# 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 num1 = 1,  num2 = 1,  num3 = 1,  num4 = 1,  num5 = 1;  string name1 = "Bob Jones",  name2 = "Sarah Davis",  name3 = "Amy Friendly",  name4 = "Johnny Smith",  name5 = "Carol Spears"; | Insufficient data validation  Global variable that can cause bugs |
| int CheckUserPermissionAccess() {  string password;  cout << "Enter your password:" << endl;  cin >> password;  if (password == "123") {  return 1;  }  else {  return 2;  }  } | Missing authentication and validation  Weak password and is exposed |
| void ChangeCustomerChoice() {  int changechoice, newservice;  cout << "Enter the number of the client that you wish to change" << endl;  cin >> changechoice;  cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;  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;  }  }} | Input is not validated  Missing exception handling to catch errors  Possibility of throwing itself into a loop. |
| while (true) {  cout << "What would you like to do?" << endl;  cout << "DISPLAY the client list (enter 1)" << endl;  cout << "CHANGE a client's choice (enter 2)" << endl;  cout << "Exit the program (enter 3)" << endl << endl;  cin >> answer;  cout << "You chose " << answer << endl << endl;  if (answer == 1) {  DisplayInfo();  }  else if (answer == 2){  ChangeCustomerChoice();  }  else if (answer == 3) {  break;  }  } | Input is not validated  Missing exception handling to catch errors.  If user submits a bad input, program have a chance cause the system to crash. |
|  |  |

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

The first code snippet is hard-coded, making it vulnerable to security risks. Anyone with access to sensitive data makes the code more susceptible to security risks. Storing usernames directly into the code makes it easier for attackers to steal the information. Another issue with the code is that it lacks data validation, enabling attackers to inject malicious code into the input fields. Having hard-coded sensitive information also makes the code vulnerable to overflows, leading to attacks on the surface.

Within the second code block is the CheckUserPermissionAccess function. The most critical issue within the function is its lack of authentication and validation of the password and the fact that the password is stored within the code. Having the password within the code enables any user to access the code, allowing attackers to steal information or cause further damage. With proper methods for authentication and validation, any unhandled wrong input can prevent the system from crashing.

In the code block ChangeCustomerChoice, there is a lack of user input validation. Suppose a user has used symbols in the input; this would lead to the program breaking and crashing, making the system stuck in a loop, and the user will not be able to complete their task. Other issues within the validations are using numbers that surpass the options or negative numbers. The system continues to process despite the user’s negative input or numbers that are more significant than the maximum value. If a user can input any number they choose, it defeats the purpose of the code.

Lastly, once one has access to the display menu, there are a lot of vulnerabilities in this area as well. For starters, the code block does not have any form of validation. Like CheckUserPermissionAccess and ChangeCustomerChoice, users can submit any input regardless of the options provided. If a user submits symbols as input, it will cause an infinite loop that will eventually crash the system. Furthermore, there are no limitations to the input. Once a character exceeds the limit of how many characters can be inputted, the system continues into an infinite loop, causing the system to crash. Suppose a user submits an integer that is a negative number or a number greater than the choices provided; the system will throw the user back into the main menu, which may complicate the readability of the system, without informing the user that an invalid option was submitted, there is a possible of continuous error leading to a bad user experience.

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

For the first vulnerability, the best recommendation is to fix it by using a vector and storing it as an object. Vectors can help with data structures for organizing objects within the program's memory. Using vectors to store user credentials helps to improve efficiency and scalability in any project because they are faster when data needs to be modified.

Turning to the second code block, the initial step is to eliminate the hard coding of the password into the system. The subsequent step involves implementing password validation. The use of cin.ignore(numeric\_limits<streamsize>::max(), '') is instrumental in disregarding the maximum number of characters, thereby averting potential crashes. Similarly, cin.good plays a vital role in verifying the string's validity, thereby fortifying the security protocol.

Like the CheckUserPermissionAccess code block, the ChangeCustomerChoice code block also shares similar recommendations for resolving security vulnerabilities. In both cases, the use of cin.ignore(numeric\_limits<streamsize>::max() and cin.good is crucial in reducing security vulnerabilities from potential attackers. Particularly in the CheckUserPermissionAccess code block, cin.good is indispensable as it mandates users to input validated data to proceed. Without this validation, users could continue to input invalid choices, leading to system crashes.

Lastly, the last code block would follow the same recommendations as CheckUserPermissionAccess and ChangeCustomerChoice by using cin.ignore(numeric\_limits<streamsize>::max() and cin.good for the same reasons. Other considerations possible for this code block include displaying clear error messages to prompt users to re-enter a valid choice. Submitting valid decisions can assist in preventing unexpected behavior from occurring.

Besides issues within the code blocks, other recommendations to ensure security are up to date are ensuring all libraries are up to date, frequent testing to ensure quality and efficiency meet industry standards and continuing secure coding practices. Technology is advancing quickly, and developers are responsible for adapting to frequent changes to ensure security vulnerabilities can be resolved or mitigated.