# CS 410 Project Two Security Report

## Merrik Wright

CS410

## 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** |
| --- | --- |
| |  | | --- | |  |  |  | | --- | | cin >> username | | |  | | --- | |  |  |  | | --- | | No input validation | |
| if (username == "admin") | Hardcoded credentials. |
| cout << Name/ID info... | |  | | --- | |  |  |  | | --- | | Exposure of sensitive data | |
| cin >> option | Input not validated |

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

This line of code performs a simple string comparison to determine if a user should be granted administrative access. If the input matches the hardcoded string "admin", the user is given a higher permission level (in this case, a return value of 2). This is a significant vulnerability because it completely bypasses any real authentication system. There is no password required, no verification against a user database, and no encryption or hashing involved. Any user who knows or guesses the word “admin” can gain elevated access rights, potentially exposing sensitive functions or data.

This is a classic example of insecure access control. By relying solely on user input with no verification mechanism, the application opens itself to impersonation and privilege escalation attacks. In a real-world scenario, this could allow an unauthorized user to make critical changes, retrieve protected information, or disrupt application behavior.

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

* Input sanitation (getline, type check, regex)
* Replace cin with safer input
* Replace hardcoded checks with credential system
* Do not hardcode sensitive data in source files

Implement a secure authentication system that verifies both username and password. Passwords should never be hardcoded and should be securely stored in hashed form (e.g., using SHA-256 or bcrypt). Access should be granted only after checking the entered credentials against a secure backend (e.g., a user database or authentication service). Additionally, use role-based access control (RBAC) to define user capabilities based on permission level.