# ZAP Scanning Report

Uneeq Interns

## Site: http://192.168.1.10

### Generated on Sun, 7 Jul 2024 14:49:39

### ZAP Version: 2.15.0

**ZAP is supported by the Crash Override Open Source Fellowship**

### Summary of Alerts

| Risk Level | Number of Alerts |
|---|---|
| High | 9 |
| Medium | 4 |
| Low | 0 |
| Informational | 2 |

### Alerts

| Name | Risk Level | Number of Instances |
|---|---|---|
| Cross Site Scripting (Reflected) | High | 24 |
| External Redirect | High | 1 |
| Path Traversal | High | 11 |
| Remote Code Execution - CVE-2012-1823 | High | 3 |
| Remote File Inclusion | High | 1 |
| Remote OS Command Injection | High | 1 |
| SQL Injection - MySQL | High | 8 |
| SQL Injection - SQLite | High | 6 |
| Source Code Disclosure - CVE-2012-1823 | High | 3 |
| Directory Browsing | Medium | 6 |
| Hidden File Found | Medium | 1 |
| Parameter Tampering | Medium | 14 |
| XSLT Injection | Medium | 6 |
| GET for POST | Informational | 1 |
| User Agent Fuzzer | Informational | 312 |

## Full Mitigation Strategies for Each Vulnerability

### High-Risk Vulnerabilities:

**1. Cross Site Scripting (Reflected)**
**Description:** This vulnerability allows an attacker to inject malicious scripts into web pages viewed by other users.

**Mitigation:**

**(1) Input Validation and Sanitization:**

- Validate all inputs against a strict whitelist.
- Use libraries like OWASP AntiSamy to sanitize HTML input.
- Encode user inputs before rendering them in the UI.

**(2) Output Encoding:**

- Encode dynamic data before outputting it to the browser using functions like htmlspecialchars() in PHP or similar in other languages.

**(3) Content Security Policy (CSP):**

- Implement CSP headers to restrict sources from which scripts can be executed.

**(4) HTTPOnly and Secure Flags:**

- Use HTTPOnly and Secure flags on cookies to prevent access via JavaScript and enforce secure transmission.

**2. External Redirect**
**Description:** This allows an attacker to redirect users to untrusted and potentially malicious websites.

**Mitigation:**

**(1) Input Validation:**

- Use a whitelist of allowed URLs.
- Validate and sanitize input parameters used for redirects.

**(2) Avoid Dynamic URLs:**

- Where possible, avoid using user-supplied input for URL redirection.

**(3) User Confirmation:**

- If redirection is necessary, prompt users for confirmation before redirecting.

**3. Path Traversal**
**Description:** This vulnerability allows attackers to access restricted directories and execute commands outside of the web root directory.

**Mitigation:**

**(1) Input Sanitization:**

- Sanitize file paths by removing or encoding special characters (e.g., .., /, \).

**(2) Use Fixed Paths:**

- Avoid using user input directly in file paths. Use fixed, known directories wherever possible.

**(3) Least Privilege Principle:**

- Ensure the application runs with the minimum privileges necessary.

**(4) File System Permissions:**

- Configure file system permissions to restrict access to sensitive files and directories.

**4. Remote Code Execution - CVE-2012-1823**
**Description:** This vulnerability allows attackers to execute arbitrary code on the server.

**Mitigation:**

**(1) Update Software:**

- Ensure your PHP version and all related software are up-to-date.

**(2) Disable Dangerous Functions:**

- Disable PHP functions like exec(), system(), shell_exec(), and passthru() if not needed.

**(3) Use Suhosin:**

- Implement the Suhosin patch or extension for PHP to provide additional security protections.

**5. Remote File Inclusion**
**Description: This vulnerability allows attackers to include remote files through the web browser.**

**Mitigation:**

**(1) Disable Remote File Inclusions:**

- Ensure allow_url_include is disabled in the PHP configuration (php.ini).

**(2) Input Validation:**

- Validate and sanitize all inputs used in file inclusions.

**(3) Use Absolute Paths:**

- Use absolute paths or predefined path constants to limit the scope of file inclusions.

**6. Remote OS Command Injection**
**Description:** This vulnerability allows an attacker to execute arbitrary commands on the server.

**Mitigation:**

**(1) Avoid System Calls:**
- Use built-in functions or APIs instead of system calls for command execution.

**(2) Input Validation:**
- Validate and sanitize all user inputs before using them in system calls.

**(3) Escape Shell Commands:**
- If system calls are unavoidable, escape all shell commands properly.

**(4) Use Least Privilege:**
- Run the application with the least privileges necessary.

**7. SQL Injection - MySQL & SQLite**
**Description:** This vulnerability allows attackers to execute arbitrary SQL code.

**Mitigation:**

**(1) Prepared Statements:**
- Use prepared statements and parameterized queries to prevent SQL injection.

**(2) Object-Relational Mapping (ORM):**
- Use ORM frameworks that handle query building and parameterization securely.

**(3) Input Validation:**
- Validate and sanitize all inputs before using them in SQL queries.

**(4) Database Permissions:**
- Use the principle of least privilege for database accounts.

**8. Source Code Disclosure - CVE-2012-1823**
**Description:** This vulnerability allows attackers to view source code, which may contain sensitive information.

**Mitigation:**

**(1) Server Configuration:**
- Ensure web server configurations are set to prevent source code disclosure (e.g., use Options -Indexes in Apache).

**(2) Update Server Software:**
- Keep server software up-to-date to mitigate known vulnerabilities.

**(3) Restrict Access:**
- Use .htaccess or other mechanisms to restrict access to sensitive files and directories.

**Medium-Risk Vulnerabilities**

**1. Directory Browsing**
**Description:** This allows attackers to view directory listings, which may contain sensitive files.

**Mitigation:**

**(1) Disable Directory Listing:**

- Disable directory listing in the web server configuration (e.g., Options -Indexes in Apache).

**(2) Use Index Files:**

- Ensure all directories contain an index.html or equivalent file to prevent directory listing.

**2. Hidden File Found**
**Description:** Hidden files may contain sensitive information that can be exploited.

**Mitigation:**

**(1) Remove Unnecessary Files:**

- Regularly audit and remove unnecessary files from the server.

**(2) Restrict Access:**

- Use .htaccess or other mechanisms to restrict access to sensitive files.

**3. Parameter Tampering**
**Description:** This vulnerability allows attackers to manipulate parameters to achieve unintended actions.

**Mitigation:**

**(1) Input Validation and Sanitization:**

- Validate and sanitize all input parameters.

**(2) Use Server-Side Checks:**

- Implement server-side validation to ensure parameters are not tampered with.

**(3) Cryptographic Measures:**

- Use cryptographic techniques to ensure data integrity (e.g., HMAC).

**4. XSLT Injection**
**Description:** This vulnerability allows an attacker to inject malicious XSLT code.

**Mitigation:**

**(1) Input Validation and Sanitization:**

- Validate and sanitize inputs used in XSLT transformations.

**(2) Use Secure Libraries:**

- Use secure XSLT processors that do not allow external entity references or unsafe constructs.

**(3) Least Privilege:**

- Run the XSLT processor with the minimum privileges necessary.

**Informational Vulnerabilities**

**1. GET for POST**
**Description:** Sensitive operations are performed via GET requests instead of POST, which may expose data in URLs.

**Mitigation:**

**(1) Use POST for Sensitive Operations:**

- Ensure that sensitive operations use POST requests.

**(2) Validate Request Methods:**

- Validate that requests use the appropriate HTTP methods.

**2. User Agent Fuzzer**
**Description:** Indicates that the application is potentially vulnerable to user-agent based attacks.

**Mitigation:**

**(1) Analyze Results:**

- Analyze fuzzing results to identify potential vulnerabilities.

**(2) Input Validation:**

- Validate and sanitize user-agent strings and other headers.

**(3) Implement Security Controls:**

- Implement necessary security controls based on the identified patterns and results.