# CS 410 Project Two Security Report

## Instructions

Fill in the table in step one. In steps two and three, replace the bracketed text with your answer in your own words.

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** |
| --- | --- |
| vector<Client> clients; // Vector for storing clients  Client client; // Initialized client object for later use | vector and client object have a global scope meaning they are not protected |
| if (password == "123") { // If the password is 123  permissionGranted = true; // set permissionGranted to true  } | only the password is checked, username should also be verified to ensure security. There should also be a maximum number of attempts to prevent brute force attempts. |
| cout << "Enter the number of the client that you wish to change" << endl;  cin >> clientNumber;  cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;  cin >> clientChoice; | Original file used int as variables for client number and choice, this allowed the program to break if the user entered anything else. I used string variables to prevent this. |
| string userChoice = "0"; | userchoice was also an int in the original file which allowed the same bug mentioned above. My solution was also the same. |
| else {  // Not a part of original program but I feel that this helps avoid issues with menu selection  cout << "Invalid Input. Please try again" << endl;  } | In the original file there was not a catchall for if the user entered something that was not an option, this could leave the program open to abuse |
|  |  |
|  |  |
|  |  |

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

For the first vulnerability, having the variables that store user data, like the client’s names and choices, leave the data extremely easy to access without proper authority. I was not able to find a solution to this problem however, since multiple other functions need access to these variables and the variables need to maintain their state. If I could alter the code significantly, I would write to a protected document and read from that document instead, or had the data stored in a database.

For the second vulnerability, there was no check in the original program to verify that the username was correct. This makes it much easier to break into the program by simple brute forcing passwords. There is also no maximum number of attempts, which leaves the program open to the same problems.

The third and fourth vulnerabilities are the same. In the original program these variables were handled with int and the user input was not validated before being passed. This meant that if a user entered in any input that was not an int, the program would break.

The final entry that I included was that in the original program there seemed to be no catchall for if the user inputted an option that was not on the menu, while this might not be a vulnerability it does make the use of the program less clear to the user.

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

For the first vulnerability, while global variables are almost always considered bad, for this case I think that it is appropriate since multiple functions need access to it, and it needs to maintain its data. My recommendation to solve this and make the data more private would be to either have the information stored in a protected document or database using encryption to protect the data being stored, since I was unable to find a solution that would solve it within the CPP source code.

For the second vulnerability, adding a counter that tracked the number of attempts and checking that the username also matches will fix this vulnerability and make the code much more secure and less vulnerable to brute force attacks.

For the third and fourth, there are two ways this vulnerability can be negated as far as I know. The first is to validate that the user input was an int. This could be done with a try and catch block. The other is to use a variable that can handle all inputs, like a string variable. Since the string variables memory is allocated dynamically and can handle all possible inputs, I found this to be the best solution. For the third vulnerability specifically, I also verified that the client number inputed was within the range of the client list, and that the choice was one of the two options.