# 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 approvedUsers[] = {"User","user","", " "};  string approvedPasswords[] = {"123"}; | -Authentication pairs in string text. Allows a reverse to be able to see this information.  -Nonbounded information |
| string clientList[] = {"Bob Jones","Sarah Davis","Amy Friendly","Johnny Smith","Carol Spears"};  int clientChoice[] = { 1,1,2,1,2 }; | -Personal Identifiable Information Present in String Text  - Nonbounded choice and user  -Arbitrary numerical choice |
| cout << "Enter the number of the client you want to change" << endl;  cin >> clientNum;  clientNum = clientNum - 1;  cout << endl;  cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl;  cin >> newChoice;  clientChoice[clientNum] = newChoice; | -Client number not clearly defined and utilizes an arbitrary numbering system  -Choice of service not defining parameters for input (Overflow errors)  -No Confirmation of actions |
| if (find(begin(approvedUsers), end(approvedUsers), user) != end(approvedUsers))  { errorValue = 0;  } else {  errorValue = 1;  }  if (find(begin(approvedPasswords), end(approvedPasswords), password) != end(approvedPasswords))  {  errorValue = 0;  } else {  errorValue = 1;  } | -While it does check for user and password it does not check that they are bound to each other. Allows a user to enter the password of another entity and get access. |
| string user;  string password; | -User and password variables exist as basic strings instead of secure |

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

Identifying the authentication arrays as a vulnerability highlights the ease of access for a third party to be able to obtain that information. A third party who chooses to reverse engineer the application whether legal or illegal, will easily be able to extrapolate the strings and connect the pieces of information.

The password and username arrays are not bound to each other. As such they don’t exist in the same purview. When changing one it allows for information to be orphaned more easily or allow for unlinked access.

Client List as a plain text string allows a third party to identify who the clients are. In this manner this makes personal identifiable information available as plain text and easily extrapolated through reverse engineering.

The client list and choices are not bound to each other. While not as much of a threat as the username and password, this can still allot for information mismatch more easily than if the information is bound to each other.

The client Choice existing as numerical values makes the system more susceptible to information mismatch if choices are to expand or change. In a financial institution this can lead to mismanagement of client funds if the value for choice one is changed but the client record is never updated.

When updating a client choice, choosing a client by number, specifically by how they appear on a list is an inefficient manner. There’s no indication as to who is which number in this listing method and does not allow for changes should new clients be added and old ones leave.

While choosing the services seems obvious to enter a one or two, the end-user may not see it so. As such it leaves it to open interpretation as what two put in for the services. Enter a non-integer at this point leads to an overflow error.

When completing actions, it is not clear whether the action completed or not. This issue provides questionable doubt that the changes made are saved and completed. This can also lead to actions being repeated or changes not occurring due to errors and not being forewarned. Which again can lead to mismanagement issues.

The username and password check are on the right track but has a critical flaw in the way it checks for the information. While the check for password and username is performed, it looks for the existence of each separately and not that they are tied to each other. As such a user can enter one person’s username and entirely different persons password and gain access.

When entering the user and password when prompted these values get stored as basic strings. This can lead to that information being scraped and causing a data breach to occur.

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

For the identification arrays, we can employ a way of securely storing the information. This can either be done by use of an SQL database that can be referenced, an external document, or utilizing an obfuscation method to store the strings. Ideally not keeping the information hard coded in the application and external may be the best practice, but if that is not a feasibility, one of the aforementioned methods can substitute.

Username and password can link to each other by utilizing a connection method such as a map or pair. By doing so the data between who the user is and what the password is are connected thus allowing for only the password associated with that user to be a proper authentication.

To prevent PII data from being accessible, when the string is stored, it can undergo a process to encrypt it. This method could be as simple as using a cipher on the string and storing that. Thus, only the ciphered text is available.

The client list and choices can as well be a paired method that can bound the two together. This will ensure that when information is updated for one element, the other side receives that update as well.

The client choice as a number can still be using numbers to enter a choice. The background should have that number converted to a readable text to identify what that choice is should the numbering scheme change in the future. While the numbers may change, the service may not.

Selecting a client by an arbitrary number can be alleviated by assigning an account ID number to the client and that being the means by which they are selected. As such this information will also need to be bound to the client’s name and choices.

We can clarify the instructions when choosing what services to assign by placing a print statement that clarify the user to enter an integer number. This will prevent confusion for the user as well as prevent overflow errors that would occur from entering an incorrect datatype value.

Completing actions can be corrected in a much similar manner by implementing a method to check for the changes and if they exist then print a complete statement or error statement if changes have not occurred.

Checking the username and password can be handled with how we tie together the records and implement a better method that checks for both the username and password and return true when and only when both the criteria match and are correct.

When accepting the username and password as input, they get stored as basic strings. We can change this to store as secured string to prevent any type of memory issue where that information could accessed.