# CS 410 Project Two Security Report Template

## 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** |
| --- | --- |
| int CheckUserPermissionAccess() {  std::string username = "";  std::string password = "";  std::cout << "Enter your username: \n";  std::cin >> username;  std::cout << "Enter your password: \n";  std::cin >> password;  if (password == "123") {  return 1;  }  else {  return 2;  }  } | No input validation found in either of the cin statements. Invalid types or overflow can cause undefined behavior. Passwords should also be encrypted and not hard coded so that system breaches are less likely to leak personal information. |
| void ChangeCustomerChoice(int& num1, int& num2, int& num3, int& num4, int& num5) {  int changechoice = NULL;  int newservice = NULL;  std::cout << "Enter the number of the client that you wish to change\n";  std::cin >> changechoice;  std::cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)\n";  std::cin >> newservice;  if (changechoice == 1) {  num1 = newservice;  }  else if (changechoice == 2) {  num2 = newservice;  }  else if (changechoice == 3) {  num3 = newservice;  }  else if (changechoice == 4) {  num4 = newservice;  }  else if (changechoice == 5) {  num5 = newservice;  }  return;  } | Input validation is missing again. Variables changechoice and newservice update user profiles depending on what is selected. A number selected outside of the available range will have no effect and not display a warning to the user. User of characters or other unsupported types along with overflow can cause undefined behavior. |
| while (answer != 1) {  answer = CheckUserPermissionAccess();  if (answer == 1) {  break;  }  else {  std::cout << "Invalid Password. Please try again\n";  }  } | In the main function this while loop checks the password inputted against the one in the system. If the password is correct, it breaks and moves forward otherwise it outputs a message and loops again. In order to prevent malicious attacks, there should be a loop counter that blocks login attempts if a threshold is exceeded. |
| while (choice != 3) {  std::cout << "What would you like to do?\n";  std::cout << "DISPLAY the client list (enter 1)\n";  std::cout << "CHANGE a client's choice (enter 2)\n";  std::cout << "Exit the program.. (enter 3)\n";  std::cin >> choice;  std::cout << "You chose " << choice << "\n";  if (choice == 1) {  DisplayInfo(num1, num2, num3, num4, num5);  }  else if (choice == 2) {  ChangeCustomerChoice(num1, num2, num3, num4, num5);  }  else if (choice == 3) {  return 1;  }  } | In the main function after authentication has been verified, there is a while loop menu that takes a value from the user after a selection screen is displayed. There is no input validation which can cause undefined behavior if a different data type is entered. If an integer outside of the accepted range of values is entered then no warning is displayed and the selection menu is outputted to the terminal again. |
|  |  |
|  |  |
|  |  |
|  |  |

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

Most of the vulnerabilities found in the code had to do with input validation. By taking in an input and not checking it against accepted values the programmer is allowing unexpected scenarios to occur. In the CheckUserPermissionAccess function the username and password of the user is accepted by input and is checked against the password. Strings that are too long can overwrite other sections of memory in C++ which can cause undefined behavior. ChangeCustomerChoice also accepts user input with no checks against types or limits. There is also no feedback to the user to indicate that the selection inputted was incorrect before prompting them again. These issues show up again in the main function during the selection screen as well. The last security issue discovered was that there are no checks in place on the number off times a user can try to input a password at authentication. Authentication attempts higher than three increases the likelihood of a malicious attacker trying to gain access to the system. Checks need to be in place to identify and stop access to users who are trying many different passwords unsuccessfully.

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

The main concern in this application is input validation. It occurs frequently and remains unchecked. An easy solution is to create a function that handles cin statements and runs checks on the input provided to verify that it is within acceptable parameters. This lowers code redundancy since I would not have to add validation to every function in the application. Lastly, there needs to be a way of checking how many times a user has attempted to authenticate in the system. By adding a counter that terminates the program when three is reached, the user will not be able to gain access to the system after three attempts have been made which is standard for many authentication platforms.