Nicholas C. Chlumsky

**Project Security Assessment**

1. **Summary**

The goal of the security assessment was to find vulnerabilities and weaknesses in the project which manages adding, removing, and managing user accounts. Some methods used to identify vulnerabilities include penetration testing, code reviews, and research into different topics.

1. **Assessment Scope**

The tools and software used in the security assessment include GitHub and Visual Studio used to identify the vulnerabilities in the project. Some major limitations would include finances and lack of multiple OSs to test the codebase on different systems.

1. **Summary of Findings**

Of the findings in the assessment, 1 was considered a high risk, 5 were considered moderate risks, and 5 were considered low risks. These risks were determined to be either vulnerabilities or security issues from the previous security assessments on the code. The findings from the assessment are displayed in the figure below:

**A screenshot of a computer program

Description automatically generated with medium confidence**

1. **Summary of Recommendations**

The original assessments into the vulnerabilities of the project that were fixed include several security issues such as displaying user passwords, not encrypting passwords when storing them, and potential buffer overflow/buffer attacks. Additional security issues include removing the std namespace from the project and adding better validation and sanitization. Some changes still needed include finding a better user account storage solution rather than in a CSV-like text file.

1. **Goals, Findings, and Recommendations**
2. **Assessment Goals**

The goal of the security assessment was to find vulnerabilities in the project and fix weaknesses in the code that would lead to unauthorized access, attacks, breaches, or other unforeseen third-party attacks. The assessment aim is to fix these issues and reduce the risk of any potential threats.

1. **Detailed Findings**

The security assessment found several issues with the code. These issues include but are not limited to:

1. Insufficient Data Validation

Insufficient data validation was determined to be a threat as an attacker would be able to use it to gain unauthorized access to private data and result in security breaches.

1. Insufficient Input Validation

Insufficient input validation was determined to be a vulnerability associated with unsecure data storage. The code lacked input validation for some methods which would result in potential data breaches.

1. Weak User Passwords

Weak user passwords were identified to be a vulnerability as any string is allowed to be entered in as a valid password. This means that extremely unsecure passwords or common passwords are allowed.

1. Unsecure Data Storage

Unsecure data store was determined to be a threat and vulnerability as some commands in the project would display users’ passwords. Additionally, user data was stored in plaintext which is not recommended.

1. No Encryption

In the project, there was originally no encryption which is a huge vulnerability, as any data breach would immediately result in users’ passwords and information being leaked. After finding the vulnerability, encryption was implemented into the project.

1. Lack of Input Sanitization

The project lacked input sanitization which could lead to security breaches. One example is the addUser method which did not have any sanitization, which could lead to unauthorized access.

1. Insufficient Error Handling

The code lacks error handling and logging which could lead to potential security issues and possibly data breaches, as any exceptions thrown during the execution are not logged or stored anywhere.

1. Potential Buffer Overflow Vulnerabilities

The code is vulnerable to buffer overflow attacks which could result due to insufficient input validation/sanitization.

1. **Recommendations**

Some recommendations regarding these vulnerabilities include:

* Removing c\_str from methods which would result in potential race conditions.
* Add input validation for all methods and ensure that all inputs are correctly sanitized.
* Add additional error handling and ensure that exceptions are logged for future reference.
* Ensure that all user information is securely encrypted and stored in a valid data storage format.
* Ensure that user passwords are long and secure enough to ensure that the potential for users’ accounts being breached is lower.
* Ensure that file handling is done correctly to ensure that any issues related to reading or writing to the file does not happen.
* Ensure the code is easily readable and understandable so that any issues related to security are easily readable.

1. **Methodology for the Security Control Assessment**

The tools used during the security assessment include GitHub and Visual Studio. During the security assessment some of the processes we did include penetration testing, testing different scenarios and inputs, and doing code reviews which all helped in identifying potential vulnerabilities and weaknesses. The manual inspection and testing of the code also led to the finding and identification of additional vulnerabilities. Additional research was also done to learn about vulnerabilities in methods used when handling user input and validation techniques were implemented to ensure that any race conditions are not met.

Penetration testing with a wide range of different user inputs were done which resulted in the finding and fixing of several user input vulnerabilities including buffer overflow attacks, string validation issues, and other security vulnerabilities in the code.

1. **Figures and Code**

The process flow of the system includes logging the current user in which determines their permission level. This permission level determines what methods and information they can run and access.

|  |  |
| --- | --- |
| Step | Description |
| 1 | Initialize Variables and Do Prep |
| 2 | Log the active user in by asking for their username and password |
| 3 | Continue asking for login information if account details are incorrect |
| 4 | When user is logged in, give the user the available methods based on their permission level |
| 5 | When user adds or removes a user, we modify the user data file and write changes back to the file using one filestream. |
| 6 | Depending on the user’s permission level, add\_user, delete\_user, print\_all\_users, reset\_password, and modify\_permissions may or may not be available. |

The use case diagram for the project would look like the following:  
1. Log active user in  
2. Add User  
3. Delete User  
4. View all Users  
5. Reset user password  
6. Modify user permissions.

The user would first initiate the login portion when the project first runs, this will prompt them for their username and password. Once they are logged in, they can perform the remaining use cases depending on their permission level. IE: Users with permission level 1 can add new users, users with permission level 2 can delete users, users with permission level 3 can view all users and reset passwords, and users with permission level 4 can modify user permissions. The add user would allow users to add new users by providing a new username and password, and permission level if the current users permission level is at least 3. The delete user method would allow you to delete any user with a lower permission level than the current user. And the view all users prints a list of all users’ usernames and permission levels. The reset password method allows you to reset a user’s password. And finally, the modify user permissions allows you to change the permission level of an existing user by providing their username and a new permission level.

1. **Works Cited**
2. 77INFINITE77. (2021, April 26). C++ Best Practice #1: Don't simply use: Using namespace STD; DEV Community. Retrieved May 1, 2023, from https://dev.to/77bala7790/c-best-practice-1-don-t-simply-use-using-namespace-std-4n6b
3. Common weakness enumeration. CWE. (n.d.). Retrieved May 1, 2023, from https://cwe.mitre.org/data/definitions/119.html
4. Common weakness enumeration. CWE. (n.d.). Retrieved May 1, 2023, from <https://cwe.mitre.org/data/definitions/120.html>
5. Khintibidze, L. (2021, January 1). Validate user input in C++. Delft Stack. Retrieved May 1, 2023, from https://www.delftstack.com/howto/cpp/cpp-input-validation/
6. How do I sanitise cin? Stack Overflow. (1959, August 1). Retrieved May 1, 2023, from https://stackoverflow.com/questions/13281669/how-do-i-sanitise-cin
7. Introduction to owasp zap. TryHackMe. (n.d.). Retrieved May 1, 2023, from https://tryhackme.com/room/learnowaspzap