CS 410 Project Two: Security Report

Security Vulnerabilities		
	Block of C++ Code	**Identified Security Vulnerability**
1	int num1 = 1, num2 = 1, num3 = 1, num4 = 1, num5 = 1; string name1 = "Bob Jones", name2 = "Sarah Davis", name3 = "Amy Friendly", name4 = "Johnny Smith", name5 = "Carol Spears";	User values are hardcoded in the program.

2	int CheckUserPermissionAccess() {
	string password;
	cout << "Enter your password:"
	<< endl;
	cin >> password;
	
	if (password == "123") {
	return 1;
	}
	else {
	return 2;
	}
	}
		Username is not used, and the password is hardcoded.

3	cout << "1. " << name1 << " selected option " << num1 << endl; cout << "2. " << name2 << " selected option " << num2 << endl; cout << "3. " << name3 << " selected option " << num3 << endl; cout << "4. " << name4 << " selected option " << num4 << endl; cout << "5. " << name5 << " selected option " << num5 << endl;	Output is not dynamic, depends on hardcoded values.
4	cout << "Enter the number of the client that you wish to change" << endl; cin >> changechoice;	Input is not validated.

5	cout << "Please enter the client's new service choice (1 = Brokerage, 2 = Retirement)" << endl; cin >> newservice;	Input is not validated.

6	// Update service for client 1
	if (changechoice == 1) {
	num1 = newservice;
	}
	// Update service for client 2
	else if (changechoice == 2) {
	num2 = newservice;
	}
	// Update service for client 3
	else if (changechoice == 3) {
	num3 = newservice;
	}
	// Update service for client 4
	else if (changechoice == 4) {
	num4 = newservice;
	}
	// Update service for client 5
	else if (changechoice == 5) {
	num5 = newservice;
	}
		Depends on hardcoded values and magic numbers.

7	cout << "Enter your username:" << endl; cin >> username;	Username is never validated or used.
8	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 >> answer; cout << "You chose " << answer << endl;	User menu selection input is not validated.

Security Vulnerability Explanation and Recommendations

1. Explanation: The customer information is hardcoded into the program. This is an issue because is exposes the customer information in the source code and can be extracted from the binary. Additionally, it limits the program from being able to act dynamically, adding or removing customers.

   Recommendation: This can be fixed by creating a class and turning customers into an object. A list of customers can then be kept and changed as needed.

2. Explanation: The entered username is not used in the permission check process. This is an issue because then anyone can gain access with just the password. Additionally, the password is hardcoded. That means anyone can view the password in the source code or extract it from the binary.

   Recommendation: This can be fixed by verifying the username along with the password and not hardcoding any passwords into the program.

3. Explanation: The display output is not dynamic and depends on those exact values to exist. This is an issue because it limits the use of the program and will break if any value is missing.

   Recommendation: This can be fixed by utilizing the list of customer objects to dynamically generate the output for the customers.

4. Explanation: The input is not validated for selecting a customer. An integer is expected, so if something else is entered it can cause the program to crash. Additionally, even if an integer is selected, there is no validation the customer actually exists for that value. This would also cause unexpected behavior.

Recommendation: This can be fixed by first using the built in C++ methods to check that the input is good, i.e., an integer was entered. From there, utilizing the list of customer objects, you can check that the integer entered falls within the size of the customer list.

5. Explanation: The input is not validated for selecting a new service choice. An integer is expected, so if something else is entered it can cause the program to crash. Additionally, even if an integer is selected, there is no validation that a service choice is actually associated with that value. This would also cause unexpected behavior.

   Recommendation: This can be fixed by using the built in C++ methods to check that the input is good, i.e., an integer was entered. Then you can check that the integer entered falls within the expected range for service choices.

6. Explanation: The logic for updating the service choice for the specified customer requires using hardcoded values and checking each customer to see if they were selected. Then the service choice can be updated for that customer.

   Recommendation: This can be addressed by using the list of customer values. Instead of a magic number being used to represent a customer, a unique id number is created each time a new customer object is instantiated. You can then use the built in C++ methods for vectors to find the correct customer and update their service choice.

7. Explanation: The username is never verified after entering it. This is an issue because it allows anyone to login with just a password.

   Recommendation: This can be fixed by including the username entry with the CheckUserPermissionAccess method and then validating that the username exists, and that the password matches for that username.

8. Explanation: The input is not validated for selecting a menu option. An integer is expected, so if something else is entered it can cause the program to crash. Additionally, even if an integer is selected, there is no validation that a menu option is actually associated with that value. This would also cause unexpected behavior.

   Recommendation: This can be fixed by using the built in C++ methods to check that the input is good, i.e., an integer was entered. Then you can check that the integer entered falls within the expected range for menu options.