a red box: 'GET: http://testphp.vulnweb.com/artists.php?artist=4-2' and 'GET: http://testphp.vulnweb.com/product.php?pic=6-2'. The details for the first alert are expanded, showing a high risk, medium confidence, and evidence of a successful SQL injection attack using the payload '4-2'.

SQL Injection Alert Details:

- URL: `http://testphp.vulnweb.com/artists.php?artist=4-2`
- Risk: High
- Confidence: Medium
- Parameter: `artist`
- Attack: `4-2`
- Evidence:
 - CWE ID: 89
 - WASC ID: 19
 - Source: Active (40018 - SQL Injection)
 - Input Vector: URL Query String
 - Description: SQL injection may be possible.
- Other Info:
 - The original page results were successfully replicated using the expression `[4-2]` as the parameter value
 - The parameter value being modified was stripped from the HTML output for the purposes of the comparison.

2.3 SQL Injection - MySQL

Parameter	Description
Severity	High
Impact	High
Risk Score	8.9
Affected URL	http://testphp.vulnweb.com/secured/newuser.php
Threat	<p>Impact of SQL Injection (MySQL):</p> <ol style="list-style-type: none">1. Data Breach: Exposure of sensitive information such as user data.2. Data Manipulation: Unauthorized alteration or deletion of database records.3. Authentication Bypass: Circumvention of login mechanisms.4. Database Control: Potential full control of the database by attackers.

SQL Injection is a critical vulnerability that arises when user input is improperly sanitized, allowing attackers to inject malicious SQL statements. This can result in unauthorized access to data, data manipulation, and potential control of the database.

Resolution Measures

To mitigate SQL Injection vulnerabilities in MySQL, implement the following security measures:

1. **Input Validation:**
 - Never trust client-side input, even if client-side validation exists.
 - Strictly validate user inputs on the server side to ensure compliance with expected data types.
2. **Parameterized Queries:**
 - Use **PreparedStatement** or **CallableStatement** in JDBC to prevent direct SQL query concatenation.
 - Ensure parameters are passed using placeholders such as ?.
3. **Stored Procedures:**
 - Use stored procedures instead of dynamically concatenated queries.

- Avoid commands like `exec`, `exec immediate`, or equivalent dynamic execution functions.

4. Escape User Input:

- Properly escape all input from users to avoid injection of special characters.
- Implement an allow list or deny list for input validation.

5. Least Privilege Principle:

- Use the minimum privileges necessary for database interactions.
- Avoid using privileged database users such as `root` or `db-admin`.

6. Database Configuration:

- Grant only the minimum access required for application operations.

7. Error Message Masking:

- Ensure that database error messages are not exposed to users to prevent information leakage.

8. Regular Security Testing:

- Conduct regular penetration testing and automated code analysis to detect SQL Injection vulnerabilities.

Evidence:

The screenshot shows the Burp Suite interface. The top pane displays the HTTP response body for a request to `http://testphp.vulnweb.com/secured/newuser.php`. The response is an HTML document with a message: "Unable to access user database: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '''' at line 1". The bottom pane shows the 'Alerts' tab, which lists several security alerts. A red box highlights the 'SQL Injection - MySQL (5)' alert. The alert details include the URL, the parameter 'uname', and the evidence 'You have an error in your SQL syntax'.

2.4 SQL Injection - Oracle Time Based

Parameter	Description
Severity	High
Impact	High
Risk Score	9.0
Affected URL	http://testphp.vulnweb.com/userinfo.php
Threat	Impact of SQL Injection (Oracle Time Based): <ol style="list-style-type: none">1. Data Breach: Unauthorized extraction of sensitive information.2. Privilege Escalation: Execution of administrative commands.3. Service Disruption: Resource exhaustion and potential Denial of Service (DoS).4. Data Integrity: Unauthorized modification of critical data.

A time-based blind SQL Injection vulnerability was identified in the `uname` parameter. The attacker manipulated the database query execution time using Oracle's `UTL_INADDR.get_host_name()` function calls. This demonstrates the capability to execute arbitrary SQL commands and infer query results based on response delays.

Resolution Measures

To prevent SQL Injection vulnerabilities, implement the following best practices:

1. Parameterized Queries:

- Use **PreparedStatement** or **CallableStatement** in Java, or parameterized queries in other languages.
- Avoid dynamic SQL queries constructed using string concatenation.

2. Input Validation:

- Validate all user inputs against a strict **allow list** of expected characters. Reject any input that does not conform.

3. Stored Procedures:

- Prefer using stored procedures that do not build dynamic SQL statements internally.

4. Privilege Management:

- Use **least privileged database users**, and avoid high-privileged accounts (**sa**, **db-owner**).
- Grant only the minimum access required for the application.

5. Escaping Special Characters:

- Apply appropriate database-specific escaping for user inputs when parameterized queries are not possible.

6. Database Security Features:

- Enable database security features such as Oracle Database Vault and auditing mechanisms.

Evidence:

The screenshot displays the ZAP interface with an alert for an SQL injection attack on the URL `http://testphp.vulnweb.com/userinfo.php`. The alert is titled "SQL Injection - Oracle - Time Based" and is categorized as "High" risk. The attack payload is shown as a union-based query: `uname-->SELECT--UTL_INADDR.get_host_name(2832718.0.0.1%27%29+from-dual+union+SELECT--UTL_INADDR.get_host_name(2832718.0.0.2%27%29+from-dual+union+SELECT--UTL_INADDR.get_host_name(2832718.0.0.3%27%29+from-dual+union+SELECT--UTL_INADDR.get_host_name(2832718.0.0.4%27%29+from-dual+union+SELECT--UTL_INADDR.get_host_name(2832718.0.0.5%27%29+from-dual+union+SELECT--pass-ZAP`. The evidence section shows the response time for the attack query is 20,321 milliseconds, compared to 603 milliseconds for the original query.

The HTTP request details are also visible, showing a POST request to `http://testphp.vulnweb.com/userinfo.php` with a user-agent of `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36`.

The response time with the payload was approximately 20,321 milliseconds, significantly higher than the unmodified query, which took 603 milliseconds, confirming the potential for a time-based attack.

2.5 SQL Injection - SQLite

Parameter	Description
Severity	High
Impact	High
Risk Score	9.0
Affected URL	GET http://testphp.vulnweb.com/product.php?pic=case randblob(100000000) when not null then 1 else 1 end
Threat	Impact of SQL Injection (SQLite): <ol style="list-style-type: none">1. Data Breach: Unauthorized access to sensitive database information.2. Denial of Service: Prolonged query execution times leading to resource exhaustion.3. Data Integrity: Unauthorized modification of data.4. Privilege Escalation: Execution of malicious commands.

A time-based SQL Injection vulnerability was identified in the **pic** parameter. The SQLite **randblob()** function was exploited to influence query execution time. The response time was manipulated to infer query execution success, confirming a potential SQL Injection vulnerability.

Resolution Measures

To mitigate SQL Injection vulnerabilities, implement the following security measures:

1. **Parameterized Queries:**
 - Use parameterized queries to safely handle user input and avoid query manipulation.
2. **Input Validation:**
 - Validate all user inputs against a strict **allow list**. Reject any unexpected input values.
3. **Input Length Restrictions:**
 - Limit the length of input parameters to reasonable bounds to reduce the effectiveness of large payload attacks.

4. Escaping Special Characters:

- Use database-specific methods to escape special characters when parameterized queries are not an option.

5. Privilege Management:

- Ensure that the database user has the minimum privileges required for the application.
- Avoid granting unnecessary privileges that could exacerbate the impact of a successful attack.

6. Error Handling:

- Suppress detailed database error messages to prevent attackers from gaining insight into the database schema.

7. Monitoring and Alerts:

- Implement query execution time monitoring and alert systems to detect abnormal delays.

8. Security Testing:

- Conduct regular penetration tests to identify and remediate SQL Injection vulnerabilities.

Evidence:

The screenshot displays the Burp Suite interface during a security scan. The 'Alerts' tab is active, showing a list of detected issues. An alert for 'SQL Injection - SQLite' is highlighted, with a red box around its entry in the list. The details pane on the right provides information about this alert:

- URL:** http://testphp.vulnweb.com/product.php?pic=4
- Risk:** High
- Confidence:** Medium
- Parameter:** pic
- Attack:** case randblob(100000000) when not null then 1 else 1 end
- Evidence:** The query time is controllable using parameter value [case randblob(100000000) when not null then 1 else 1 end], which caused the request to take [573] milliseconds, parameter value [case randblob(1000000000) when not null then 1 else 1 end], which caused the request to take [766] milliseconds, when the original unmodified query with value [4] took [633] milliseconds.
- CWE ID:** 89
- WASC ID:** 19
- Source:** Active (40024 - SQL Injection - SQLite)
- Input Vector:** URL Query String
- Description:** SQL injection may be possible.
- Other Info:** The query time is controllable using parameter value [case randblob(100000000) when not null then 1 else 1 end], which caused the request to take [573] milliseconds, parameter value [case randblob(1000000000) when not null then 1 else 1 end], which caused the request to take [766] milliseconds, when the original unmodified query with value [4] took [633] milliseconds.

The top pane shows the HTTP response body, which contains a JavaScript function `popupWindow` that opens a new window. The bottom pane shows the 'History' tab with a list of requests, including the one that triggered the alert.

Payload `case randblob(100000000) when not null then 1 else 1 end` increased response time to 766 ms.

2.6 Absence of Anti-CSRF Tokens

Parameter	Description
Severity	Medium
Impact	High
Risk Score	7.5
Affected URL	http://testphp.vulnweb.com/guestbook.php
Threat	The impact of absence of unpredictable Anti-CSRF tokens: <ol style="list-style-type: none">1. Unauthorized actions performed using the victim's session.2. Information disclosure, especially when combined with XSS vulnerabilities.3. Potential for session hijacking and privilege abuse.

Cross-Site Request Forgery (CSRF) is a security vulnerability that allows an attacker to trick a victim into submitting unintended and potentially harmful requests on a target website where the user is authenticated. This vulnerability arises due to the absence of unpredictable Anti-CSRF tokens in HTML submission forms, allowing attackers to craft malicious requests that appear legitimate.

Conditions for Successful Attack:

- Victim has an active session on the target site.
- Victim is authenticated via HTTP auth or shares a local network with the target.
- Application functionality relies on predictable or repeatable URL/form actions.

Resolution Measures

To mitigate the risk of CSRF attacks and secure web applications:

1. **Anti-CSRF Token Implementation:**
 - Generate a unique, unpredictable token for each form submission.
 - Include the token in form submissions and validate it on the server side.
2. **Secure Framework Usage:**
 - Utilize frameworks or libraries that natively provide CSRF protection, such as OWASP CSRFGuard.

3. Non-Predictable Nonce:

- Ensure that each form contains a unique and non-predictable nonce for validation.

4. Referrer Header Checking:

- Verify the Referrer header to ensure requests originate from expected sources. (Note: This may break legitimate functionality due to privacy settings.)

5. Avoid GET Requests for State Changes:

- Use POST requests for operations that trigger state changes in the application.

6. Dangerous Operation Confirmation:

- Require additional confirmation from users for critical operations.

7. Regular Security Testing:

- Continuously test for the presence of Anti-CSRF tokens and other CSRF defenses.

Evidence:

The screenshot displays the Burp Suite interface. The top pane shows a web request to `http://testphp.vulnweb.com/guestbook.php` with a POST method. The request body contains a form action and a hidden name parameter. The bottom pane shows a security alert titled "Absence of Anti-CSRF Tokens" for the same URL. The alert details the risk, confidence, and evidence, including the form action and the absence of Anti-CSRF tokens in the HTML submission form.

Request Details:

- Method: POST
- URL: `http://testphp.vulnweb.com/guestbook.php`
- Body: `<form action="" method="post" name="faddentry">`

Alert Details:

- Title: Absence of Anti-CSRF Tokens
- URL: `http://testphp.vulnweb.com/guestbook.php`
- Risk: Medium
- Confidence: Low
- Parameter: `name`
- Attack: `<form action="" method="post" name="faddentry">`
- Evidence: `<form action="" method="post" name="faddentry">`
- CWE ID: 352
- WASC ID: 9
- Source: Passive (10202 - Absence of Anti-CSRF Tokens)
- Description: No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting

2.7 Content Security Policy (CSP) Header Not Set

Parameter	Description
Severity	Medium
Impact	Medium
Risk Score	6.0
Affected URL	http://testphp.vulnweb.com/robots.txt
Threat	<p>The Impact of Content Security Policy (CSP) Header Not Set:</p> <ol style="list-style-type: none">1. Cross-Site Scripting (XSS): Malicious code injection leading to data theft or session hijacking.2. Clickjacking: Overlaying malicious content to deceive users.3. Data Exfiltration: Untrusted scripts exfiltrating sensitive data.4. Resource Loading Exploits: Loading unauthorized scripts or multimedia files.

The web application does not set a **Content Security Policy (CSP)** header, which leaves it vulnerable to content injection attacks such as **Cross-Site Scripting (XSS)** and **data injection**. CSP is a critical security control that instructs browsers to only load resources from trusted sources, significantly reducing the attack surface.

Resolution Measures

To mitigate the risk of content injection, configure your server to include a secure **Content-Security-Policy** header.

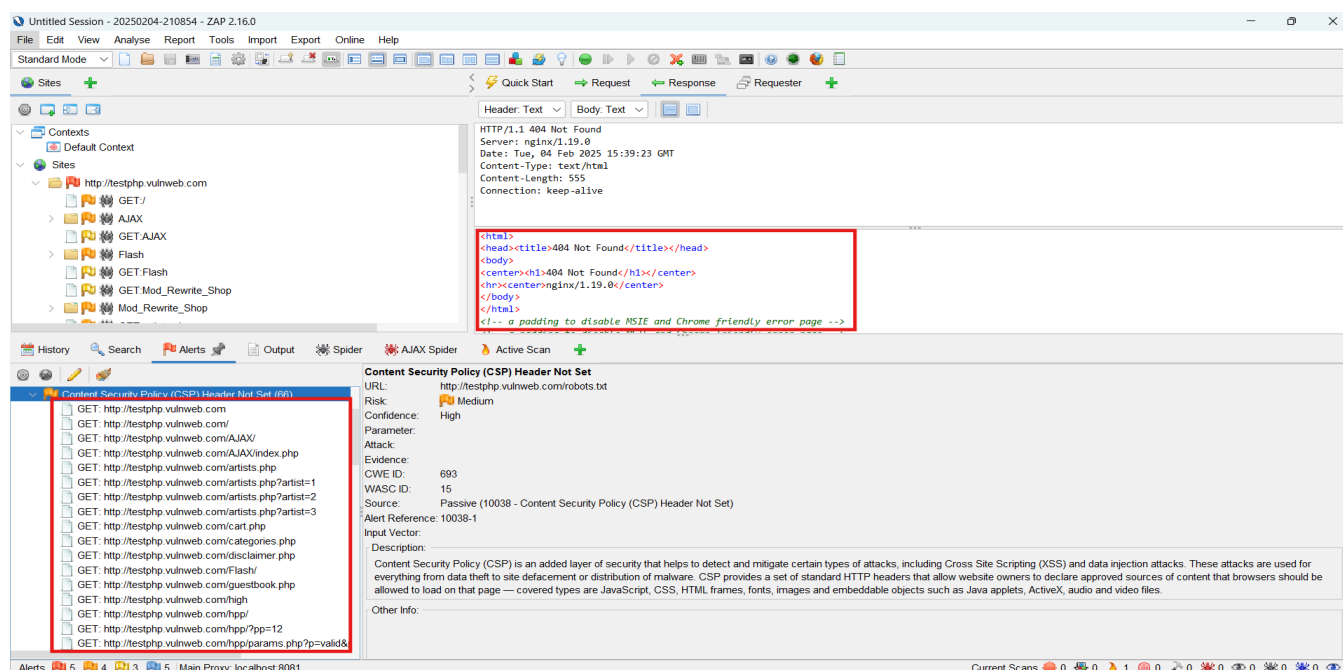
1. Testing and Verification

- Use browser developer tools to verify that the CSP header is properly set.
- Monitor the browser console for CSP violations.

2. Regular CSP Review

- Review and update the CSP policy periodically as new resources are integrated into the website.

Evidence:



2.8 Directory Browsing

Parameter	Description
Severity	Medium
Impact	Medium
Risk Score	5.0
Affected URL	http://testphp.vulnweb.com/Flash/
Threat	<p>The Impact of Directory Browsing:</p> <ol style="list-style-type: none"> Sensitive Information Disclosure: Exposed files may reveal critical details about the application structure or sensitive data. Unauthorized Access: Accessible files may be exploited to gain further access to the system. Targeting Vulnerable Files: Attackers can identify older files or unused scripts that may contain vulnerabilities.

The web server allows directory browsing, making it possible for attackers to view and access directory contents. Exposed files may include sensitive information such as scripts, configuration files, backup source files, or other resources that can be exploited.

Resolution Measures

1. Apply Access Controls

Limit access to sensitive directories by configuring appropriate permissions.

2. Remove Unnecessary Files

- Delete backup, temporary, and obsolete files.
- Regularly audit directory contents.

3. Verify the Fix

- Attempt to access the directory URL after configuration changes to ensure directory listing is no longer accessible.
- Use automated scanners to detect any further directory browsing vulnerabilities.

Evidence:

The screenshot displays the Burp Suite interface. The top pane shows an HTTP response from a directory listing. The response body contains HTML code for a directory listing of `/Flash/`. A red box highlights the `<title>Index of /Flash/</title>` line. The bottom pane shows a list of detected vulnerabilities. The 'Directory Browsing' vulnerability is highlighted with a red box. The details for this vulnerability are as follows:

URL	Risk	Confidence	Parameter	Attack	Evidence	CWE ID	WASC ID	Source	Input Vector	Description	Other Info
http://testphp.vulnweb.com/Flash/	Medium	Medium			<title>Index of /Flash/</title>	548	16	Passive (10033 - Directory Browsing)		It is possible to view a listing of the directory contents. Directory listings may reveal hidden scripts, include files, backup source files, etc., which can be accessed to reveal sensitive information.	Web server identified: Apache 2

2.9 Missing Anti-clickjacking Header

Parameter	Description
Severity	Medium
Impact	Medium
Risk Score	5.0
Affected URL	http://testphp.vulnweb.com/Flash/
Threat	The Impact of Missing Anti-clickjacking Header: <ol style="list-style-type: none">1. User Interaction Manipulation2. Privacy Violations3. Unauthorized Actions4. Session Hijacking

Missing anti-clickjacking protection puts users at risk of clickjacking attacks. Clickjacking occurs when a malicious website embeds the target site inside a frame or iframe, potentially tricking the user into clicking on something they didn't intend, leading to unintended actions like account changes or malicious activity.

The website's response does not include proper protection against clickjacking attacks. Specifically, it lacks the `X-Frame-Options` or `Content-Security-Policy` headers to prevent the page from being framed by an external site. As a result, an attacker could embed the site in an invisible iframe, misleading users into performing unintended actions.

Resolution of Vulnerability:

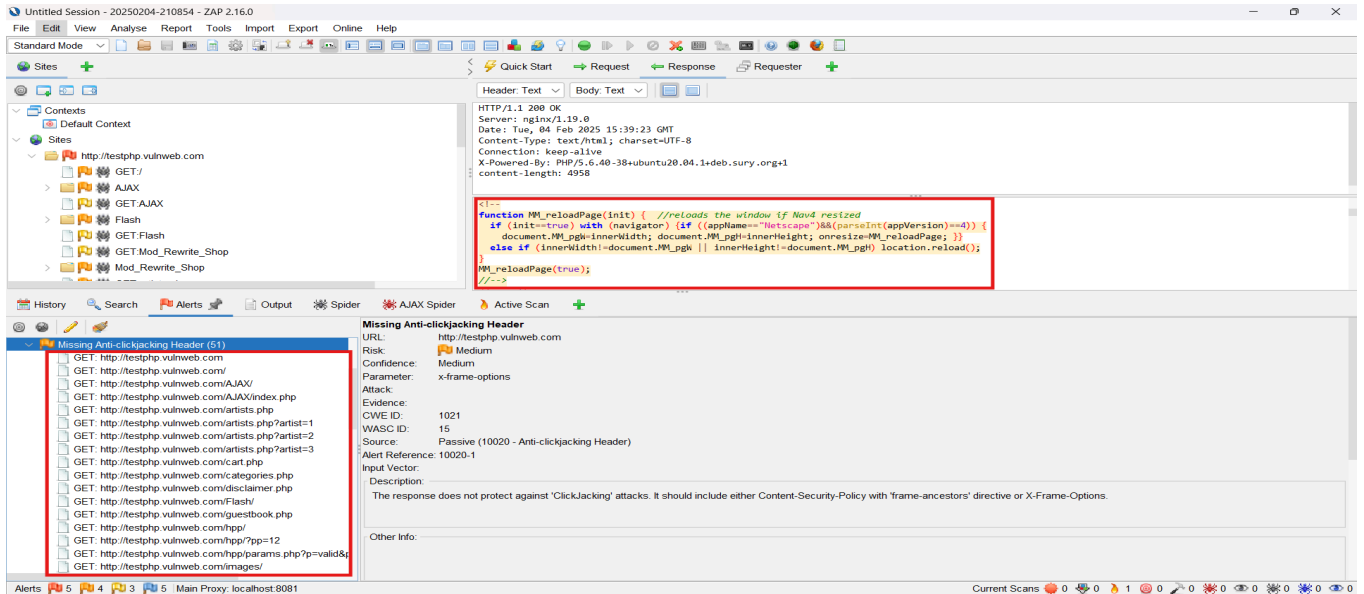
To mitigate this risk and prevent clickjacking attacks, implement one of the following security headers:

1. **X-Frame-Options:**
 - Set this header to **DENY** to prevent all sites from framing the page, or **SAMEORIGIN** to allow only your site to frame it.
2. **Content-Security-Policy (CSP):**
 - Implement the `frame-ancestors` directive to define which domains can

embed the content in a frame.

- Example: **Content-Security-Policy: frame-ancestors 'self';**

Evidence:



2.10 Server Leaks Information via "X-Powered-By" HTTP Response Header

Parameter	Description
Severity	Medium
Impact	Low
Risk Score	4.5
Affected URL	http://testphp.vulnweb.com/index.php
Threat	The Impact of Server Leaks Information via "X-Powered-By" HTTP Response Header: <ol style="list-style-type: none">1. Targeted Attacks on Known Vulnerabilities2. Information Disclosure3. Increased Attack Surface

Leaking information about the web server or application through the **X-Powered-By** HTTP response header allows attackers to gain insight into the technologies in use, such as the server type, framework, or version. This can aid attackers in identifying known vulnerabilities in these components, increasing the risk of exploitation.

The web server is leaking information via the X-Powered-By HTTP header, which reveals details about the PHP version. This could potentially help attackers identify vulnerabilities in the version of PHP or other technologies used, making it easier for them to target specific weaknesses.

Resolution of Vulnerability:

To reduce the exposure of sensitive information:

1. Disable the X-Powered-By Header:

Configure the web server (e.g., Nginx, Apache) to suppress the X-Powered-By header.

In Nginx, this can be done by adding `fastcgi_hide_header X-Powered-By;` in the configuration.

2. Server and Application Configuration:

Make sure that other server or application headers that may reveal information are also configured to suppress unnecessary details.

Evidence:

The screenshot displays the Burp Suite interface. The top pane shows an HTTP response from `http://testphp.vulnweb.com`. The response headers include `X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1deb.sury.org+1`. The response body contains a JavaScript function `function MM_reloadPage(init) { //reloads the window if Nav4 resized`. The bottom pane shows a list of alerts, with one alert titled "Server Leaks Information via 'X-Powered-By' HTTP Response Header Field(s)" selected. The alert details include the URL `http://testphp.vulnweb.com/index.php`, risk level `Low`, confidence `Medium`, and a description stating that the web application is leaking information via the X-Powered-By header.

2.11 Server Leaks Version Information via "Server" HTTP Response Header

Parameter	Description
Severity	Low
Impact	Low
Risk Score	4.0
Affected URL	http://testphp.vulnweb.com/sitemap.xml
Threat	The Impact of Server Leaks Version Information via "Server" HTTP Response Header: <ul style="list-style-type: none">1. Version-Specific Exploits2. Information Disclosure3. Targeted Attack Preparation

The Server HTTP response header reveals version information about the web server (nginx/1.19.0). This can assist attackers in identifying which version of the server is in use and searching for known vulnerabilities associated with that version. Even though a 404 response is returned, the header still exposes valuable information to attackers.

Resolution of Vulnerability:

To mitigate the risk of exposing version information:

- 1. Disable the Server Header:**

Configure the web server (e.g., Nginx, Apache) to suppress the Server header. In Nginx, add `server_tokens off;` to the configuration to prevent the server version from being exposed.

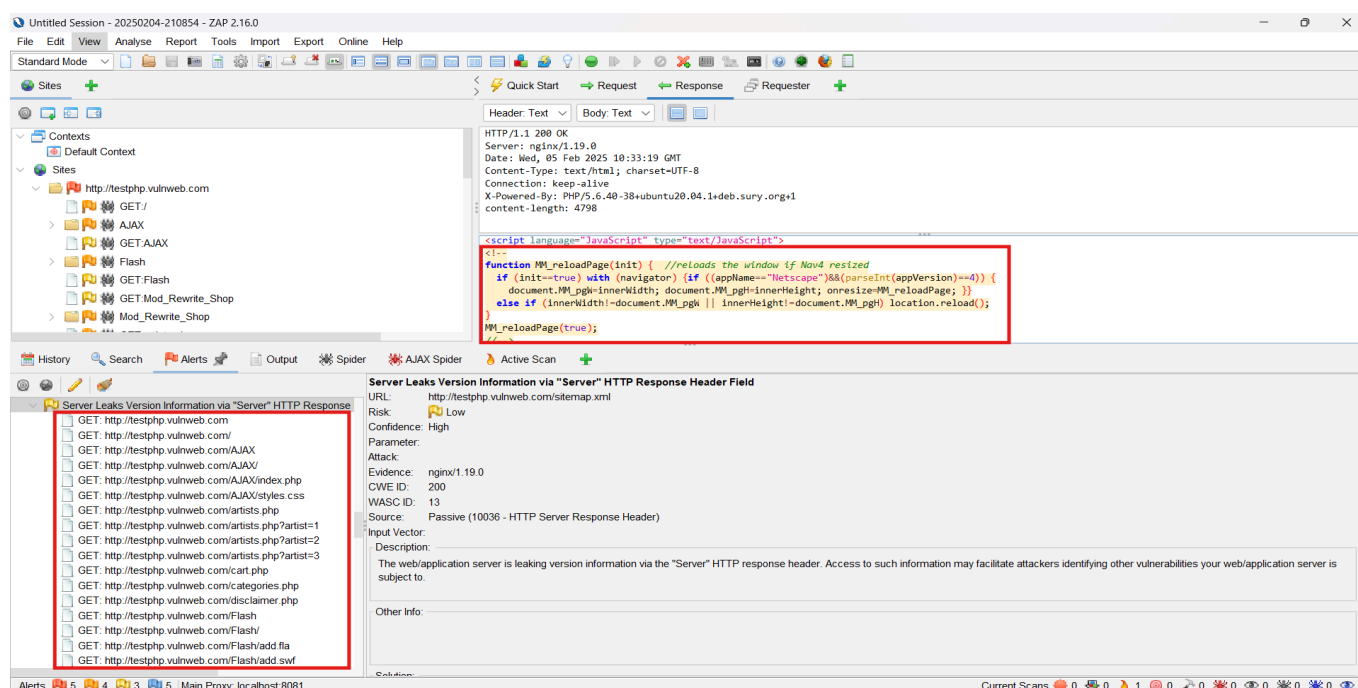
- 2. Generic Server Response:**

Consider providing a generic response in the Server header, such as "Apache" or "nginx," without revealing the specific version.

- 3. Regular Updates:**

Ensure the server and related components are regularly updated to minimize the risk of exploiting known vulnerabilities.


Evidence:



2.12 X-Content-Type-Options Header Missing

Parameter	Description
Severity	Low
Impact	Medium
Risk Score	5.0
Affected URL	http://testphp.vulnweb.com/index.php
Threat	The Impact of X-Content-Type-Options Header Missing: <ul style="list-style-type: none"> • MIME-Sniffing Attacks • Content Type Confusion • Injection Issues in Error Pages

The absence of the **X-Content-Type-Options** header with the **nosniff** directive allows browsers, especially older versions of Internet Explorer and Chrome, to perform MIME-sniffing. This could lead to the response body being interpreted as a different content type than intended, potentially causing unexpected behavior or security issues.



While modern browsers (such as Firefox) honor the declared content type, legacy browsers might misinterpret the data.

The response does not include the X-Content-Type-Options header with the nosniff directive, which is used to prevent browsers from interpreting files as a different type than declared in the Content-Type header. This issue is especially concerning for error pages (401, 403, 500), where malicious actors could inject content into responses. MIME-sniffing vulnerabilities are more likely in older browsers, leading to unexpected behavior or security issues.

Resolution of Vulnerability:

To mitigate the risk of MIME-sniffing attacks:

1. Set the X-Content-Type-Options Header:

Configure the web server to include the header X-Content-Type-Options: nosniff for all responses. This will prevent browsers from performing MIME-sniffing on the response body.

2. Ensure Proper Content-Type Declaration:

Ensure that all responses, especially error pages, have the correct Content-Type header.

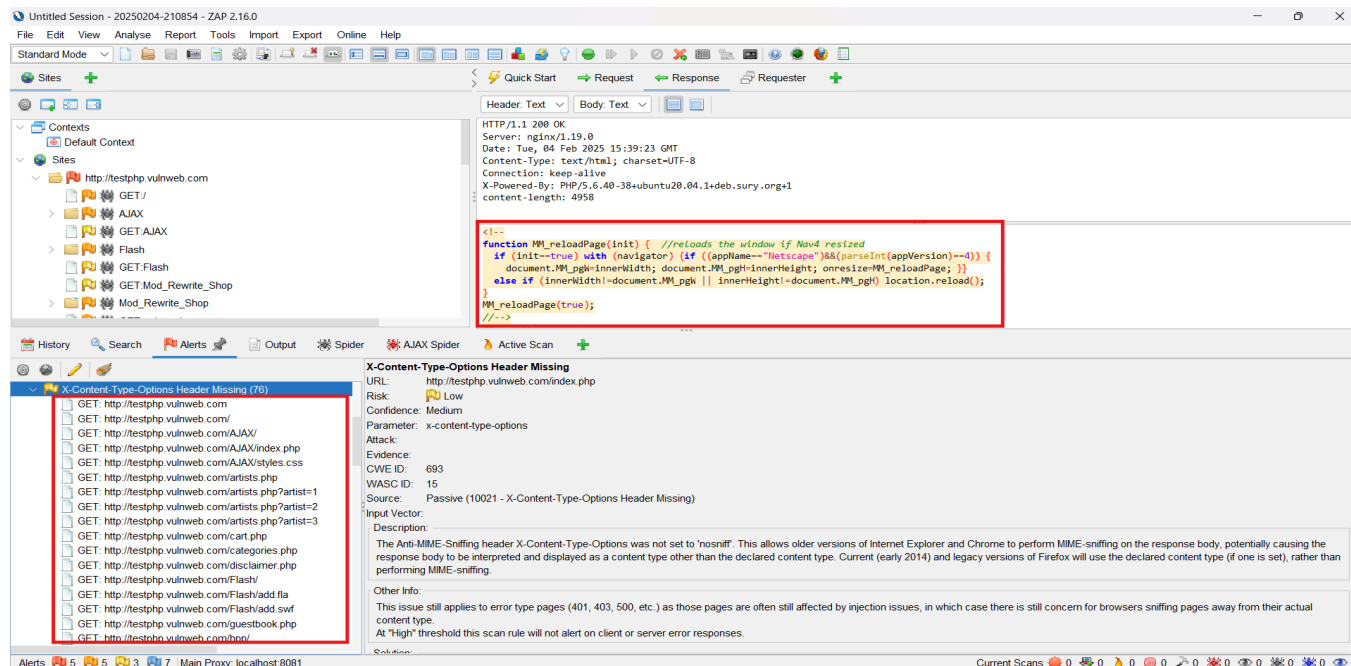
3. Encourage Modern Browsers:

Promote the use of modern browsers that do not rely on MIME-sniffing or provide configurations that avoid it.

4. Review and Update Legacy Browsers:

Encourage users to upgrade to modern, standards-compliant browsers that do not rely on MIME-sniffing. Older browsers may be more vulnerable to these types of attacks.


Evidence:



2.13 XSLT Injection

Parameter	Description
Severity	Medium
Impact	High
Risk Score	8.2
Affected URL	<code>http://testphp.vulnweb.com/showimage.php?file=%3C%3Avalue-of-select%3D%22document%28%27http%3A%2F%2Ftestphp.vulnweb.com%3A22%27%29%22%2F%3E</code>
Threat	<p>Potential issues include:</p> <ul style="list-style-type: none"> Access or manipulate sensitive system information Perform unauthorized file read and write operations Execute arbitrary code on the server Potentially conduct internal port scanning

XSLT (Extensible Stylesheet Language Transformations) Injection occurs when user-controlled input is directly processed by an XSLT engine without proper validation or sanitization. Injection using XSL transformations may be possible, and may



allow an attacker to read system information, read and write files, or execute arbitrary code. The request in this instance attempted to read external system resources, resulting in the server producing warnings due to failed stream access.

Resolution of Vulnerability

1. Input Validation and Sanitization:

- Strictly validate and sanitize all user-supplied input before processing.
- Reject or escape potentially dangerous characters (<, >, ", ', &) in input intended for XSL transformations.
- Implement whitelisting techniques for allowed input formats.

2. Disable Dangerous XSL Features:

- Configure the XSLT processor to disable or restrict insecure functions, such as access to external resources through the `document()` function.

3. Use Parameterized Stylesheets:

- Where possible, utilize parameterized stylesheets that safely handle user data without exposing XSLT functions directly.

4. Apply Content Security Policies (CSP):

- Implement CSP headers to restrict the execution of unauthorized scripts and resources in the application.

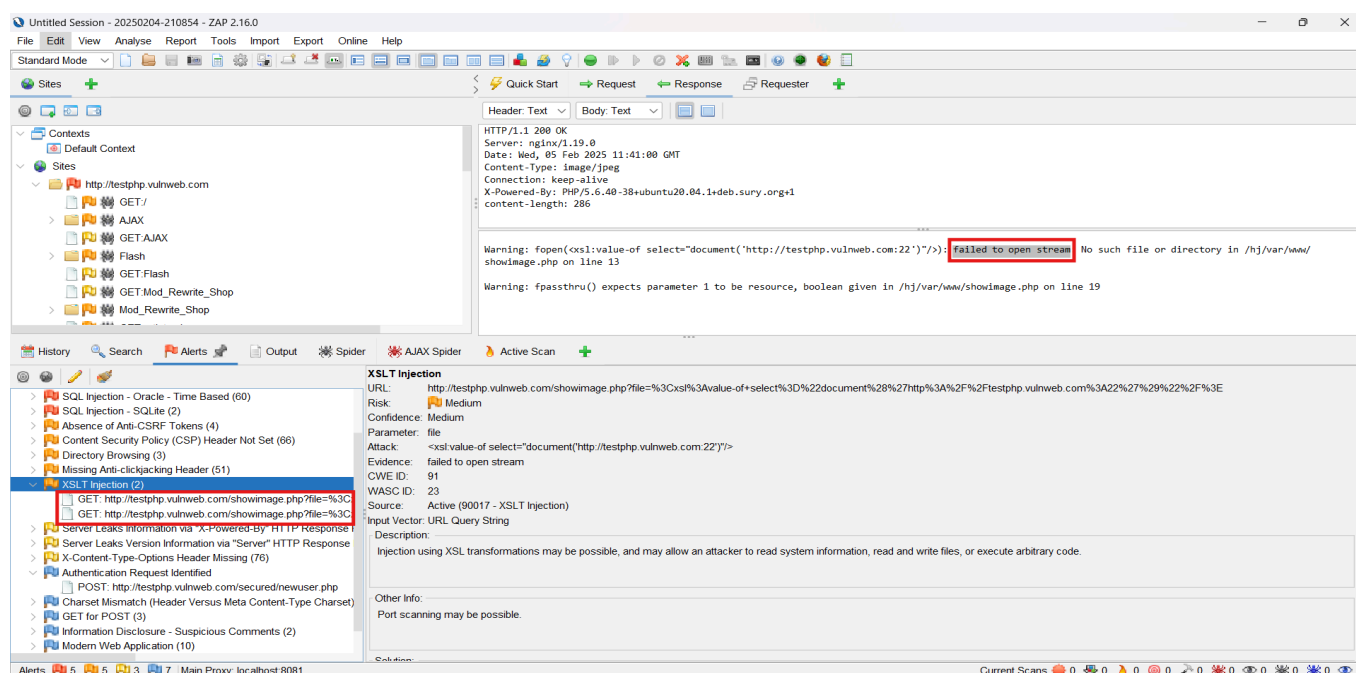
5. Error Handling:

- Suppress detailed error messages in production environments.
- Provide user-friendly error messages while logging full error details server-side for diagnostics.

6. Access Control:

- Limit access to file system functions and directories that the web application can access.

Evidence:



2.14 Modern Web Application Identified

Parameter	Description
Severity	Informational
Impact	Medium
Risk Score	3.0
Affected URL	http://testphp.vulnweb.com/AJAX/index.php
Threat	<p>Potential issues include:</p> <ul style="list-style-type: none">• The Impact of X-Content-Type-Options Header Missing:• Difficulties in automated crawling• Incomplete vulnerability scanning due to non-traditional navigation• Misinterpretation of page structure during manual testing

The application appears to be a modern web application that relies on JavaScript-driven functionality, which means traditional crawling methods may not be sufficient. Specifically, links on the page lack traditional href attributes and use JavaScript functions (e.g., `onclick="loadSomething('titles.php')"`) for navigation. This suggests the application is using AJAX or dynamic content loading, which might require specialized tools for proper exploration.

The presence of links with onclick JavaScript events rather than standard href attributes indicates that the application is using modern JavaScript-based techniques such as AJAX for dynamic content loading. This can make it more challenging for traditional web application scanners to perform full exploration. While this is not inherently a vulnerability, it does indicate that automated testing tools might need to be configured differently to properly explore the application.

Resolution of Vulnerability:

As this is an informational alert, no changes are required. However, consider the following recommendations:

1. Use of AJAX Spider Tools:

When performing security scans or tests, use specialized tools such as the Ajax Spider to ensure that dynamic content and JavaScript-driven interactions are properly explored.

2. Manual Testing Awareness:

Manual testers should be aware of the application's use of AJAX and JavaScript for navigation and data loading. Ensure that all parts of the application are tested, including dynamically loaded content.

Evidence:

The screenshot displays the Burp Suite interface with the following components:

- Left Panel (Sites):** A tree view showing the scanned site `http://testphp.vulnweb.com` with sub-items for `GET /`, `GET AJAX`, `Flash`, `GET.Flash`, `GET.Mod_Rewrite_Shop`, and `Mod_Rewrite_Shop`.
- Top Panel (Request/Response):** Shows an HTTP 1.1 200 OK response from `nginx/1.19.8`. The body contains HTML code with a `<tr class="bordered">` element. Inside, there are three `<a>` tags: `artists`, `categories`, and ``. The third tag is highlighted with a red box.
- Bottom Panel (Alerts):** A list of alerts under the "Modern Web Application" category. The first alert is highlighted with a red box and reads: `GET: http://testphp.vulnweb.com/AJAX/`. Other alerts include `GET: http://testphp.vulnweb.com/AJAX/index.php`, `GET: http://testphp.vulnweb.com/artists.php`, `GET: http://testphp.vulnweb.com/artists.php?artist=1`, `GET: http://testphp.vulnweb.com/artists.php?artist=2`, `GET: http://testphp.vulnweb.com/artists.php?artist=3`, `GET: http://testphp.vulnweb.com/listproducts.php?artist=1`, `GET: http://testphp.vulnweb.com/listproducts.php?artist=2`, `GET: http://testphp.vulnweb.com/listproducts.php?cat=1`, and `GET: http://testphp.vulnweb.com/listproducts.php?cat=2`.
- Right Panel (Details):** Provides details for the selected alert, including the URL `http://testphp.vulnweb.com/AJAX/index.php`, risk level `Informational`, confidence `Medium`, and a description stating: "The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one."

2.15 Authentication Request Identified

Parameter	Description
Severity	Informational
Impact	Low
Risk Score	2.1
Affected URL	http://testphp.vulnweb.com/secured/newuser.php
Threat	Potential issues include: <ol style="list-style-type: none">1. Information Exposure2. Insecure Communication3. Lack of Input Validation

The system identified an authentication request where sensitive user information (username, password, email, etc.) was submitted over the network. Although this is informational, such requests can reveal security gaps if transmitted insecurely or stored improperly.

Resolution and Security Recommendations

1. Enforce HTTPS:

- Ensure that all authentication requests and sensitive information exchanges are encrypted using HTTPS to prevent exposure to attackers during transmission.

2. Mask Sensitive Data:

- Avoid echoing sensitive information (password, credit card number) back to users in responses. Only display minimal, necessary details in confirmation pages.

3. Implement Secure Storage for Credentials:

- Ensure that user passwords are stored securely using robust hashing algorithms such as Argon2, bcrypt, or PBKDF2.

4. Set Secure HTTP Headers:

- Add headers like Strict-Transport-Security, Content-Security-Policy, and X-Content-Type-Options to improve the security of authentication pages.

5. Utilize Parameterized Queries:

- To prevent any injection attacks during user creation, ensure that all user inputs are passed to the database using parameterized queries.

6. Apply Account Creation Best Practices:

- Implement CAPTCHA to prevent automated account creation.
- Validate input fields to prevent malicious entries, ensuring strong password policies.

Evidence:

Request body contained sensitive parameters, such as `uname`, `upass`, and `uemail`.

The response displayed user input in plain text on a confirmation page.

The screenshot displays the ZAP (Zed Attack Proxy) interface. The top pane shows a POST request to `http://testphp.vulnweb.com/secured/newuser.php`. The request body contains the following parameters: `uname-ZAP%upass-ZAP%upass2-ZAP%uname-ZAP%ucc-ZAP%uemail-ZAP%uphone-ZAP%uaddress-signup-signup`. The bottom pane shows a list of alerts, with 'Authentication Request Identified' selected. The alert details include the URL, risk level (Informational), confidence (Low), parameter (uemail), attack type (upass), and a description of the authentication request.

2.16 Charset Mismatch (Header Versus Meta Content-Type Charset)

Parameter	Description
Severity	Informational
Impact	Low
Risk Score	2.1
Affected URL	http://testphp.vulnweb.com
Threat	Potential issues include: <ul style="list-style-type: none">1. Content Sniffing Vulnerability2. Cross-Site Scripting (XSS) Risk3. Data Corruption

When a charset mismatch occurs between the HTTP Content-Type header and the charset declared in HTML or XML meta tags, browsers may resort to content-sniffing to determine the correct encoding. Attackers can exploit this behavior by injecting content using non-standard encodings like UTF-7, leading to potential script execution or content manipulation.

Resolution Measures:

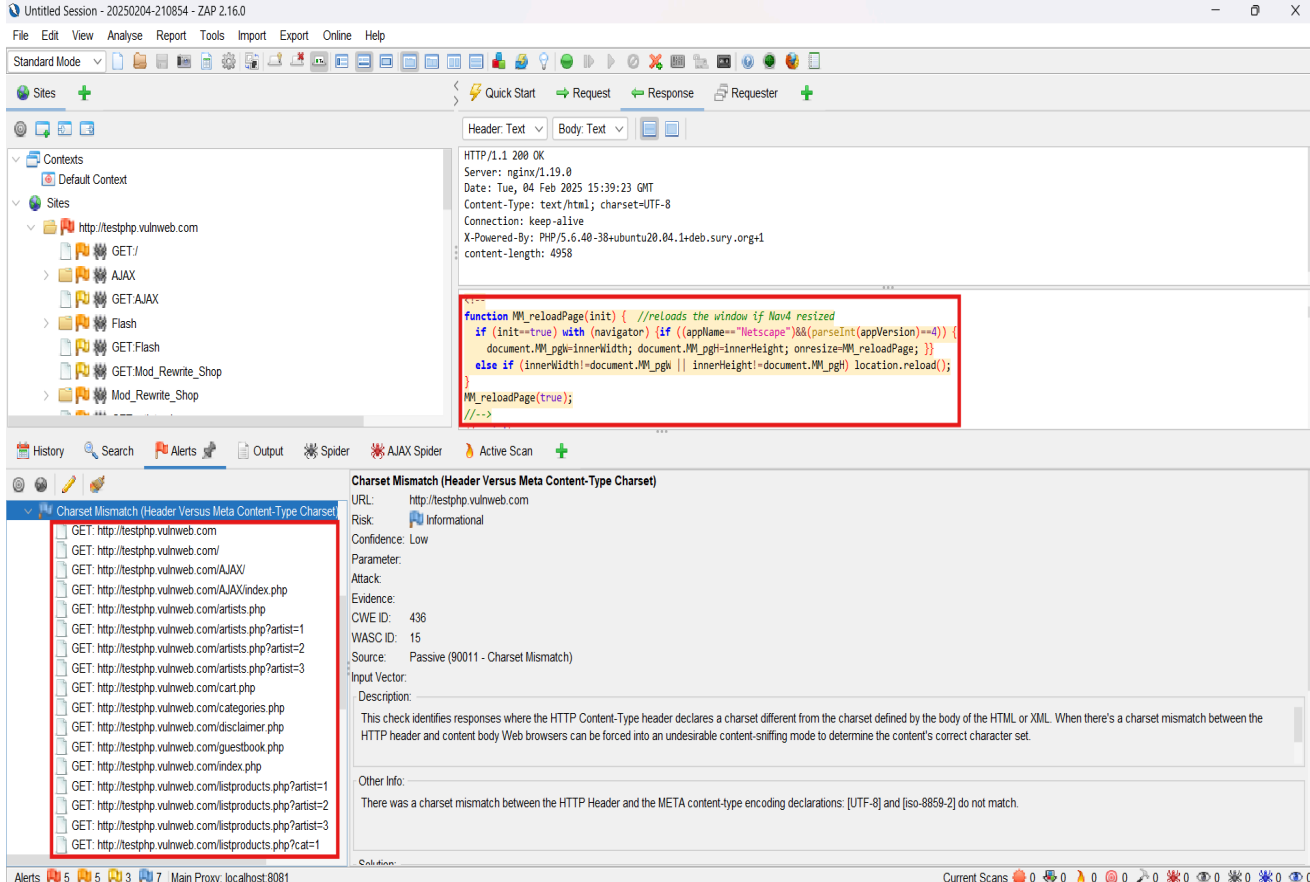
1. Consistent Charset Declaration:

Set a consistent character set (charset=UTF-8) in both HTTP headers and HTML meta tags to avoid charset mismatches.

2. Server Configuration Update:

Configure the web server to enforce charset=UTF-8 for all text-based content responses.

© 2006 The Authors



2.17 GET for POST

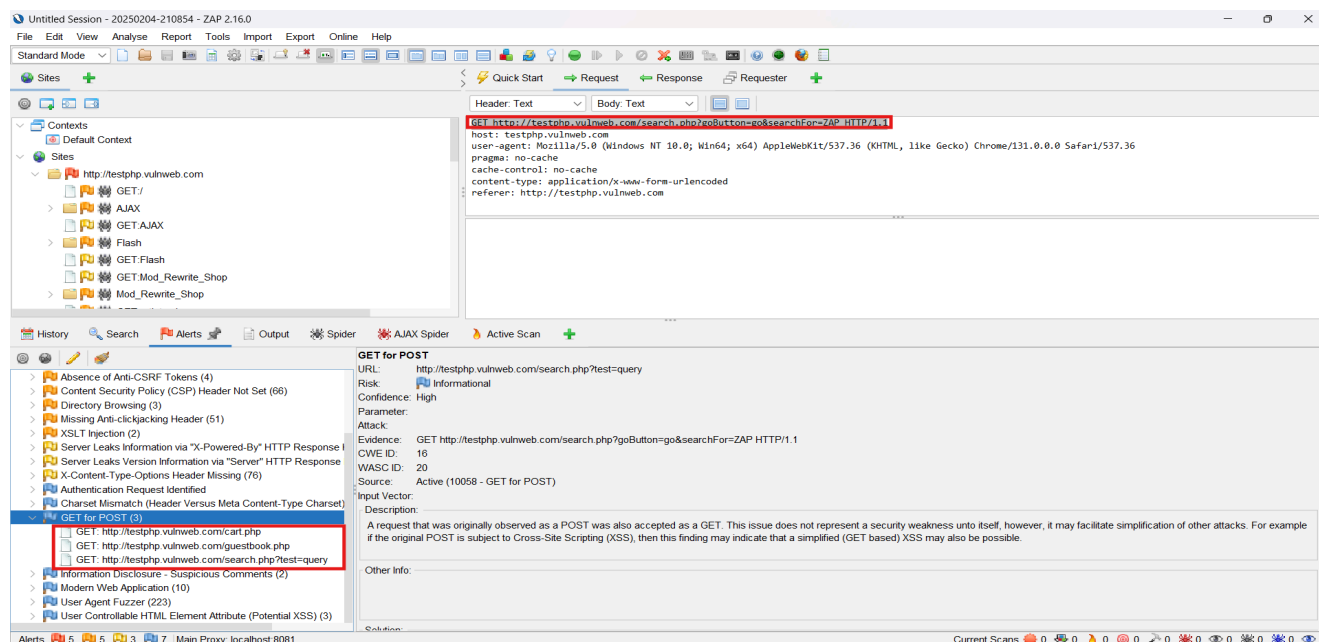
Parameter	Description
Severity	Informational
Impact	High
Risk Score	3.0
Affected URL	http://testphp.vulnweb.com/search.php?goButton=go&searchFor=ZAP
Threat	<p>Potential issues include:</p> <ol style="list-style-type: none">1. Cross-Site Scripting (XSS) Facilitation2. Sensitive Information Exposure3. Cross-Site Request Forgery (CSRF) Risks4. Inconsistent API Design

The application accepts requests originally intended for POST as GET requests. Although this does not inherently constitute a security vulnerability, it can simplify attacks by enabling easier exploitation scenarios such as GET-based Cross-Site Scripting (XSS). Additionally, sensitive data might inadvertently be exposed in the URL.

Resolution Measures:

1. **Strict Method Enforcement:** Configure the server to strictly enforce HTTP method validation. Only accept POST for sensitive operations.
2. **Input Sanitization:** Ensure all user input is properly validated and sanitized regardless of the request method.
3. **Redirection on Invalid Methods:** Redirect users attempting GET requests on POST-only routes to appropriate error pages or notifications.
4. **Security Headers:** Include security headers (such as Content-Security-Policy) to mitigate content injection risks.
5. **Log Monitoring:** Continuously monitor logs for unauthorized GET requests on POST endpoints as a potential indicator of attack reconnaissance.

Evidence:



2.18 Information Disclosure - Suspicious Comments

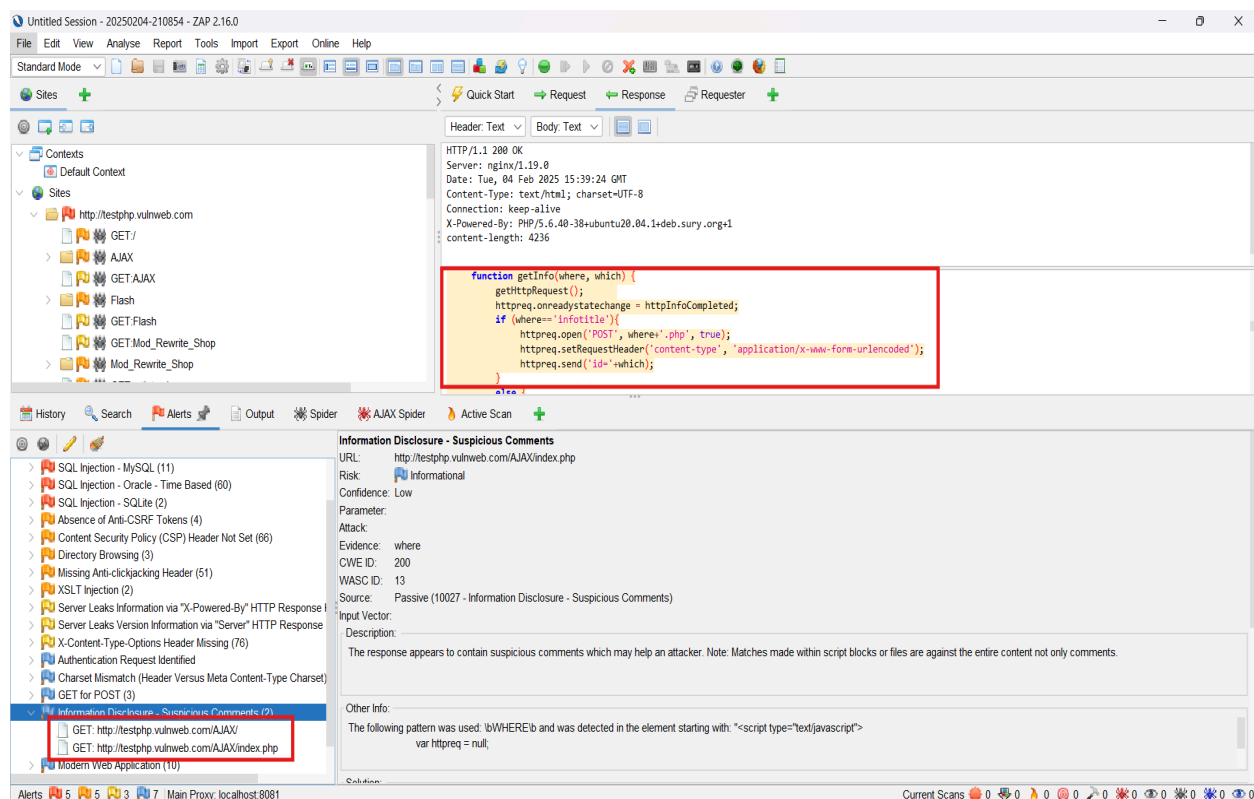
Parameter	Description
Severity	Informational
Impact	Low
Risk Score	3.0
Affected URL	http://testphp.vulnweb.com/AJAX/index.php
Threat	Potential issues include: <ol style="list-style-type: none">1. Sensitive Information Exposure2. Ease of Reconnaissance for Attackers3. Circumventing Security Controls

The response contains comments or snippets of code with potentially sensitive information. In this case, the term "WHERE" was detected, which suggests exposure of SQL-related information or query logic. If such information is visible to users, it can aid attackers in identifying and exploiting vulnerabilities.

Page 10 of 10

- **Comment Cleanup:** Review and remove all comments containing sensitive or implementation-specific details from code before deploying to production.
- **Code Obfuscation:** Minify or obfuscate JavaScript files to prevent exposure of code logic and comments.
- **Secure Debugging Practices:** Ensure that debugging information is never deployed to production environments.
- **Security Reviews:** Conduct periodic code reviews to identify and remove any sensitive comments or debug information.
- **Access Controls:** Restrict access to internal APIs and sensitive endpoints to authenticated and authorized users only.

Evidence:



2.19 User Agent Fuzzer

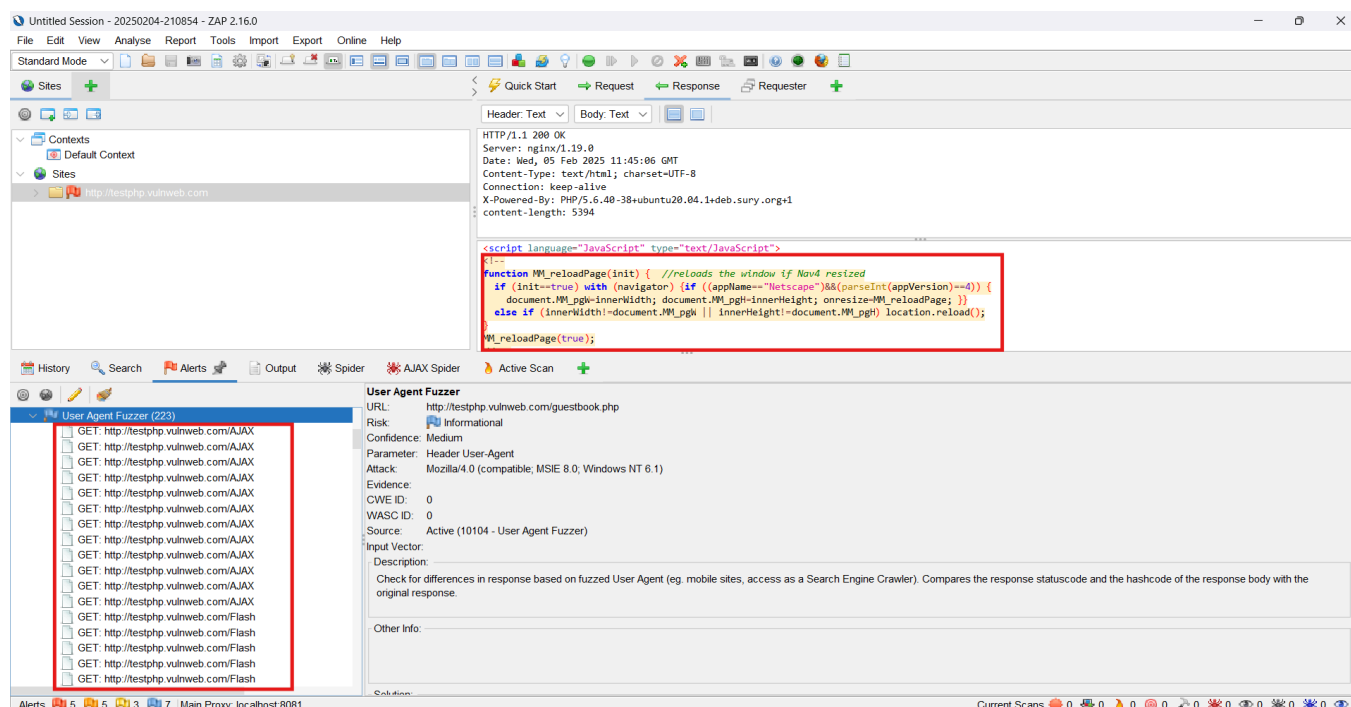
Parameter	Description
Severity	Informational
Impact	High
Risk Score	3.0
Affected URL	http://testphp.vulnweb.com/guestbook.php
Threat	Potential issues include: <ul style="list-style-type: none">1. Fingerprinting2. Access Control Bypass3. Information Disclosure

A fuzzed User-Agent was detected, indicating that the application may respond differently based on the User-Agent value provided in the request header. If not handled securely, this behavior could potentially expose unintended content, enable fingerprinting, or provide attackers with information on restricted access areas.

Resolution Measures:

1. **Consistent Response Handling:** Ensure that application responses remain consistent regardless of the User-Agent provided unless there is a legitimate business need.
2. **Access Control:** Verify that sensitive resources are protected and cannot be accessed simply by altering the User-Agent header.
3. **Content Filtering:** Regularly monitor and filter responses to ensure no unintentional information is exposed to specific user agents.
4. **User-Agent Header Validation:** Restrict or validate User-Agent values to accepted and known legitimate user agents.
5. **Security Testing:** Conduct further fuzzing tests to identify and rectify any unexpected application responses.

Evidence:



2.20 User Controllable HTML Element Attribute

Parameter	Description
Severity	Informational
Impact	High
Risk Score	3.0
Affected URL	http://testphp.vulnweb.com/guestbook.php
Threat	Potential issues include: <ol style="list-style-type: none"> 1. XSS Exploitation 2. Bypass of HTML Encoding 3. Session Hijacking and Data Theft

The application includes user input in an HTML attribute (value attribute of the input tag) without proper validation or sanitization. If special characters or malicious payloads are injected, this may allow attackers to manipulate webpage content or inject scripts.

3. APPENDIX

APPENDIX A - MEASUREMENT SCALE

High: Vulnerability introduces significant technical risk to the system that is not contingent on other issues being present to exploit.

Medium: Vulnerability does not in isolation lead directly to the exposure of sensitive business data. However, it can be leveraged in conjunction with another issue to expose business risk.

Low: Vulnerability may result in limited risk or require the presence of multiple additional vulnerabilities to become exploitable.

Informational: Finding does not have a direct security impact but represents an opportunity to add an additional layer of security, is a deviation from best practices, or is a security-relevant observation that may lead to exploitable vulnerabilities in the future.

APPENDIX B - RISK SCORE

SCORE RANGE	RISK LEVEL	DESCRIPTION
7-10	HIGH	Critical vulnerability, immediate action required
4-6	MEDIUM	Moderate risk, needs attention within a reasonable timeframe
1-3	LOW	Low risk, minimal impact, can be addressed later

