# ENPM809W - Introduction to Secure Coding

## Project Phase 3 Report

*Author: Syed Mohammad Ibrahim*
*UID: iamibi*
*Email: iamibi@umd.edu*

Assigned Project: ENPM809WProject-xhmei

## Summary

The review found that there are at least four critical issues related the input handing, authentication and authorization that could result in arbitrary code execution and loss of data. These issues are required to be addressed as soon as possible or else the application may be compromised by attackers who will leverage these vulnerabilities. There are three high level severity, four medium level severity and two low level severity weaknesses found. Furthermore, there are two weaknesses that are listed as part of 2021 CWE Top 25 Most Dangerous Software Weaknesses.

## Introduction

Developer: Xiaohan Mei
Code URL: https://code.umd.edu/xhmei/ENPM809WProject-xhmei/-/tree/522a4723ff6bab3052de2e2164e3181aa8988fab

Assumptions:

- The website is intended for the usage of public and will be hosted on a public network.
- The security team along with SRE team has made the required security and deployment changes to the production environment.
- The website will have SSL setup in the production.
- The category of Judge and Chief judge are people who know about the working of the system.
- The judge(s) and chief judge(s) are already present in the system which was setup by the IT team.

Process for Code Review:

1. As the first step, I had a discussion with the developer on how the application operated. He showed me all the flows and I took notes of them.
2. I started my code review from the login and register pages where user inputs were expected. While going over the register flow in the code, I was able to figure out that the input was not being sanitized when it was being stored. This flow was open to input validation weakness. Similar thing was observed with the login flow as well, where the user can enter any arbitrary data as input, and it was not being validated or neutralized. The login page contained the weakness of input neutralization which I deemed with critical severity.
3. In the same flows, I went through the logging flows of the code. The logs were placed in places where the logging is necessary and sufficient information was being sent to the logs. However, there was no output neutralization that was being done while the data was being entered in the logs. This vulnerability can potentially cause a Denial of Service or Cross-Site scripting attack on the logs and is severity level high.
4. While registering, the passwords were being stored as salt + hash combination in the database. The algorithm being used is SHA256 with random salt generator. Internal libraries are being used such as “System.Security.Cryptography” for generating passwords and salts. This is a suggestive of good password storage scheme.
5. The password verification when the user tries to login appropriately returns a generic error message which is a good scheme to not let more information out of the system and show it to the user.
6. Going over the error pages, the pages don’t show any kind of sensitive information or stack traces. A good way to handle exception is being implemented here.
7. Session management is being done and the session is being cleared when the user is logged out. The cookies are timed out and the sessions are cleared during that time.
8. Trying to access a random page with path traversal techniques was unsuccessful as the application handled the routing correctly and threw appropriate generic error pages.
9. Functionality, there are three users with different roles present on the application. Contestant, Judge and Chief Judge. The functionalities are isolated. However, the view of judge and chief judge is based on the contestant’s poem that was submitted and is loaded on the screen in a text box without neutralization. This can potentially cause a Cross-Site scripting attack if an attacker participates as a contestant and submits a malicious code. The severity level is high for this one as it can lead to theft of credentials or session.
10. The primary authentication mechanism being used is the email of the user.
11. Error handling mechanisms were not adequate and required more detailed error handling by the application. This is a medium/high level severity depending on the range of errors that can either be insignificant or can cause a Denial of Service of some sort bringing down the application.
12. The application uses Captcha to avoid any kind of bot attack using brute force on the login page. However, the register page is still open to brute forcing.
13. The logout is not being tracked by the logging system. This is a low-level severity.
14. While registering, the password is visible and is not being covered by a mask. The severity of this is medium.

# Interview

The interview with the developer of the application had the following meeting notes:

- I asked about overall functionality of the application and the type of people that have different roles on the platform. The application is based on Poem Contest and there are three types of users that can be present on the platform (Contestant, Judge, and Chief Judge).
- What assumptions were made while setting up the production environment in contrast to the development environment when the application was being setup?
    - The said system is going to run SSL on the production environment.
    - Judge and Chief Judge role-based users will be added by the IT team beforehand and thus will not have the ability to sign up.
- What authentication mechanism is being used by the system? The application uses email-based authentication.
- Are the inputs being sanitized/neutralized by the applications? The application doesn't sanitize or neutralize the inputs. This was a big red flag as it has so many potential weaknesses that can be leveraged by an attacker.
- Is there any cryptographic usage in the application? The password is being stored as SHA256 with secure random salt generator. This was pointing me towards a good password storage technique that I would investigate the code.
- Does the application have authorization implemented? The application holds multiple users which are authorized to perform their roles. A contestant can register/login/submit a poem, a Judge and chief Judge can rate the poems and a Chief Judge can review the poem.
- Does the application use session management techniques? The application relies on the session management provided by the ASP.NET framework under "HttpContext.Session".
- Does the application handle error? If so, how? The application throws appropriate errors depending on the context. This required investigation through the code.
- Is the application logging any kind of data? The application logs required information like logging in, error raised.
- What other security mechanisms does the application implement? The application uses captcha to avoid any kind of bot attack using brute force mechanism at the time of logging in. This requires investigation.
- Which database is being used by the application and how many tables exist? MySQL is being used by the application and there are two tables present, namely Poem and Sysuser.

# Code Review Findings

| **CWE-20: Improper Input Validation** | |
|---|---|
| **Criticality** | Critical |
| **Security Category** | Input Handling |

| | |
|---|---|
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 47 |
| **Description** | The code doesn't neutralize the input characters that are being entered by the user. An attacker can enter html-based scripts as part of the input. |
| **Technical Impact** | If the input contains characters that can potentially be rendered as part of the html code, then an XSS attack can be performed causing the application severe issues depending on the attack type. |
| **Potential Mitigations** | Sanitizing the inputs with invalid characters can help in mitigating the threats. |
| **Comments** | The changes are present in the release page and require immediate action. |

| **CWE-20: Improper Input Validation** | |
|---|---|
| **Criticality** | Critical |
| **Security Category** | Input Handling |
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 112 |
| **Description** | The code doesn't neutralize the input characters that are being entered by the user. An attacker can enter html-based scripts as part of the input. |
| **Technical Impact** | If the input contains characters that can potentially be rendered as part of the html code, then an XSS attack can be performed causing the application severe issues depending on the attack type. |
| **Potential Mitigations** | Sanitizing the inputs with invalid characters can help in mitigating the threats. |
| **Comments** | The changes are present in the release page and require immediate action. |

| **CWE-434: Unrestricted Upload of File with Dangerous Type** | |
|---|---|
| **Criticality** | Critical |
| **Security Category** | Input Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 76 |
| **Description** | The code doesn't restrict the file uploads which can contain malicious content, even though the extension might be valid. |
| **Technical Impact** | The application doesn't check the contents of the file when it was uploaded and doesn't restrict the number of times a user can upload a file with malicious content. This can cause the server to either run out of memory or an arbitrary code execution on server. |
| **Potential Mitigations** | Validating the files and restricting the file upload for the user if the user's uploaded file contains malicious code. |

| | |
|---|---|
| **Comments** | The changes are present in the release page and require immediate action. |

| **CWE-307: Improper Restriction of Excessive Authentication Attempts** | |
|---|---|
| **Criticality** | Critical |
| **Security Category** | Authentication and Authorization |
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 35 - 84 |
| **Description** | The code doesn’t restrict the login attempts made by a user. |
| **Technical Impact** | An attacker can leverage a type of brute forcing technique depending on how they are able to bypass the captcha successfully. |
| **Potential Mitigations** | Adding a restricting login mechanism like failed attempts logging in a short span of time and making sure that the system fails silently with minimal message provided to the user. |
| **Comments** | |

| **CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')** | |
|---|---|
| **Criticality** | High |
| **Security Category** | Input Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 95 |
| **Description** | The code doesn’t neutralize the input characters that are being entered by the contestant while uploading the poem. An attacker can potentially upload a file containing malicious code that can cause cross-site scripting attack when rendered on the page. The poem is rendered to the judges as is. |
| **Technical Impact** | Depending on the type of malicious script, an attacker can perform a stored XSS or reflected XSS or credentials spoofing types of attacks. |
| **Potential Mitigations** | The poem when being rendered on the web page should be neutralized beforehand so that any malicious code can be escaped. Another alternative is to validate the poem when the user uploads it, before saving it in the database. |
| **Comments** | |

| **CWE-521: Weak Password Requirements** | |
|---|---|
| **Criticality** | High |
| **Security Category** | Authentication and Authorization |
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 112 |

| | |
|---|---|
| **Description** | The password input doesn't have a good password policy implemented. |
| **Technical Impact** | An attacker can potentially try guessing the password or can simply brute force leading to access to the account. |
| **Potential Mitigations** | Adding good password policies like length of password, addition of special characters and mix of alpha-numeric characters will resolve this issue. |
| **Comments** | |

| **CWE-117: Improper Output Neutralization for Logs** | |
|---|---|
| **Criticality** | High/Medium |
| **Security Category** | Logging |
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 52 |
| **Description** | The code doesn't neutralize the output being written to the log files. |
| **Technical Impact** | An attacker can potentially send in malicious code which is being directly written to the log. This can cause a denial of service by filling up the logs with invalid information. |
| **Potential Mitigations** | The input logs should be sanitized, and the security team should make sure that if a DOS based attack is being made like from a single IP address, it should be monitored and blocked. |
| **Comments** | The fix requires a hybrid of security team and SRE team to act on the requests and the developer to fix the neutralization issue. |

| **CWE-703: Improper Check or Handling of Exceptional Conditions** | |
|---|---|
| **Criticality** | Medium |
| **Security Category** | Error Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 94 |
| **Description** | The code doesn't handle errors sufficiently like in this case, the file being read might not be available or the memory is not enough to read the whole file. |
| **Technical Impact** | This can cause an application to run out of memory or can expose errors to the users in some way. |
| **Potential Mitigations** | Adding more error handling in critical places like login, logout, db related operations. |
| **Comments** | |

| **CWE-703: Improper Check or Handling of Exceptional Conditions** | |
|---|---|
| **Criticality** | Medium |
| **Security Category** | Error Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 127 |

| | |
|---|---|
| **Description** | The code doesn’t handle errors sufficiently like in this case, the file being read might not be available or the memory is not enough to read the whole file. |
| **Technical Impact** | This can cause an application to run out of memory or can expose errors to the users in some way. |
| **Potential Mitigations** | Adding more error handling in critical places like login, logout, db related operations. |
| **Comments** | |

| **CWE-703: Improper Check or Handling of Exceptional Conditions** | |
|---|---|
| **Criticality** | Medium |
| **Security Category** | Error Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 159 |
| **Description** | The code doesn’t handle errors sufficiently like in this case, the file being read might not be available or the memory is not enough to read the whole file. |
| **Technical Impact** | This can cause an application to run out of memory or can expose errors to the users in some way. |
| **Potential Mitigations** | Adding more error handling in critical places like login, logout, db related operations. |
| **Comments** | |

| **CWE-549: Missing Password Field Masking** | |
|---|---|
| **Criticality** | Medium |
| **Security Category** | Input Handling |
| **Filename** | PoetryContestSystem\Views\Account\Register.cshtml |
| **Line Number/Position** | 52 |
| **Description** | The code doesn’t mask the password field on the register page. |
| **Technical Impact** | The issue increases the potential for attackers to observe and capture passwords. |
| **Potential Mitigations** | Adding mask to the password field can resolve this issue. |
| **Comments** | |

| **CWE-755: Improper Handling of Exceptional Conditions** | |
|---|---|
| **Criticality** | Medium |
| **Security Category** | Error Handling |
| **Filename** | PoetryContestSystem\Controllers\PoemController.cs |
| **Line Number/Position** | 130 |
| **Description** | The code doesn’t handle the conversion value inside an exception block which in turn will be propagated to the client. |

| | |
|---|---|
| **Technical Impact** | This can give an attacker an insight on the type of operations being performed inside the application and that the application can potentially be crashed if some crafty value is passed. |
| **Potential Mitigations** | Adding an exception handling block and logging appropriately will help in remediating the problem. |
| **Comments** | |

| **CWE-778: Insufficient Logging** | |
|---|---|
| **Criticality** | Low |
| **Security Category** | Logging |
| **Filename** | PoetryContestSystem\Controllers\AccountController.cs |
| **Line Number/Position** | 126 - 144 |
| **Description** | The code doesn't log sufficiently in critical places like logout. |
| **Technical Impact** | While auditing, the issues can be hard to backtrack if relevant information is not present to paint the whole picture. |
| **Potential Mitigations** | Adding logs to the security-critical areas. |
| **Comments** | |

| **CWE-561: Dead Code** | |
|---|---|
| **Criticality** | Low |
| **Security Category** | Debugging |
| **Filename** | Common\SecurityMgr.cs |
| **Line Number/Position** | 17 |
| **Description** | The code is never executed by the program. |
| **Technical Impact** | An attacker can potentially make use of this dead code as a form of attack vector. |
| **Potential Mitigations** | Remove or refactor any/all dead code from the code base. |
| **Comments** | |

# Impact

As the report mentions, the application requires immediate remediation of some critical/high severity weaknesses which can potentially bring down the services of the application, data breach or even a denial of service for regular user. It is advisable that the release should be delayed, and another iteration of review should be done.