# Daulton Truett

06/23/2024

CS-410: Software Reverse Engineering

Southern New Hampshire University

# CS 410 Project Two Security Report

## 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.

*‘Project2\_DT.cpp - Global’*

| **Block of C++ Code** | **Identified Security Vulnerability** |
| --- | --- |
| #include <iostream>  #include <string>  //Global variables <nums>  int num1 = 1;  int num2 = 2;  int num3 = 1;  int num4 = 1;  int num5 = 2;  std::string name1 = "Bob Jones";  std::string name2 = "Sarah Davis";  std::string name3 = "Amy Friendly";  std::string name4 = "Johnny Smith";  std::string name5 = "Carol Spears"; | <- VULNERABILITY  Global variables. |

***‘main()’***

| **Block of C++ Code** | **Identified Security Vulnerability** |
| --- | --- |
| while(choice != 3){      std::cout << "What would you like  to do?" << std::endl;  std::cout << "DISPLAY the client  list (enter 1)" << std::endl;      std::cout << "CHANGE a client's  choice (enter 2)" << std::endl;      std::cout << "Exit the program..  (enter 3)" << std::endl;      std::cin >> choice;      std::cout << "You chose " <<  choice << std::endl;      if(choice == 1){        DisplayInfo();      }else if (choice == 2){        ChangeCustomerChoice();      }    } | <- VULNERABILITY  No input validation or exception handling when receiving input from the user with assignment to an ‘int’ variable. |

***‘CheckUserPermissionAccess()’***

| **Block of C++ Code** | **Identified Security Vulnerability** |
| --- | --- |
| std::string password = "123";  std::string userName;  std::string userPassword;  std::cout << "Enter your username: "  << std::endl;  std::cin >> userName;  std::cout << "Enter your password: "  << std::endl;  std::cin >> userPassword;  if(userPassword == password){  return 1;  }; | <- VULNERABILITY:  Hardcoded sensitive data.  <- VULNERABILITY  Lack of encryption after receiving input for sensitive data.  <- VULNERABILITY  Comparing unencrypted sensitive data (user password) received via input with ‘std::cin’ with unencrypted hardcoded sensitive data (password) for authentication purposes. |

***‘ChangeCustomerChoice()’***

| **Block of C++ Code** | **Identified Security Vulnerability** |
| --- | --- |
| std::cout << "Enter the number of the  client that you wish to change" << std::endl;  std::cin >> changeChoice;  std::cout << "Please enter the  client's new service choice (1 = Brokerage, 2 = Retirement)" << std::endl;  std::cin >> newService; | <- VULNERABILITY  No input validation or exception handling when receiving input from the user with assignment to an ‘int’ variable.  <- VULNERABILITY  No input validation or exception handling when receiving input from the user with assignment to an ‘int’ variable. |

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

**VULNERABILITY:** Global variables.

Global variables disclose potentially valuable information about the architecture of the application if/when the application is reverse engineered. Malicious actors can utilize this important information, especially if global variables are associated with sensitive data, to leverage an attack or unauthorized access to a system.

**VULNERABILITY:** *Lack of input validation and exception handling.*

When receiving input via ‘std::cin’ for variable assignment of type ‘int’, it is important to validate the input data. Whether malicious or accidental, the potential of overflow/underflow is present when accepting input external to the application as a user can input a value larger than the variable can handle. This can lead to data corruption, malicious pointers to execute arbitrary code, or in the event of a type mismatch such as assigning a ‘char’ to an ‘int’ from input, unexpected program behavior/crash.

***VULNERABILITY:*** *Hardcoded sensitive data (password).*

Sensitive data such as passwords must not be hardcoded in the code as it allows malicious users to examine the codebase with reverse engineering techniques to gain access to that important information. This poses a significant security risk as unauthorized users may utilize this information to gain access to accounts and/or systems they are not authorized for. The potential damage is vast with potential losses of data, services, and user trust.

**VULNERABILITY:** *Lack of sensitive data encryption.*

If sensitive data is going to be received from a user, such as for authentication purposes, it must be encrypted to prevent potential malicious users from gaining access to the sensitive data. This data will be visible within the application, and if stored/used in plaintext, the sensitive data is not secure and can be used to leverage access to a user account leading to loss of data/services, corruption of data/services, loss of user trust, privilege escalation, and denial of service.

**VULNERABLITY:** *Authentication with unencrypted and hardcoded sensitive data.*

The use of authenticationfor user accounts must be strict and follow security best practices to ensure that unauthenticatedusers cannot gain access to a system and/or resources. When authenticationrelies on hardcoded data and user provided data that is unencrypted, malicious actors have the ability to access that sensitive data with reverse engineering techniques and network monitoring (man in the middle) techniques to obtain the sensitive data to bypass authentication. This leads to loss of data/services, corruption of data/services, privilege escalation, a loss of user trust, and denial of service.

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

**VULNERABILITY:** Global variables.

The use of global variables can be fixed by simply moving the variables to local functions and passing the variables as arguments to other functions that may need to access them.

**VULNERABILITY:** *Lack of input validation and exception handling.*

After receiving external data, whether from a user or file, validate that the input is of the correct type of the variable the data is being assigned to and that the variable can hold the amount of data received. Code that receives input data should be with a ‘try-catch’ block to ensure that if/when the data causes an exception, the error is caught and handled appropriately.

***VULNERABILITY:*** *Hardcoded sensitive data (password).*

Remove all hardcoded sensitive information. The best method for utilizing sensitive data in an application is to use services such as ‘Oauth’ and/or an API with encrypted data to retrieve the data for temporary use only when required.

**VULNERABILITY:** *Lack of sensitive data encryption.*

Utilize trusted and tested encryption algorithms on all data deemed as ‘sensitive’ to ensure that it cannot be discovered and utilized by malicious actors. Once the sensitive data has been encrypted, the data can be used temporarily as needed or stored in a DB in its encrypted state for security best practices.

**VULNERABLITY:** *Authentication with unencrypted and hardcoded sensitive data.*

The process of authenticating and authorizing users is an important step that must ensure security best practices are followed. When authenticating a user with externally provided input, the information must be encrypted and retrieved in a secure manner, such as with an API call or the use of a system such as ‘Oauth’. Authentication must not be done with two application variables that hold sensitive data in plaintext. My approach to this issue would be to search for the user within a DB via an API call with a provided name, username, email, etc.. Then I would retrieve the encrypted password and compare that to the encrypted password provided by the user via their input without storing the sensitive data in the application, then proceed with user authorization.