# CS 410 Project Two Security Report

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** |
| --- | --- |
| checkUserPermissionAccess() | Hardcoded credentials (ADMIN\_USER & ADMIN\_PASS) allow reverse engineering and unauthorized access. |
| main() login handling | No login attempt rate-limiting (brute force risk) |
| cin >> clientIndex; and cin >> newChoice; | No input validation allows invalid or out-of-range data (potential crashes and logic errors). |
| changeClientChoice() | Lack of input validation for index/service values can cause buffer overflow/out-of-range errors |
| clients vector and user input handling | No sanitization of input, allowing potentially bad data or out-of-range vector access |
| cin >> choice; | Input handling without error checking can cause buffer overflow or leave variables in bad state. |
| Application logic as a whole | No audit logs or role-based access all authorized users can do all operations without trace. |

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

**1. Hardcoded Credentials**

In checkUserPermissionAccess(), the credentials ("admin", "123") are directly embedded in the source code. This is a serious issue because an attacker who decompiles the binary can retrieve them easily. It also cannot scale if passwords must change.

**2. Unlimited Login Attempts**

The original version of the program allowed unlimited login attempts. This enabled brute force attacks where an attacker could repeatedly attempt password guesses without restriction.

**3. Input Validation Issues**

Throughout the program, inputs like clientIndex, newChoice, and menu choice were read without verifying that they were integers or in a valid range. Invalid inputs could cause buffer overflow or the program to behave unpredictably.

**4. Sensitive Data Hardcoded into Binary**

The clients vector contains client names and services directly in the source. Anyone who can read the binary can view this sensitive data. This poses privacy and compliance risks.

**5. Poor Logging and Role Management**

There is no log file for failed logins or client data changes, and any user who passes login can change client data. This lacks traceability and role separation — serious security and accountability risks.

**Recommendations for Fixing Vulnerabilities**

**1. Implement Secure Credential Storage**

Use hashed and salted passwords, store them outside the code in a secure configuration file or database, and authenticate against that securely.

**2. Implement Login Rate-Limiting**

Allow a limited number of login attempts (e.g. 3). Implement account lockout or a back-off period between failed login attempts to slow brute force attacks.

**3. Improve Input Validation**

Always check cin.fail() after reading input. Validate that all integer inputs (e.g. clientIndex, newChoice, choice) fall within expected ranges before using them.

**4. Secure Client Data Storage**

Store client information in an encrypted file or a secured database, not compiled into the program. Implement proper access controls so only authorized users can read this data.

**5. Implement Logging and Roles**

Add logging for successful/failed login attempts and for all client data changes. Implement role-based access so only specific roles can change client choices, and log all updates for auditing.