# CS 410 Project Two Security Report Template

# Aaron Ciminelli

SNHU

CS-410-R1913 Software Reverse Engineering 23EW1

JOE L. PARKER, (B.S, M.S.I.T, Ed.D (c))

October 11th , 2023

# 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** |
| --- | --- |
| vector<int> serviceChoice{ 1, 2, 1, 1, 2 }; | Use of Global Variables |
| string name1 = "Bob Jones"; …. | Hardcoded customer names |
| int num1 = serviceChoice[0]; …. | Array Indexing without Bounds Checking |
| void displayInfo() { | Information Disclosure |
| void checkUserPermission() { | Weak Authentication Mechanism |
| cin >> password; | No input length check |
| if (password == 123) | Hardcoded password |
| while (loopMenu){ | Potential for Infinite Loop |
| getline(cin, user); | Buffer Overflow Potential |

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

The CPP file contains several security vulnerabilities, one of which is using hardcoded customer names in the application's code. This practice can put sensitive user information at risk and make it easier for malicious entities to carry out targeted scams or social engineering attacks. To ensure greater security, storing such data in encrypted databases with restricted and authorized access would be better.

It is essential to avoid using hardcoded indices to access elements within vectors, such as the serviceChoice vector. Doing so can lead to out-of-bounds access, causing unpredictable behavior or application crashes. A safer alternative would be to check the bounds before accessing any array or vector indices. This ensures that any out-of-range indices are handled gracefully.

Additionally, the changeCustomerChoice function needs input validation, which can result in incorrect behavior or out-of-range access if invalid client or service numbers are provided. The solution to this issue is simple: inputs should always be validated to ensure they conform to the expected ranges or the required formats.

The function "displayInfo" poses a security threat as it can reveal sensitive customer information, such as their name and choices, without proper authentication and authorization measures. To mitigate this risk, it is essential to implement robust authentication and authorization mechanisms and encryption of sensitive data at rest and in transit. Additionally, the "checkUserPermission" function's dependence on a hardcoded and easily guessable password raises concerns. Replacing this weak authentication mechanism with a more robust method and rate limiting or account lockout strategies is crucial to discourage brute force attacks.

The main function has a potential buffer overflow due to the `getline(cin, user)` statement not checking input length. To prevent this, input lengths should be limited to a predefined size.

The software system does not have encryption or secure transmission, which makes transmitted data vulnerable to interception. Therefore, it is necessary to use encryption and secure transmission protocols. Additionally, the system lacks a logging and monitoring mechanism, meaning there is no way to trace suspicious activities. This highlights the importance of employing comprehensive logging and monitoring solutions.

The repetition of code blocks in software development complicates maintenance and increases the risk of security oversights. The DRY (Do not Repeat Yourself) principle consolidates repetitive code blocks into reusable functions, improving maintainability and reducing security risks.

Programs that lack proper error handling can behave unpredictably. Recursion, particularly from user input, can cause stack overflows. These issues highlight the need for robust error handling and a shift from using recursion in user input validation. In addition, infinite loops, especially those that users can trigger, can be exploited for denial-of-service attacks. Therefore, it is crucial to ensure that loops can terminate under certain conditions.

The usage of global variables in the system raises concerns, as other code sections may accidentally access or change them. To address this issue, it is recommended to minimize or eliminate global variables instead of using local variables or encapsulating them within classes. Additionally, the absence of a Role-Based Access Control (RBAC) system poses challenges in managing user access. To enhance both security and organizational structure, it is advisable to implement an RBAC system, which ensures users are granted access based on their roles.

**Resources:**

Armerding, T. (2020, March 21). *Legacy vulnerabilities: How to find and Remediate them:* . Legacy Vulnerabilities: How to Find and Remediate Them | Synopsys Blog. https://www.synopsys.com/blogs/software-security/how-to-deal-with-legacy-vulnerabilities.html#:~:text=Standards%20such%20as%20HIPAA%20,lead%20to%20fees%20and%20fines

Higgins, S. (2020, September 17). *Finding and fixing C++ vulnerabilities*. SecureCoding. https://www.securecoding.com/blog/finding-and-fixing-c-vulnerabilities/

ISACA. (2023, September 11). *SQL databases and Data Privacy: Features to know -*. MSSP Alert. https://www.msspalert.com/post/sql-databases-data-privacy

NIST. (n.d.). *Legacy environment - glossary: CSRC*. CSRC Content Editor. https://csrc.nist.gov/glossary/term/Legacy\_Environment#:~:text=A%20Custom%20environment%20containing%20older%20systems%20or%20applications,to%20be%20able%20to%20communicate%20with%20other%20systems.

Taylor, T. (2023, June 19). *8 common security risks associated with cloud-based DevOps and how to mitigate them*. Amazic. https://amazic.com/8-common-security-risks-associated-with-cloud-based-devops-and-how-to-mitigate-them/

VanDerStock, A., Glas, B., Smithline, N., Gigler, T., Manico, J., & Mackowski, J. (2021, September). *A06:2021 – vulnerable and outdated components*. A06 Vulnerable and Outdated Components - OWASP Top 10:2021. https://owasp.org/Top10/A06\_2021-Vulnerable\_and\_Outdated\_Components/