# 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** |
| --- | --- |
| string user = "";  string password = "";  int choice = 0;  int service = 0;  string one = "Brokerage";  string two = "Retirement"; | * Non-Const Global Variables. |
| void ChangeCustomerChoice() {  cout << "Please enter the client's new service (1 = Brokerage, 2 = Retirement)" << endl;  cin >> service;  if (service == 1) {  cout << "You have selected: " << one << endl << endl;;  }  else if (service == 2) {  cout << "You have selected: " << two << endl << endl;  }  else {  cout << "Incorrect command. Please try again" << endl;  }  } | * No try catch for user input. * Use of global variables-can be manipulated. * Little use of authentication. * No sign of authorization. |
| void DisplayInfo() {  while (true) {  cout << "What would you like to do?" << endl;  cout << "DISPLAY client name & service (enter 1)" << endl;  cout << "CHANGE clients choice (enter 2)" << endl;  cout << "Exit program.. (enter 3)" << endl;  cin >> choice;  if (choice == 1) {  cout << "Client: " << user << endl;  cout << "Service: ";  if (service == 0) {  cout << "None" << endl << endl;  }  else if (service == 1) {  cout << one << endl << endl;  }  else if (service == 2) {  cout << two << endl << endl;  }  }  else if (choice == 2) {  ChangeCustomerChoice();  }  else if (choice == 3) {  exit(0);  }  else {  cout << "Incorrect command. Try again\n" << endl;  continue;  }  }  } | * Using true as condition in while loop. * Displays sensitive data without further authentication in choice 1. * Does not perform authentication of any means for choice 2 in changing client’s choice. * Variables one and two can be mutated because of being non-const. |
| void CheckUserPermissionAccess() {  while (true) {  cout << "You may enter <q> at any time to exit Authentication" << endl;  cout << "Please enter your username: ";  cin >> user;  cout << endl;  if (user == "q") {  exit(0);  }  cout << "Enter password: ";  cin >> password;  cout << endl;    if (password == "q") {  exit(0);  }  else if (password == "SNHU") {  cout << "Authentication Successful\n" << endl;  break;  }  else {  cout << "Authentication Unsuccessful. Please try again\n" << endl;  continue;  }  }    } | * No practice of least privilege. * No conditions for username. * Weak password. * Lack of try catch for user inputs. * Lack of advanced security checks such as two step verification. |
| int main() {  cout << "Created by John Brungard" << endl;  cout << "Welcome to SNHU Investment\n" << endl;  CheckUserPermissionAccess();  DisplayInfo();  ChangeCustomerChoice();  } |  |

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

* Non-Const Global Variables: Global variables can be mutated in any part of the application, making it difficult to remember every possible use. Because of this, debugging or maintenance involving global variables can be more tedious and possibly open security vulnerabilities that are harder to track down. Also because of the global variables scope, it can cause an out of memory issue.
* Try Catch for user input: The try statement allows you test a block of code for errors while it is executing. You can use a throw statement to customize the error which can be another way of validating user input. This can be incredibly useful to deter injection attacks if the program is connected to a SQL database.
* Little use of authentication: Authentication is important for controlling access. Without authentication, a user’s privileges cannot be determined to practice the principle of least privilege. With authentication, we want to confirm the identity of someone is legitimate. When an account is going to alter something significant, such as changing a client’s service, authentication should be used to prevent actions from a illegitimate user.
* No sign of authorization/practice of least privilege: Authorization determines access rights once a user is authenticated. Authorization should follow the principle of least privilege in which users are given just enough rights to perform their role successfully. In the current program, no real authorization is present meaning all users have the same access rights. This can be problematic if there is a security breach as the damage done can be higher compared to if authorization best practices were present.
* Using the true statement as a condition in while loop: Using while(true) can potentially be a security vulnerability if breaks are not implemented correctly as it will cause the loop to run indefinitely or crash the program is the loop consumes too much resource. While it is not technically terrible practice to use, caution should be used to when using true as the condition to avoid crashing the program as crashes can give attackers valuable information about sensitive details or at the very least create temporary vulnerabilities.
* No conditions for username: While some may view the conditions for a username not as sensitive as a password, there are some important considerations involving conditions that should be met. Usernames should be unique. Usernames can pose security vulnerabilities if it is related to any personal information or sensitive information such as your password or your security question(s).
* Weak password: Weak passwords can be ones that are easy to guess or are related to personal information. Weak passwords that are short and do not contain a variety of characters can be vulnerable to a brute force attack or other hacking means. This can compromise an account and allow malicious users to cause harm to the user’s information.
* Lack of advanced security measures: The lack of advanced security measures such as two step verification can be the determination of whether a malicious user can access your account. Having an additional layer of security makes it harder for attackers to cause harm to your account. This makes a password by itself useless if the malicious user cannot defeat your 2nd layer of security.

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

* Non-Const Global Variables: One recommendation is to make the global variables constants if possible. This will not allow the variable’s value to be manipulated. We can do this by prepending the word const before the selected variables. We also can replace the global variables with local variables and have them reach other functions through return and argument statements.
* Try catch for user input: Put the sections of the code that involve user input or are suspected of causing an error in a try block. Next, set up a catch statement that will take any error using the statement catch(. . .). Finally, use throw 505 for custom errors so you can recover the program in a flexible manner to avoid a crash.
* Little use of authentication: Have users reverify their password or other means of information when they choose important decisions such as changing their service or logging in. Make users go through several different layers of authentication before continuing such as answering a security question or sending an SMS or email link. We can do this by checking user input for the correct credentials stored in a database or an application.
* No sign of authorization/practice of least privilege: During the SDLC, consider collecting requirements and designing a UML diagram or other diagrams to represent the different types of users and what they are allowed and not allowed to do within their certain role. For example, an admin should be able to change the type of service a user has while a customer may only be able to see their service and would have to contact the company or admin to select those changes. We would want to make sure that each user has the least amount of privilege to do perform this.
* Using the true statement as a condition in while loop: Consider using a sentinel value such as a user entering ‘q’ so that the program can be exited at the user’s choice while running the application. You can also choose a statement that makes sense in the code such as while(choice != 3). As stated before, it would be wise to provide breaks into the code to make sure the loop does not run indefinitely.
* No conditions for username: Consider usernames being at least 5 characters long and starting with a letter. Allow usernames to only use letters and numbers, meaning no special characters, spaces or symbols. Finally, make sure the username is unique and does not give away real names or other information to where people can guess your credentials.
* Weak password: Consider setting a minimum length of passwords such as 12 characters. Also try to use a combination of upper- and lower-case letters along with numbers and symbols. Make sure your password is significantly different than past passwords you have used and should be easy for you to memorize but harder for others to guess. Having your password not be a word found in a dictionary is also another way to create a stronger password.
* Lack of advanced security measures: Implement two step verification through the application through means of a security question or by receiving a link to the user’s phone and/or email address. If needed, users may be required to provide both or more forms of verification when logging in or making important changes to their account such as changing services.