# CS 410 Project Two Security Report – Kerrian Offermann

Identify where multiple security vulnerabilities are present within the blocks of C++ code. You may add columns and extend this table as you see fit.

| **Block of C++ Code** | **Identified Security Vulnerability** |
| --- | --- |
| #include<iostream> | There is no system in place to verify if user is allowed into the system. In the long run, there should be a database set in place where verified users and their passwords are compared to what is being entered in the fields. |
| using namespace std; |
|  |
| int CheckUserPermissionAccess() { |
| string username; |
| int password; |
| int correct = 0; |
| cin >> username; |
| cout << "Enter your password:" << endl; | There is no system in place to ensure that the password matches a database |
| cin >> password; |
| if (password == 123) { | The password only works if it is 123 which is an easy password to guess. Even if it is a test password for a test account, there should be some complexity to it so hackers cannot break into the system easily. |
| correct = 1; |
| } |
| else { |
| correct = 2; |
| } |
| return correct; |
| } |
|  |  |
| int main() { |  |
| cout << "Created by SNHU and Kerrian Offermann for CS-410."; |  |
| string username; |  |
| int password; |  |
| cout << "Hello! Welcome to our Investment Company" << endl; |  |
| cout << "Enter your username:" << endl; |  |
| cin >> username; |  |
|  |  |
| cout << "Enter your password:"; |  |
| cin >> password; | Outside of checking '123', there is no system in place to compare password to verified user |
| int correct = CheckUserPermissionAccess(); |  |
| if (correct = 2) { // | This should be "1" since '123' is the correct password and it returns "1" |
| while (1) { |  |
| int choice, service1, service2, service3, service4, service5; |  |
| string name1, name2, name3, name4, name5; | Variables were never used (meant for names listed below) |
| cout << "What would you like to do?"; |  |
| cin >> choice; |  |
| if (choice = 1) { |  |
| cout << "Client's Name" << " " << "Service Selected (1 = Brokerage, 2 = Retirement)"; |  |
| cout << "1. Bob Jones selected option" << " " << service1; |  |
| cout << "2. Sarah Jones selected option" << " " << service2; |  |
| cout << "3. Amy Friendly selected option" << " " << service3; |  |
| cout << "4. Johnny Smith selected option" << " " << service4; |  |
| cout << "5. Carol Spears selected option" << " " << service5; |  |
| break; | Placing a break here will close the menu instead of returning to the main loop |
| } |  |
| if (choice = 2) { |  |
| int client, service\_choice; |  |
| cout << "Enter the number of the client that you wish to change" << endl; |  |
| cin >> client; |  |
|  |  |
| cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl; | This question should be asked after the client is selected rather than right after entering the client number |
| cin >> service\_choice; |  |
| if (client == 1) { |  |
| service1 = service\_choice; | Should return user to main menu after selection |
| } |  |
| if (client == 2) { |  |
| service2 = service\_choice; | Should return user to main menu after selection |
| } |  |
| if (client == 3) { |  |
| service3 = service\_choice; | Should return user to main menu after selection |
| } |  |
| if (client == 4) { |  |
| service4 = service\_choice; | Should return user to main menu after selection |
| } |  |
| if (client == 5) { |  |
| service5 = service\_choice; | Should return user to main menu after selection |
| } |  |
| } |  |
| if (choice = 3) { |  |
| break; |  |
| } |  |
| } |  |
| } |  |
| else { |  |
| cout << "Invalid password. Please try again."; | Should return user to place to re-enter credentials |
| } |  |
| } |  |

Explain the *security vulnerabilities* that are found in the blocks of C++ code.

The first security vulnerability I came across is a lack of verifying if the usernames belong to verified users. This is not something that can be constructed with just one page of C++. It will require SNHU Investments to look into creating a database where users are added or given the ability to register as verified users. Another security vulnerability that I found is the default password of ‘123’. Even with a test account, complex passwords should be a requirement.

An error that could become a vulnerability are the unused variables. Even if there is no known way to utilize unused yet initialized variables for malicious reasons, it is generally programming etiquette to not leave variables lying about the program that have no purpose. There are also a few codes that would be better off removed (such as the break after displaying clients) instead of looping back to the menu or relocated (like asking the client’s choice after choosing the client).

**Describe *recommendations* for how the security vulnerabilities can be fixed.**

The account verification mentioned previously is the first recommendation. Any software or program that requires users to access sensitive information should have a system in place to divide wanted users from unwanted users. In other words, a database that can let the right users in and keep unwanted users out. As mentioned before, it cannot be established in a page of C++ code alone and will require the creation of a database, so this is a vulnerability that SNHU Investments will have to start an entire project around possibly.

Another recommendation for fixing a security vulnerability is to require passwords to meet certain requirements so they cannot be predictable. “123” is just a test code for this project, but in the future, users should be prohibited from creating simple passwords so hackers cannot break into their accounts to access sensitive information.

The last recommendations are the part of the code that can have immediate fixes. This includes adding clarification to menu options, adding and removing loop breaks where needed, and cleaning up unused variables.