# CS 410 Project Two Security Report Template

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** |
| --- | --- |
| #include <iostream>  #include <string>  #include <cstdlib>  using namespace std;  string aHelloWelcomeTo = "Hello! Welcome to our Investment Company";  string aInvalidPasswor = "Invalid Password. Please try again";  string aWhatWouldYouLi = "What would you like to do?";  string aDisplayTheClie = "DISPLAY the client list (enter 1)";  string aChangeAClientS = "CHANGE a client's choice (enter 2)";  string aExitTheProgram = "Exit the program.. (enter 3)";  string aYouChose = "You chose ";  string aEnterYourUsern = "Enter your username: ";  string aEnterYourPassw = "Enter your password: ";  string a123 = "123";  string aClientSNameSer =  " Client's Name Service Selected (1 = Brokerage, 2 = Retirement)";  string a1 = "1. ";  string a2 = "2. ";  string a3 = "3. ";  string a4 = "4. ";  string a5 = "5. "; |  |
| string aEnterTheNumber =  "Enter the number of the client that you wish to change";  string aPleaseEnterThe =  "Please enter the client's new service choice (1 = Brokerage. 2 = Retirement)";  string aSelected = " selected option ";  string name1 = "Bob Jones";  string name2 = "Sarah Davis";  string name3 = "Amy Friendly";  string name4 = "Johnny Smith";  string name5 = "Carol Spears"; |  |
| int num1 = 1;  int num2 = 2;  int num3 = 1;  int num4 = 1;  int num5 = 2;  int CheckUserPermission(void) {  string str, pass;  cout << aEnterYourUsern << endl;  cin >> str;  cout << aEnterYourPassw << endl;  cin >> pass;  if (pass == a123) {  return 1;  }  return 2;  } | There is no check for Username, therefore it’s open for anyone if the password is correct. Invalid access, need to deploy Role base access control to prevent un-authorized access. |
| void DisplayInfo(void) {  cout << aClientSNameSer << endl;  cout << a1 << name1 << aSelected << num1 << endl;  cout << a2 << name2 << aSelected << num2 << endl;  cout << a3 << name3 << aSelected << num3 << endl;  cout << a4 << name4 << aSelected << num4 << endl;  cout << a5 << name5 << aSelected << num5 << endl;  } |  |
|  |  |
|  |  |
|  |  |
| void ChangeCustomerChoice(void) {  int client = 0, newChoice = 0;  cout << aEnterTheNumber << endl;  cin >> client;  cout << aPleaseEnterThe << endl;  cin >> newChoice;  if (client == 1) {  num1 = newChoice;  } else if (client == 2) {  num2 = newChoice;  } else if (client == 3) {  num3 = newChoice;  } else if (client == 4) {  num4 = newChoice;  } else if (client == 5) {  num5 = newChoice;  }  } | No validation checks for entered value, nor check for range or exception handling. This vulnerability causes buffer overflow |
|  |  |
| int main() {  cout << aHelloWelcomeTo << endl;  int choice = 0;  while(choice != 1) {  choice = CheckUserPermission();  if(choice != 1) {  cout << aInvalidPasswor << endl;  }  } |  |
| while (1) {  cout << aWhatWouldYouLi << endl;  cout <<aDisplayTheClie<< endl;  cout <<aChangeAClientS << endl;  cout << aExitTheProgram << endl;  cin >> choice;  cout << aYouChose << choice << endl; | Choice could cause buffer overflow if the user entered a value greater than max range. |
| if(choice == 1) {  DisplayInfo();  }  else if(choice ==2 )  {  ChangeCustomerChoice();  }  else if(choice == 3)  {  exit(0);  }  }  return 0;  } |  |

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

After analyzing the binary file provided and converting it to C++ code, two significant vulnerabilities were identified. The first issue is related to access control or authentication. The code lacks any validation for usernames, which could allow any user to access the system if a password exists. The second vulnerability is a buffer overflow, a common issue in C++. This occurs when a program writes more data to a buffer than its allocated capacity, potentially leading to unexpected behavior, crashes, or even exploitation by attackers to execute malicious code. Both vulnerabilities pose serious security risks and require immediate attention.

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

The recommendations to resolve these vulnerabilities are as follows.

1. **Authentication:** To ensure proper user authentication, we should assign a specific username and compare it against a stored value. Additionally, we can implement an algorithm to track the number of incorrect login attempts. If the user exceeds the allowed number of tries, the program should terminate the process immediately upon initiation. This approach enhances security by preventing unauthorized access and limiting brute-force attacks.
2. **Buffer Overflow:** To mitigate buffer overflow vulnerabilities, we should implement exception handling and expand the range of acceptable values. The input can be read as a string and then converted to the desired data type. If the value falls outside the expected range, the exception can be caught, and the process can be repeated until a valid value within the specified range is provided. This approach ensures robust input validation and prevents potential buffer overflow issues.
3. **Refactoring:** To optimize the program, we can refactor the code to enhance its extensibility and maintainability, ensuring the system is easier to update and manage in the future.