# 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 = "";    while (true) { *// While loop added to keep taking input until valid submission is obtained*      std::cout << "Enter your username: \n";      username = StringInputValidation(); *// String input validation*      if (username != "") {        break;      }    }    while (true) { *// While loop added to keep taking input until valid submission is obtained*      std::cout << "Enter your password: \n";      password = StringInputValidation(); *// String input validation added*      if (password != "") {        break;      }    }    if (password == "123") {      return 1;    }    else {      return 2;    }  } | The main issue with this type of coding is having the password hardcoded and not hashed or obscured in anyway |
| void DisplayInfo(int& num1, int& num2, int& num3, int& num4, int& num5) {    std::cout << "Client's Name    Service Selected (1 = Brokerage, 2 = Retirement)" << "\n";    std::cout << "1. " << "Bob Jones" << " selected option " << num1 << "\n";    std::cout << "2. " << "Sarah Davis" << " selected option " << num2 << "\n";    std::cout << "3. " << "Amy Friendly" << " selected option " << num3 << "\n";    std::cout << "4. " << "Johnny Smith" << " selected option " << num4 << "\n";    std::cout << "5. " << "Carol Spears" << " selected option " << num5 << "\n";    return;  }  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;    if (changechoice >= 1 && changechoice <= 5) {      return;    }    std::cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)\n";    std::cin >> newservice;    if (newservice == 1 || newservice == 2) {      return;    }    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;    }    else {      std::cout << "You have entered a number out of the expected range. \n" << "Please enter a number between 1 and 5\n";    }    return;  } | All of the info is shown before checking and validating the user’s credentials again. |
|  |  |

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

Before I had made changes to the original code there were a lot of problems with input validation such as the CheckUserPermissionAccess function didn’t check that both the username and password were correct or even valid which wouldn’t have stopped an overflow attack or just having the password and ignoring the username all together, Another main issues is the hard coding of the password which would tell anyone that looks at the source code of the app or program that there is a guaranteed to work password and it is written in plain text. Also Originally there was no input validation when it came to changing the ChangeCustomerChoice function which originally didn’t have any protections against overflow or choosing an incorrect option and wouldn’t alert the user that they have entered an incorrect number or choice. Another thing that wasn’t originally addressed was how to stop brute force attempts so implementing a limit to login attempts helps with reducing them

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

The main things that need to be fixed and have been fixed is the issue of input validation and making it so that overflow or incorrect choices aren’t used in the program, this can be done by using a function that will handle these issues such as limiting the user’s input to only 21 characters. This is a good idea to move it to its own function as it helps with code readability and less code redundancy. The last thing I incorporated was a login attempts system that would limit the number of attempts a given user has when trying to login to the system and would tell the user when the attempts have been used up.