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.

1. **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

`bool CheckUserPermissionAccess() {`
`  while (1) {`
`  string username;`
`  string password;`
`  cout << "Enter username: " << endl;`
`  cin >> username;`
`  cout << "Enter password: " << endl;`
`  cin >> password;`
`  if (password == "123") {`
`  return true;`
`  }`
`  else {`
`  cout << "Invalid password. Please try again." << endl;`
`  }`
`  }`
`  return false;`
`}`
	The username variable is never used, meaning every username presented will work if the password is correct. Additionally, the password is not very secure.
	
	We'll compare the username to "admin" and compare the password to "@SecurePassword123".
	
	Additionally, we'll add logic branches to show if the username, password, or both is incorrect.

`void ChangeCustomerChoice() {`
`  int clientChoice;`
`  int serviceChoice;`
`  cout << "Enter the number of the client that you wish to change" << endl;`
`  cin >> clientChoice;`
`  cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;`
`  cin >> serviceChoice;`
`  if (clientChoice > 0 && clientChoice < 6) {`
`  service[clientChoice - 1] = serviceChoice;`
`  }`
`}`
	This function creates no guardrails to prevent unexpected input. Additionally, there is no logic branch to make sure that serviceChoice is a given choice(1 or 2). For example, you can pick any integer and will be assigned as the service choice.
	
	We will implement a loop, with a try-catch branch to handle unexpected input. We will implement a logic branch to check if the serviceChoice is within the boundaries.

```
int main() {
  int userChoice;
  bool access;
  cout << "This program was
adapted from binary to C++ by
Cody Faircloth on April 6,
2024" << endl;
  cout << "Hello! Welcome to
our investment company" <<
endl;
  access =
CheckUserPermissionAccess();
  if (access) {
    while(1) {
      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;
      cin >> userChoice;
      cout << "You chose " <<
userChoice << endl;
      if (userChoice == 1) {
        DisplayInfo();
      }
      else if (userChoice ==
2) {

ChangeCustomerChoice();
      }
      else if (userChoice ==
3) {
        break;
      }
    }
  }
  else {
    cout << "Permission
Denied." << endl;
  }
  return 0;
}
```

This function creates no guardrails for unexpected input.

We will implement a try-catch block to handle all unexpected input

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

This project has several security vulnerabilities. The first major security vulnerability that persists between functions is the method of input processing. In main() and ChangeUserChoice(), integer values are taken as input. However, no guardrails are present to ensure that an integer is entered. When any value besides an integer is entered, an endless loop occurs since the input buffer is never cleared and the program can't convert a string to an integer. This produces unexpected behavior and can lead to exploitation.
Next, in ChangeCustomerChoice(), there is no check to ensure the service that is entered is valid. For example, you could enter any integer value as the service choice, and it will save that value into the vector. This creates unexpected behavior.
Finally, in CheckUserPermissionAccess(), the username variable is created and assigned but is never used. Therefore, if given any username with the valid password, access will be granted. Additionally, the password is very simple and unsecure. There is also an issue hard-coding in your password and username checks for security, but I believe this is beyond the scope of this project.

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

To handle unexpected input, try-catch blocks will be implemented in the relevant locations, in main() and in ChangeCustomerChoice(). Input will be taken in as a string and attempted to be converted to an integer. If unsuccessful, the catch block will execute, clearing the input buffer, ignoring limits, and prompting the user to enter a numeric value. To support the try-catch block, a loop will be added that will only break once accepted input is received.
To handle the unexpected input of any service choice other than 1 or 2, additional parameters on the if statement will be added. The if statement will check if the customer choice is in the vector, 1 – 5, and if the service choice is either 1 or 2. If both conditions are met the corresponding customer service will be updated. Additional logic branches will be implemented to provide feedback to the user regarding what information provided was incorrect. For example, if the user provides an incorrect service request, it will be conveyed to the user with an output statement. Similar statements will be output for invalid customer choices and if both choices are invalid.
To better authenticate users, a username will be checked as well. For this situation, it will be "admin". Additionally, the password will be updated to "@SecurePassword123" creating a more secure password less prone to brute force attacks. Feedback will be provided to the user if username is not found and if the password is incorrect. Further mitigation could be implemented to further secure user authentication by moving credentials to a database and using hashing and encryption to secure the password and username information. However, this is beyond the scope of this project. It is important to understand this vulnerability is present and can be mitigated in the future.