University of Roehampton London

Penetration Test Report

Maisie's Application

172.16.252.239

Security Testing

Sawsan Mounes

Student ID: MOU23620003

27/03/2028

Table of Contents

Overview

Background

This penetration testing engagement was conducted independently by myself as part of my studies at the University of Roehampton. The target was the "Maisie" web application — a PHP-based platform supported by a MySQL database, developed to showcase Maisie the Cat and allow fans to interact through messages and social features. The assessment was undertaken to evaluate the application’s resilience against common web vulnerabilities, particularly those outlined in the OWASP Top 10. Given the nature of the platform and its handling of potentially sensitive data such as user credentials and session management, a proactive security audit was deemed necessary prior to any live deployment.

The test focused on identifying misconfigurations, coding flaws, and exposure points that could be exploited by an attacker. A range of vulnerabilities were identified — from critical issues such as SQL Injection and Cross-Site Scripting (XSS), to weak session management, insecure HTTP usage, and insufficient input sanitisation

Objective

The primary objective of this engagement was to assess the security posture of the Maisie web application in order to:

- Identify and classify potential security vulnerabilities.
- Evaluate the application's susceptibility to critical issues such as SQL Injection, XSS, brute-force attacks, and insecure session handling.
- Attempt privilege escalation to access the admin account (Maisie).
- Validate the application’s resilience against the OWASP Top 10 vulnerabilities.
- Provide remediation advice and risk ratings for each identified issue.

The test aimed to simulate real-world attack scenarios without compromising the availability or integrity of the system.

Scope

The scope of this assessment was restricted to the web application hosted at IP 172.16.252.239 on port 80. This included:

- User-facing features (registration, login, posting messages).
- File upload functionality.
- Session handling and account management.
- Server-side technologies (PHP/MySQL stack).

Excluded from scope:

- All other ports and services beyond port 80.
- Automatic vulnerability scanners automated tools (e.g., Burp Suite Intruder, OWASP ZAP Active Scan). (OWASP, 2023)
- Full exploitation of critical vulnerabilities (e.g., data deletion, persistent XSS attacks).
- The admin account's actual login credentials—brute-force against this account was not permitted without explicit authorization.

These exclusions were defined to prevent unintended disruptions in the test environment and ensure the ethical boundaries of student-conducted testing were respected.

Constraints and Limitation

- **No Automated Scanning:** Tools like Burp Suite Intruder or OWASP ZAP active scans were not allowed, limiting discovery of low-level or obscure vulnerabilities.
- **Local Access Only:** The app was hosted on an internal network (DB117), preventing external or remote testing scenarios.
- **Incomplete Features:** Certain functionalities such as email verification and password recovery were not active, reducing the realism of user workflows.

- **IDOR Untested:** Role-based access controls were limited, making it difficult to test for Insecure Direct Object Reference (IDOR) vulnerabilities.
- **Admin Areas Found, Not Accessed:** Paths to sensitive admin pages (e.g., /admin.php, /maisies_stash.php) were discovered, but no access attempts were made, in line with scope restrictions.
- **No Destructive Testing:** SQL Injection was confirmed and could have allowed table deletion or data tampering, but no such actions were taken due to ethical and scope limitations.

Methodology

Methodology- Black Box testing

In this approach, the tester had no prior knowledge of the application's internal code, database structure, or administrative access, simulating the perspective of an external attacker. Testing focused on:

- Identifying input points that could be manipulated.
- Observing system behaviour and responses.
- Attempting to bypass authentication or access restricted features.
- Exploring how the application handles user input and session data.

This methodology allowed the tester to interact with the application from the outside, focusing on vulnerabilities based on visible components, inputs, and outputs, without any insider information about the backend or source code.

Methodology- Manual Testing

Manual testing was used to identify vulnerabilities by directly interacting with the application without automated tools. This included:

- Testing form inputs and URLs for SQL Injection and XSS.
- Observing system responses to identify errors or leaks.
- Inspecting and manipulating cookies for session-related flaws.
- Brute-forcing the /maisies_stash.php login endpoint (as permitted).
- Exploring hidden files and directories (e.g., via robots.txt and Gobuster).
- Reviewing authentication, password strength, and account features manually.

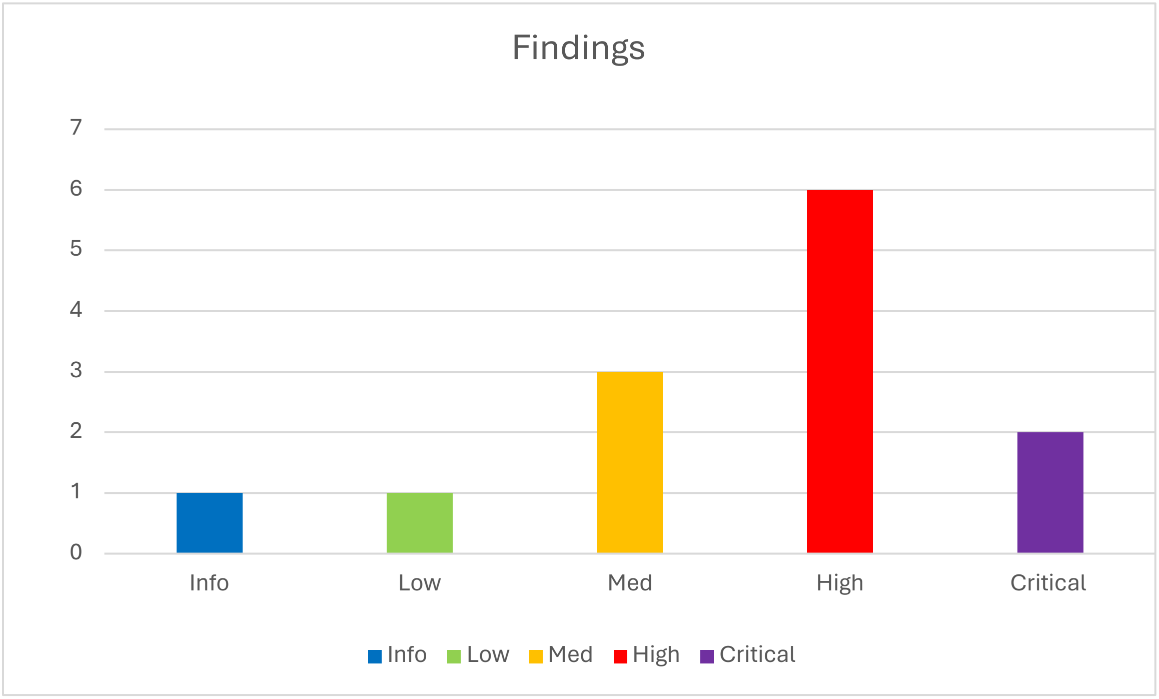
CVSS

CVSS Version 3.1: The **Common Vulnerability Scoring System (CVSS)** version 3.1 is used to rate the severity of vulnerabilities discovered. It assigns scores based on various factors, including exploitability, impact, and the affected component's configuration. A **CVSS string** typically includes factors such as Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), User Interaction (UI), Scope (S), and the impact on Confidentiality (C), Integrity (I), and Availability (A). The higher the score, the more critical the vulnerability. (National Institute of Standards and Technology, n.d.)

Severity	Severity Score Range
None(info)	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Executively Summary

Following a comprehensive security assessment of the Maisie web application hosted at 172.16.252.239, multiple vulnerabilities ranging from critical to informational were identified. These findings represent significant risks to user data confidentiality, application integrity, and overall system security.



Summary of Findings:

- **Critical Risks (2):** SQL Injection and Cross-Site Scripting (XSS) vulnerabilities were found. SQL Injection allows attackers to manipulate database queries, potentially gaining access to sensitive user data such as usernames, emails, and passwords. XSS vulnerabilities enable attackers to execute malicious scripts in users' browsers, risking session hijacking and phishing attacks.
- **High Risks (6):** Issues include insecure HTTP usage exposing sensitive data to interception, unlimited login attempts leaving the application vulnerable to brute-force attacks, Apache version disclosure, password hashes displayed in URLs, brute force susceptibility at login endpoints, and cookie poisoning enabling session hijacking.

- **Medium Risks (3):** Weak password policy allowing insecure passwords, absence of email verification and password reset capabilities, and PHP version disclosure revealing detailed server configuration.
- **Low Risk (1):** Hidden directories accessible publicly, potentially exposing sensitive files or data.
- **Informational (1):** Absence of clearly defined Terms and Conditions, potentially exposing the application to legal risks.

Technical Summary

The Maisie web application at 172.16.252.239 is affected by multiple high-risk vulnerabilities that, when chained, can lead to full application compromise. The most critical issues identified are SQL Injection (SQLi) and Reflected Cross-Site Scripting (XSS)**,** both confirmed during testing and exploitable in tandem.

SQL Injection was observed due to poor input sanitization. UNION-based payloads allowed for enumeration of the database schema, extraction of table and column names, and retrieval of actual data, including usernames, emails, and password hashes. One captured hash was in MD5 format, and ethically validated that it could be reversed using public cracking tools, confirming the real-world exploitability of leaked hashes.

XSS vulnerabilities were present across several endpoints. Reflected payloads enabled JavaScript injection using techniques like onerror and eval(String.fromCharCode(...)). These could be used to steal session cookies or redirect users to phishing sites. The lack of HttpOnly and Secure flags on cookies, along with no session binding or regeneration, allows for effective session hijacking (cookie poisoning)**.**

These two issues, when chained, form a viable attack path:

- SQLi reveals valid usernames and weak, crackable hashes
- Brute force or hash cracking (MD5) yields credentials
- Lack of CAPTCHA or lockout allows automated login attempts
- XSS enables attackers to hijack sessions or escalate access
- Combined, these allow an attacker to bypass authentication and assume user or admin roles

Additional weaknesses further expose the application:

- **No HTTPS**: Login credentials and session cookies can be intercepted over the network
- **Weak password policy**: Accepts passwords like “1234” and “test” without validation

- **No email verification or password reset feature**: Allows fake account creation with no recovery path
- **Version disclosure**: PHP and Apache version info is exposed in headers and public files like phpinfo.php
- **Sensitive data in URLs**: Password hashes appear in GET requests due to SQL errors being reflected in the browser's address bar

Session handling is also poorly implemented: sessions are not tied to user-specific attributes like IP or user-agent, and session IDs are not regenerated upon login or privilege escalation—leaving the door open to session fixation and reuse.

In conclusion, the application is susceptible to chained exploits that can lead to data theft, session takeover, and full control of backend systems. The absence of basic security controls significantly increases the likelihood of a successful compromise. Urgent remediation of input validation, authentication mechanisms, session security, and transport-layer protection is required to reduce the attack surface and restore a secure environment.

Technical Findings Details

Critical Risk Findings

Finding Title Cross-site scripting (XSS)

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:N	Severity:	critical

Background

Cross-site Scripting (XSS) is a vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. When a user visits an affected page, the browser executes the script as if it came from a trusted source. Since the browser cannot tell that the script is untrusted, it may grant the script access to sensitive information such as cookies, session tokens, or user credentials. This can allow the attacker to impersonate users, steal data, or perform unauthorized actions. XSS vulnerabilities are typically classified into two types: reflected XSS, where the malicious script is included in a request and reflected back by the server, and stored XSS, where the script is permanently stored on the website (e.g., in a database) and served to users. Both forms pose serious security risks and can lead to account compromise, data theft, or manipulation of website content.

Technical Details

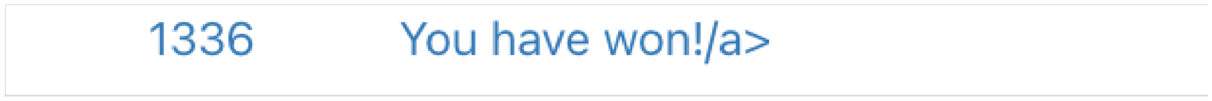
During testing, we discovered multiple instances where user-supplied input was not properly filtered or encoded before being displayed on the page. This resulted in reflected XSS vulnerabilities — meaning the script is executed immediately in the user’s browser via the URL, without being stored on the server.

These issues arise because certain characters such as "<, >, and " are not properly handled by the application, allowing attackers to inject HTML or JavaScript code into web pages. Below are examples of the payloads tested and their impact:

1. JavaScript Redirection (Stored XSS)

<a href="#" onclick="window.location = 'Have I Been Pwned: Check if your email has been compromised in a data breach'; return false;">You have won!/a>

Description: This payload redirects the user to an external site when clicked. While in this case the URL is a public breach-checking service, this demonstrates that redirection is possible, and could be abused for phishing if not properly restricted.



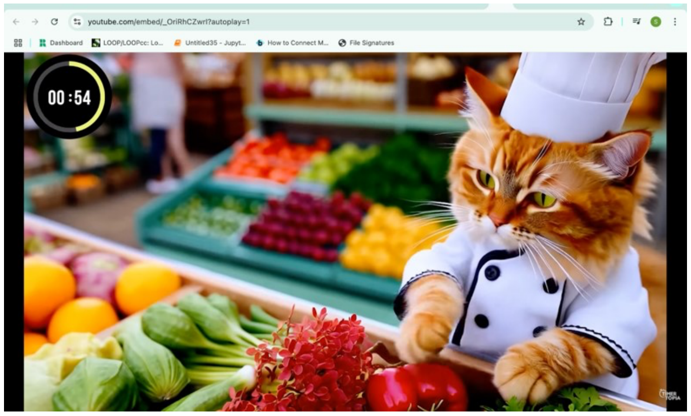
1336 You have won!/a>

2. Cookie Theft via onerror (Reflected XSS)

<img src="x" onerror="alert(document.cookie)">

Description: When an image fails to load, the onerror attribute is triggered and executes JavaScript. This example uses alert() for ethical testing purposes, but in real-world scenarios, attackers could use this to steal session cookies.

3. SVG Payload Using Obfuscated Script (Reflected XSS)



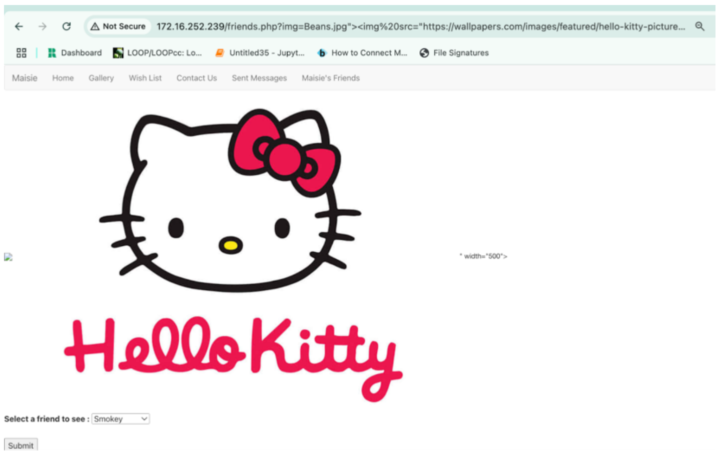
<svg/onload=eval(String.fromCharCode(119,105,110,100,111,119,46,111,112,101,110,40,3
9,104,116,116,112,115,58,47,47,119,119,119,46,121,111,117,116,117,98,101,46,99,111,10
9,47,101,109,98,101,100,47,95,79,114,105,82,104,67,90,119,114,73,63,97,117,116,111,11
2,108,97,121,61,49,39,44,39,95,98,108,97,110,107,39,44,39,119,105,100,116,104,61,56,48
,48,44,104,101,105,103,104,116,61,52,53,48,39,41))>
<svg/onload=eval(String.fromCharCode(119,105,110,100,111,119,46,111,112,101,110,40,3
9,104,116,116,112,115,58,47,47,98,117,114,110,108,101,121,46,103,111,118,46,117,107,4

```
7,119,112,45,99,111,110,116,101,110,116,47,117,112,108,111,97,100,115,47,50,48,50,51,
47,49,49,47,120,108,45,98,117,108,108,121,45,97,109,101,114,105,99,97,110,45,98,117,1
08,108,121,45,49,50,52,48,120,54,57,56,46,106,112,103,39,44,39,95,98,108,97,110,107,39
,44,39,119,105,100,116,104,61,56,48,48,44,104,101,105,103,104,116,61,54,48,48,39,41))>
```

Description: This payload uses JavaScript's `String.fromCharCode()` to obfuscate the script and bypass basic filtering. When rendered, it executes code that opens a new browser tab or performs another action. While this test version redirected to a nonmalicious resource, the technique demonstrates how scripts can run unnoticed.

Note: This payload is not "hashed" — it is **encoded** using character codes to hide the script. This is a common evasion technique used to bypass weak filtering systems.

4. Image Injection via URL Parameter (Reflected XSS)



Not Secure 172.16.252.239/friends.php?img=Beans.jpg"><img%20src="https://wallpapers.com/images/featured/hello-kitty-picture...

Dashboard | LOOP/LOOPcc: Lo... | Untitled35 - Jupyt... | How to Connect M... | File Signatures

Maisie Home Gallery Wish List Contact Us Sent Messages Maisie's Friends

" width="500">

Select a friend to see : Smokey

Submit

http://172.16.252.239/friends.php?img=Beans.jpg"><img src="https://wallpapers.com/images/featured/hello-kitty.jpg"> **Description**: This payload injects an image using the `img` parameter in the URL. Since the input is not sanitized, attackers can insert HTML tags directly into the page. While this example simply loads an external image, it demonstrates that scripts or malicious content could be inserted the same way.

Remediation

To address the identified Cross-Site Scripting (XSS) vulnerabilities, we recommend the following actions:

1. **Input Validation & Output Encoding**
   Ensure all user-supplied input is properly validated and encoded before being rendered in the browser. Use frameworks or libraries that handle HTML, JavaScript, and URL encoding automatically to prevent injection.
2. **Use Security Headers**
   Implement HTTP security headers such as Content-Security-Policy (CSP), X-XSSProtection, and X-Content-Type-Options to provide an additional layer of protection and reduce the impact of potential XSS attacks.
3. **Sanitize Parameters**
   Apply strict allowlists for parameters like img, and prevent direct HTML/script injection by rejecting or escaping characters such as <, >, ", and ' wherever not required.
4. **Avoid Dangerous Functions**
   Avoid using eval() or similar functions that interpret data as code. They are dangerous and widely exploited in XSS attacks.
5. **Regular Security Testing**
   Conduct regular penetration tests and automated scans to identify and remediate new XSS vectors as your application evolves.
6. **Educate Developers**
   Train your development team on secure coding practices, including how to prevent XSS, use encoding libraries, and safely handle user input.

By following these steps, the risk of XSS exploitation will be significantly reduced, protecting both your application and its users.

Finding Title : SQL Injection

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L	Severity:	**Critical**

Background

SQL Injection is a serious vulnerability that allows an attacker to interfere with the SQL queries an application makes to its database. Exploiting this flaw can enable an attacker to retrieve unauthorized data—such as user account information, credentials, or other sensitive records—that should otherwise remain protected. In more advanced cases, SQL Injection can also be used to modify or delete database content, which may lead to website defacement, data corruption, or full system compromise. This makes it a high-impact vulnerability that requires immediate attention. (PortSwigger, 2024 **Technical Details identifying the Injection Point:**

- **Payload Used:** ' OR 0=0 --
- During initial testing, a basic SQL injection payload was submitted to evaluate the application's handling of user input. The successful execution of this payload confirmed that the application is vulnerable to SQL injection attacks, allowing manipulation of backend SQL queries.

Determining the Number of Columns:

- **Payload Used:** ' OR 0=0 ORDER BY 5 --
- Followed by: ' OR 0=0 UNION SELECT 1, 2, 3, 4, 5 --
- These payloads helped establish that the backend SQL queries return five columns. This confirmed that UNION-based SQL injection is possible, facilitating structured exploitation attempts.

Enumerating Table Names:

- **Payload Used:** ' OR 0=0 UNION SELECT NULL, table_name, NULL, NULL, NULL FROM information_schema.tables WHERE table_schema = 'app' --
- This payload enabled the identification of existing tables within the application's database schema, demonstrating access to sensitive database metadata.

Enumerating Column Names:

- **Payload Used:** ' OR 0=0 UNION SELECT NULL, column_name, NULL, NULL, NULL FROM information_schema.columns WHERE table_name = 'users' --
- Testing confirmed that the users table includes sensitive columns such as username, email, and password, highlighting potential exposure risks.

Extracting Sample Data for Validation:

- **Payload Used:** ' OR 0=0 UNION SELECT NULL, username, email, password, NULL FROM users --
- A limited set of non-sensitive data was retrieved to confirm the vulnerability's severity. No passwords were decrypted or altered, and data extraction was strictly limited to validation purposes within the approved testing scope.

Database Version Disclosure:

- **Payload Used:** ' OR 0=0 UNION SELECT NULL, version(), NULL, NULL, NULL --
- This query revealed the database version, which, although seemingly minor, can help attackers identify specific exploits tailored to that version. For example the bversion was maria10.4.34 which was created may 16 2024 (MariaDB KnowledgeBase, 2025

Remediation

- **Use Secure Coding Practices**
  Update the website code so it handles user input safely. This means using secure methods to talk to the database that don't allow attackers to insert harmful code.
- **Check and Limit User Input**
  Make sure the website only accepts expected input — for example, if a field asks for a number, it should reject letters or symbols.
- **Limit Database Access**
  The account your website uses to talk to the database should only have access to what it needs — not full control. This reduces the damage if something goes wrong.
- **Hide System Information**
  Turn off detailed error messages that show technical details about your system. These can help attackers understand how your site works.
- **Install Security Tools**
  Use a Web Application Firewall (WAF) to block common attacks before they reach your site.
- **Keep Systems Updated**
  Make sure your database and software are up to date to protect against known vulnerabilities.
- **Test Regularly**
  Schedule regular security checks to make sure no new vulnerabilities appear as the website changes.

High Risk Findings

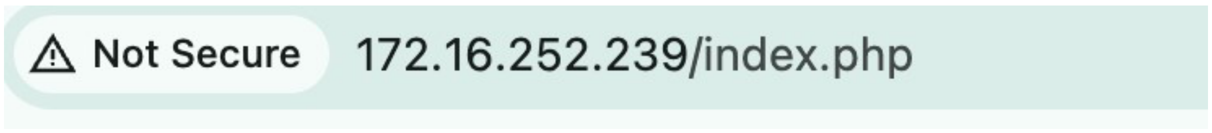
Finding Title: HTTP

Host:	172.16.252.239	Port:	80
CVSSv 3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	Severity:	High

Background

HTTP is an insecure protocol because it sends data in plain text rather than using encryption. As a result, attackers can intercept and potentially modify traffic over the network through techniques like man-in-the-middle or spoofing. This lack of encryption leaves sensitive information—such as credentials or other data—vulnerable to theft or tampering.

Technical Details



Before logging in, I noticed that the browser displayed a "Not Secure" warning next to the URL (http://172.16.252.239/index.php). This indicates that the connection is not encrypted, meaning any information you enter—such as passwords, credit card numbers, or personal details—could potentially be intercepted by an unauthorized third party.

Remediation

- Recommendation: Move from HTTP to HTTPS
- Obtain & Install an SSL/TLS Certificate ○ Purchase from a trusted provider (e.g., GoDaddy, Let's Encrypt), then install it on your server.
- Update All Site URLs to HTTPS ○ Change internal links and media references to https://.

○ Redirect old http:// links to the new https:// version.

- Verify HTTPS Configuration ○ Check that the padlock icon or "Secure" message appears in the address bar.

○ Use tools like SSL Labs to confirm proper encryption settings.

- This encryption helps protect visitors’ data, remove the “Not Secure” warning, and boosts trust in your website.

Finding Title: Login Attempt Limitation

Host:	172.16.252.239	Port:	80
CVSSv 3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	Severity:	**HIGH**

Background

A login lockout mechanism is essential for preventing brute-force attacks. This security control limits the number of incorrect login attempts and temporarily locks the account after reaching a threshold (e.g., 5 failed attempts). Without this safeguard, attackers can use automated tools to guess passwords and gain unauthorized access.

Technical Details

Testing revealed that the application allows unlimited login attempts without any lockout or throttling. There were no security measures such as account suspension, delay, or alerts triggered after repeated failures. This makes the login endpoint highly susceptible to brute-force and credential stuffing attacks.

Remediation

- **Limit Login Attempts:** Lock accounts temporarily after a set number of failures.
- **Add CAPTCHA:** Introduce CAPTCHA after multiple failed attempts.
- **Use Multi-Factor Authentication (MFA):** Add an extra layer of login security to protect against brute-force and credential stuffing.

Finding Title: Apchae version number

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	Severity:	HIGH

Background

This vulnerability exposes sensitive server information (e.g., software versions, internal IP addresses) through error messages. Attackers can use this data to identify potential exploits or map out the network, increasing the likelihood of successful attacks

Technical Details

Upon investigating 172.16.252.239, it was found that triggering an error redirects users to a page that reveals the Apache type and version **Apache/2.4.57**, the operating system (including Debian), and the internal IP address. This vulnerability discloses sensitive information that attackers can leverage to identify and exploit known weaknesses. Furthermore, the server's behavior of accepting arbitrary paths instead of returning a proper 404 error further exposes its configuration details. (Apache, 2019)

Remediation

Disable Server Info: Turn off ServerSignature and set ServerTokens to Prod to hide Apache version and OS details.

Use Custom Error Pages: Replace default error messages with generic ones that don’t expose internal information.

Hide Internal IPs: Ensure internal addresses aren’t shown in error responses.

Clean Up Headers

Finding Title : Password Hash Displayed in URL

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N	Severity:	**HIGH**

Background

Sensitive information such as hashed passwords should never be exposed in URLs. URLs can be stored in browser history, server logs, and may be intercepted over unsecured connections, leading to data leakage.

Technical Details

During testing of the login mechanism at http://172.16.252.239, we observed that submitting a valid username with an incorrect password resulted in an SQL query being reflected in the URL. This query included a password hash—a critical exposure that violates secure coding practices.

Upon inspection, the hash format appeared consistent with MD5**,** an outdated and insecure hashing algorithm. For ethical validation purposes only, we confirmed that it was possible to reverse this hash using freely available tools and databases. While we did **not** proceed further with exploiting the application or accessing additional data, the test demonstrated how easily an attacker could crack weak hashes under real-world conditions.

Note for the Client: This test was conducted under strict ethical guidelines. No user data was accessed or misused, and only a single hash was tested to validate the risk. The reference to tools like **CrackStation** is for awareness of the threat landscape.

The presence of a hash in the URL introduces two major risks:

- It reveals backend query logic, which can support SQL injection attempts.
- It discloses a weak, crackable password hash to the browser and potentially to third parties through logs and cache.

(Author, 2024)

Remediation

- **Use POST Instead of GET:**
- Ensure login forms submit credentials via POST requests, not GET, to prevent credentials or hashes from appearing in the URL.
- **Remove Sensitive Data from URLs:**
- Avoid reflecting SQL queries or hashed values in the URL or front-end responses.
- **Sanitize Error Handling:**
- Ensure backend errors and internal logic (e.g., SQL statements) are never exposed to users. Use generic error messages.
- **Secure Password Handling:**
- Ensure strong hashing algorithms are used (e.g., bcrypt, Argon2), with proper salting.
- **Review and Harden Authentication Logic:**
- Reassess how login failures are processed and logged to ensure no sensitive information is leaked.

Finding Title Cookie poisoning

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	Severity:	**High**

Background

Cookie poisoning**,** also referred to as session hijacking, is an attack method where an attacker modifies or forges a legitimate cookie to gain unauthorized access. By tampering with session-related cookies, the attacker may be able to steal sensitive data**,** bypass authentication, or manipulate user sessions, potentially compromising the security of the application.

Technical Details

During testing, we identified that the application does not properly secure session cookies, making it vulnerable to session hijacking. While XSS was used as a method to demonstrate how a session cookie could be accessed, the core issue lies in how the session itself is handled by the application.

Once a valid session ID is known, the application does not perform any additional checks to verify the user's identity beyond that cookie. For example, if User A obtains User B's session cookie (via any method), they can simply overwrite their own session ID with that of User B's using browser developer tools, and gain full access to User B's account.

This highlights a critical issue with session management, where lack of proper validation and session binding can lead to unauthorized access through cookie manipulation. (www.f5.com, n.d.)

Remediation

To mitigate the risk of session hijacking through cookie poisoning, we recommend the following:

- **Use HttpOnly and Secure flags on cookies:** This helps prevent client-side scripts from accessing session cookies and ensures cookies are only sent over HTTPS.
- **Regenerate session IDs after login and privilege changes:** This prevents session fixation attacks.
- **Implement session binding:** Link the session to user-specific information like IP address and User-Agent to make session reuse harder.
- **Enforce session expiration and idle timeouts:** Limit how long a session remains active to reduce risk if stolen.
- **Monitor for abnormal session activity:** Implement alerts for unusual session behaviors such as session reuse from different locations.

These steps will significantly reduce the risk of session hijacking and strengthen the overall security of your user sessions.

Finding Title Brute Force /maisies_stash.php

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	Severity:	**High**

Background

A brute force attack is a method used by attackers to gain unauthorized access by systematically guessing login credentials such as usernames and passwords. These attacks typically rely on automation to try a large number of combinations until valid credentials are found. Although not technically complex, brute force attacks remain effective—especially when systems lack proper rate limiting, lockout mechanisms, or strong password policies.This issue becomes especially critical in environments without safeguards like multi-factor authentication (MFA) or CAPTCHA, making it easier for malicious actors to gain access through repeated guessing.

Technical Details

- During the engagement, we simulated a brute force attack against your authentication mechanism to assess its resilience.
- Using a custom Bash script and publicly available username and password lists:
- The script read a list of usernames (user.txt) and passwords (passwords.txt).
- It generated Base64-encoded combinations (username:password) and inserted them into HTTP Authorization headers (Basic Authentication).
- These requests were then sent to the login endpoint (/maisies_stash.php).
- Successful login attempts (those returning HTTP 200 OK) were logged to a file (creds.txt), confirming the ability to authenticate using guessed credentials.
- This demonstrates a critical vulnerability, as the system:
- Does not implement account lockout after failed attempts.
- Lacks rate limiting and CAPTCHA.

- Accepts weak credentials without any password strength validation.
- **Note to Client:** This test was performed ethically and in a controlled manner. We did not attempt to use the discovered credentials beyond validating successful login, and no data was accessed or altered beyond the scope of this assessment.
- (Fortinet, n.d

.

Remediation

- We recommend implementing the following security controls to significantly reduce the risk of brute force attacks:
- **Enable Account Lockout**
  Lock or suspend accounts after a small number of failed login attempts (e.g., 5 tries).
- **Apply Rate Limiting**
  Limit the number of requests from a single IP within a specific timeframe.
- **Introduce CAPTCHA**
  Use CAPTCHA (like Google reCAPTCHA) after a few failed login attempts to stop bots.
- **Enforce Strong Password Policies**
  Require passwords to meet minimum complexity standards and block commonly used passwords.
- **Implement Multi-Factor Authentication (MFA)**
  Add an extra verification layer to protect against unauthorized access, even if passwords are compromised.
- **Log and Monitor Login Attempts**
  Set up alerts for repeated login failures and suspicious login activity to respond quickly.
- **Secure Password Storage**
  Ensure passwords are hashed using strong cryptographic functions (e.g., bcrypt) with proper salting.

Medium Risk Findings

Finding Weak Password Policy

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N	Severity:	**Medium**

Background

One of the most common and overlooked vulnerabilities in web applications is the lack of a proper password policy. Weak or poorly constructed passwords significantly reduce the effectiveness of an authentication system, making it easier for attackers to gain unauthorized access. Users often default to simple and easy-to-remember passwords such as “123456,” “password,” or personal information like names and birthdays. These predictable patterns are highly susceptible to brute-force attacks or credential stuffing, especially if attackers are using automated tools. Without a policy in place to enforce complexity, length, or password expiration, the risk of unauthorized access increases considerably.

Technical Details

During registration on the Maisie website, we were able to create accounts using extremely weak passwords such as "1234" and "test". The application did not enforce any password strength rules such as:

- Minimum character length
- Use of uppercase letters
- Numbers or special symbols

No validation warnings or guidance were provided, and weak credentials were accepted without restriction.

Remediation

To strengthen user authentication and reduce brute-force risks, we recommend the following:

- Require passwords to have a minimum of 8 characters.

- Enforce complexity rules (uppercase, lowercase, numbers, and special characters).
- Prevent the use of common passwords from known breach lists.
- Provide real-time password strength indicators and user guidance during registration.

By implementing these changes, the platform will greatly reduce the risk of unauthorized account access, improve user experience,

Finding Title Lack of Email Verification and Account Management Controls

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N	Severity:	**Medium**

Background

Email verification is a critical step in validating that users control the contact information they provide. Without proper checks, attackers can register accounts with fake or mistyped addresses. Furthermore, users should be able to update or recover account information securely to maintain long-term access and security.

Technical Details

- The application allows users to register without confirming the validity of the submitted email address.
- No verification link or confirmation step is sent.
- Users cannot update or change their email address or password after registration.
- There is no password reset mechanism in place.

This lack of verification and account management controls increases the risk of abandoned, fake, or hijacked accounts, while also limiting the user's ability to recover or protect their own account

Remediation

We recommend the following actions:

- Implement email confirmation during registration.
- Reject invalid or unreachable email addresses.
- Allow users to securely change their email or password after authenticating.
- Introduce a secure password reset feature using time-limited tokens or email verification.

Finding Title PHP Version disclosure

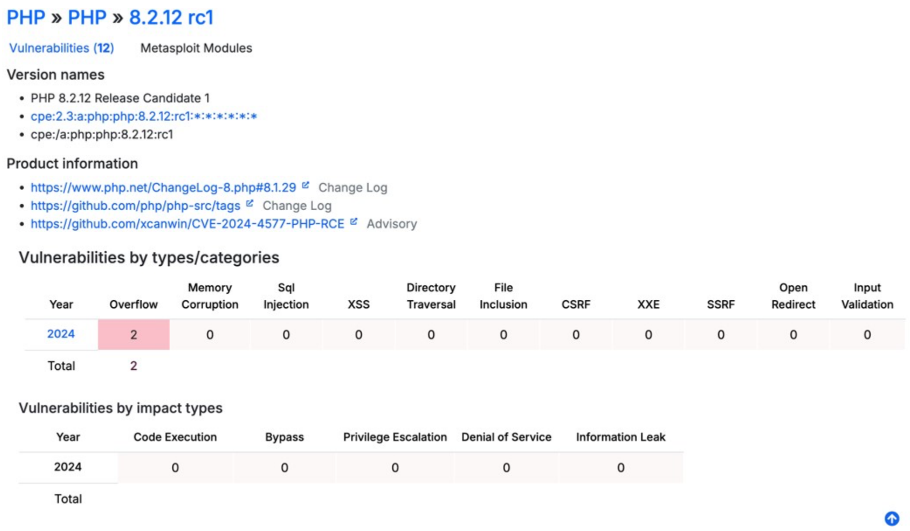
Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Severity:	**Medium**

Background

PHP is a server-side language used to handle key website functions such as user authentication, form handling, and file uploads. When a web application reveals its PHP version (e.g., PHP 8.2.12), it exposes valuable information that attackers can use to identify known vulnerabilities tied to that version. This allows an attacker to search for exploits or weaknesses online and tailor attacks accordingly. Minimizing exposed system details helps reduce the attack surface.

Technical Details

We identified the PHP version in use through two primary methods. First, while intercepting traffic using Burp Suite, we observed HTTP headers that disclosed the server's PHP version and that it was powered by PHP. This kind of information leakage offers attackers valuable reconnaissance data. Secondly, we found that the phpinfo.php file was publicly accessible. This page reveals detailed server configuration information, including the exact PHP version (8.2.12), loaded modules, file upload settings (confirming uploads are enabled), and environment variables. Such detailed exposure could enable an attacker to craft targeted exploits based on the specific server setup. Additionally, the PHP version can be cross-referenced with public vulnerability databases, which may reveal known issues associated with that version. If the system is not properly patched or hardened, this significantly increases the risk of a successful attack. (Cvedetails.com, 2024)



PHP » PHP » 8.2.12 rc1

Vulnerabilities (12) Metasploit Modules

Version names

- PHP 8.2.12 Release Candidate 1
- cpe:2.3:a:php:php:8.2.12:rc1:*:*:*:*:*:*
- cpe:/a:php:php:8.2.12:rc1

Product information

- https://www.php.net/ChangeLog-8.php#8.1.29 Change Log
- https://github.com/php/php-src/tags Change Log
- https://github.com/xcanwin/CVE-2024-4577-PHP-RCE Advisory

Vulnerabilities by types/categories

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2024	2	0	0	0	0	0	0	0	0	0	0
Total	2										

Vulnerabilities by impact types

Year	Code Execution	Bypass	Privilege Escalation	Denial of Service	Information Leak
2024	0	0	0	0	0
Total					

Remediation

To reduce this risk, we recommend the following steps:

- **Restrict Access to phpinfo() Pages:** Remove or restrict access to any pages exposing phpinfo output. These pages should not be accessible in production environments.
- **Hide Server Technology Information:** Disable expose_php in the PHP configuration (php.ini) to prevent the PHP version from being revealed in HTTP headers.
- **Keep PHP Up to Date:** Regularly patch and update your PHP version to the latest stable release to ensure known vulnerabilities are fixed.
- **Harden Server Configuration:** Disable unnecessary PHP modules, restrict file uploads to only what's necessary, and configure appropriate file upload restrictions.
- **Use Web Server Rules:** Implement .htaccess or server-level rules to block public access to internal diagnostic files like phpinfo.php.

Low risk findings

Finding Title :Hidden directories

Host:	172.16.252.239	Port:	80
CVSSv 3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Severity:	**Low**

Background

Sensitive directories or files that are not intended for public access—but are still accessible without proper authentication—pose a significant security risk. These files may contain confidential information such as database credentials, website configuration settings, or even administrative access details. If discovered, an attacker could exploit this to gain deeper access to the system or compromise the entire web application.

Technical Details

During the assessment, we discovered that http://172.16.252.239/robots.txt was publicly accessible and contained references to sensitive files such as /maisies_stash.php and /admin.php. These should not be exposed or advertised, especially without proper access controls.

To simulate a real-world attacker's approach, we used Gobuster**,** an openly available directory enumeration tool that uses wordlists to brute-force and uncover hidden files or directories. This technique is commonly used in reconnaissance phases of attacks.

While /maisies_stash.php was analyzed during the test and found to potentially contain sensitive content, we did **not** access /admin.php as it was explicitly **out of scope** for this engagement. However, its presence and lack of authentication mechanisms suggest it may be accessible to attackers. If this file contains administrative functionality, database access, or configuration details, it could open the door to further exploitation and high-severity vulnerabilities.

Remediation

- **Remove sensitive paths from robots.txt**
  Never list admin or confidential files in robots.txt.
- **Restrict access to sensitive files**
  Protect files like /admin.php and /maisies_stash.php using authentication or server-level access controls.
- **Place sensitive files outside the web root**
  Keep config files or database credentials in secure directories that are not publicly accessible.
- **Regularly audit access logs**
  Monitor for direct access to hidden or sensitive paths.
- **Perform routine internal fuzzing scans**
  Use tools like Gobuster or Dirb to detect exposed directories before attackers do.

Information Risk Findings

Finding Terms and Conditions

Host:	172.16.252.239	Port:	80
CVSSv3.1	AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N	Severity:	**Info**

Background:

Terms and Conditions (T&C) are essential to clearly define the legal boundaries and responsibilities between users and service providers. Without clearly articulated Terms and Conditions, there can be ambiguity regarding acceptable use, privacy, data handling, intellectual property, and dispute resolution.

Technical Details:

During the engagement, it was observed that the Maisie web application does not provide users with any Terms and Conditions at registration or anywhere accessible on the website. The absence of Terms and Conditions means that users may be unaware of their obligations, rights, or how their personal data may be used, potentially exposing the website owner to legal risks or disputes.

Remediation:

It is recommended to clearly publish Terms and Conditions on the website and ensure they are easily accessible to users during registration or login processes. The Terms and Conditions should cover:

- **Create and Publish Terms and Conditions:**
  Draft comprehensive Terms and Conditions (T&C) clearly outlining the legal agreements between users and your organization. Ensure the document covers:

- **Acceptable Use**: Clearly state what actions are permitted and prohibited when using the application.
- **Privacy and Data Handling**: Inform users how personal data is collected, stored, shared, and protected.

- **Intellectual Property**: Clarify ownership and acceptable use of content available through the website.
- **Liability and Warranty Disclaimers**: Limit your organization’s liability in the event of disputes or damages arising from website use.
- **Dispute Resolution**: Define mechanisms for resolving disagreements, including jurisdiction and arbitration clauses.
- **Visibility and Accessibility:**
  Ensure that the T&C document is easily accessible on your website (e.g., footer links, registration page) and requires user acknowledgment or agreement during account creation.
- **Regular Updates and Notifications:**
  Periodically review and update the T&C to align with evolving legal requirements and operational practices, and notify users about significant changes.

Reference list

Apache (2019). *httpd 2.4 vulnerabilities - The Apache HTTP Server Project*. [online] Apache.org. Available at: https://httpd.apache.org/security/vulnerabilities_24.html.

Author, V. (2024). *Password Security: Common Attacks and Best Practices*. [online] VAADATA - Ethical Hacking Services. Available at: https://www.vaadata.com/blog/password-security-vulnerabilities-attacks-and-best-practices/.

Cvedetails.com. (2024). *PHP PHP 8.2.12 security vulnerabilities, CVEs*. [online] Available at: https://www.cvedetails.com/version/1791746/PHP-PHP-8.2.12.html

[Accessed 28 Mar. 2025]

Fortinet (n.d.). *What is a Brute Force Attack? | Definition, Types & How It Works*. [online] Fortinet. Available at: https://www.fortinet.com/uk/resources/cyberglossary/brute-forceattack.

MariaDB KnowledgeBase. (2025). *MariaDB 10.4.34 Release Notes*. [online] Available at:

https://mariadb.com/kb/en/mariadb-10-4-34-release-notes/ [Accessed 28 Mar. 2025].

National Institute of Standards and Technology (n.d.). *NVD - CVSS v3 Calculator*. [online] nvd.nist.gov. Available at: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

OWASP (2023). *OWASP foundation, the open source foundation for application security*. [online] owasp.org. Available at: https://owasp.org/.

PortSwigger (n.d.). *Information disclosure vulnerabilities*. [online] PortSwigger. Available at: https://portswigger.net/web-security/information-disclosure.

PortSwigger (2024). *What is SQL Injection? Tutorial & Examples*. [online] Portswigger. Available at: https://portswigger.net/web-security/sql-injection.

www.f5.com. (n.d.). *What Is Cookie Poisoning?* [online] Available at: https://www.f5.com/glossary/cookie-poisoning.