# CS 410 Project Two Security Report

Elijah Hickey

8/14/21

CS-410

## 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** |
| --- | --- |
| string name1 = "Bob Jones";  string name2 = "Sarah Davis";  string name3 = "Amy Friendly";  string name4 = "Johnny Smith";  string name5 = "Carol Spears";  int num1 = 1;  int num2 = 2;  int num3 = 1;  int num4 = 1;  int num5 = 2; | 1. Non-const global variables are used. |
| int CheckUserPermissionAccess() {  string username;  int password;  cout << "Enter your username: " << endl;  cin >> username;  cout << "Enter your password: " << endl;  cin >> password;  return password;  } | 2. No input validation to check for correct input data type. Does not prevent user from inputting spaces or ignore spaces while using just cin. |
| void ChangeCustomerChoice() {  int changechoice, newservice;  cout << "Enter the number of the client that you wish to change" << endl;  cin >> changechoice;  cout << "Enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;  cin >> newservice;  if (changechoice == 1) {  num1 = newservice;  }  if (changechoice == 2) {  num2 = newservice;  }  if (changechoice == 3) {  num3 = newservice;  }  if (changechoice == 4) {  num4 = newservice;  }  if (changechoice == 5) {  num5 = newservice;  }  } | 3. Does not have any input validation.  Does not validate that input is the correct data type.  Does not validate that the input is within the correct range.  Does not validate that the input meets the systems needs (service choice should be 1 or 2.) |
| int main()  {  int answer, choice;  cout << " ---------------------------------------------\n|CS-410, Project One, Created by Elijah Hickey|\n ---------------------------------------------\n" << endl;  cout << "Hello! Welcome to our investment Company\n";  answer = CheckUserPermissionAccess();  while (answer != 123) {  cout << "Invalid Password. Please try again" << endl;  answer = CheckUserPermissionAccess();  }  choice = 0;  while (choice != 3) {  cout << "What would you like to do?\nDISPLAY the client list (enter 1)\nChange a client's choice (enter 2)\nExit the program.. (enter 3) " << endl;  cin >> choice;  cout << "You chose " << choice << endl;  if (choice == 1) {  DisplayInfo();  }  else if (choice == 2) {  ChangeCustomerChoice();  }  }  return 0;  } | 4. Lacks any input validation  Inputs are not validated that they are the correct types.  Inputs are not validated that they are within range.  Inputs are not sanitized while being passed to CheckUserPermissionAccess(). |

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

Security vulnerability 1

This program uses a non-constant global variable as the intended method of storing the users information. This information is meant to be altered with ChangeCustomerChoice() and is meant to be displayed with DisplayInfo(). Sections should only have access to these variables if they need to; but if they are global it is much more difficult to restrict this access, especially as a system becomes more complex. Any part of a system should only have access to what it needs and nothing more. Variables with uses like these have no business being global. If critical data in a large system was held in global variables, one system granting unintended access to these variables could be a huge problem.

Security vulnerability 2

CheckUserPermissionAccess takes the users input as a string and password as an int. This function has no validation, which means that undefined behavior can occur. For the string, since just cin is used (no ignore, getline, etc.) It will only capture the first entry before a space for username. The second after the space will be sent to password. This can lead to undefined behavior. Neither have validation that the input will match the expected data type, or validation to prevent integer overflow. Both these validations are needed to prevent other possible undefined behaviors from occurring.

Security Vulnerability 3

ChangeCustomerChoice is used to change a service choice from either 1 or 2.

The main security issue again comes to a complete lack of validation. The user passes two inputs into ints with cin, however inputs are not validated to ensure the correct data type or range of numbers is used. Incorrect data type, or entry outside of range can lead to undefine behavior or integer overflow.

Security Vulnerability 4

Main function, much like the rest of the program, completely lacks user input. This is the most common cause of undefined behavior (also the simplest to prevent in many cases.) Main has two ints that will be passed the user’s inputs. These do not validate that the input is the correct data type, or within a range to prevent buffer overflow. Additionally, this data is being passed to another part of the system. Data passed to another part of a system needs to also be sanitized, to ensure that that data is not passing unexpected commands.

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

Security Vulnerability 1 – Recommendations

Recommendations is to avoid the use of global variables except for constant variables. These type of variables (the clients and their choice) should NOT be declared as global. In this case, with how the program is set up, they can be declared in main, and passed to functions during those function calls. This would ensure that functions only have access to these variables when they are explicitly called and passed the variables.

Security Vulnerability 2 – Recommendations

For CheckUserPermissionAccess, the program should perform input validations to ensure that the password the user enters is the proper data type. Additionally, minimal and maximal inputs should be allowed, to prevent an integer overflow when the user types in a number. For the username, cin.ignore can be used with numeric limits to only pass the first word (first entry before a space.)

Security Vulnerability 3 – Recommendations

ChangeCustomerChoice has ints, and should thus be validated as we have been doing whenever ints are involved. Validation can ensure that entries are the correct data type, and that the inputs are within valid ranges. Simply, we can only accept inputs are within the expected ranges (1-5 for client choice, and 1-2 for service choice. ) While this is to handle non-valid choices, it also prevents integer overflows from happening. I plan to use an array of lists as the structure to hold the clients info. Since this is the case, we must make sure that we don’t try accessing an element of the array that does not exist.

Security Vulnerability 4 – Recommendations

Main is needing the same sort of validations. Main expects two integer values from the user. Validation should be used to make sure that their inputs are in fact integers. Additionally, we can again force entries to be within the expected range, which can be used to prevent integer overflow. If we were to pass data to other systems, it is important to understand how the other systems operate, and sanitize the data if needed. In this case, the clients are stored within main, and the changes to them are restricted in such a way that sanitization should not be needed here (especially since we aren’t dealing with something like an SQL database.) In larger or more complicated programs, sanitization might be needed.