**Journal**

Justin Starr

Department of STEM

CS 410 – Software Reverse Engineering

Dr. Philomena Ogoh

October 13, 2024

**Journal**

The following is a short analysis what a security vulnerability is, what kinds of vulnerabilities would be identified in C++ code, why someone would be looking for vulnerabilities during legacy to C++ conversion rather than during testing, and how someone would determine the appropriate fix to a security vulnerability.

A security vulnerability is a flaw in software code or a misconfiguration in a system that enables an attacker to gain unauthorized access to a system or network, which can then be used to conduct malicious activity (Snyk, 2024). For example, an attacker could potentially manipulate or gain control of a program’s execution by exploiting a buffer overflow vulnerability, which could also potentially lead to the exposure of sensitive or private data. The different kinds of security vulnerabilities that can be identified in C++ code include uninitialized variables, buffer overflows, null pointer dereferencing or dangling pointers, and improper error handling (Ikhlaq, 2023). Not initializing variables can lead to undefined behavior, and developers should never assume that a variable starts with a zero or null value. A buffer overflow occurs when invalid memory locations are accessed, for example when attempting to access a memory location that is out of bounds by writing to it. Null pointer dereferences or dangling pointers vulnerabilities occur when a program is attempting to access an object or memory location that doesn’t exist, which typically happens when objects are de-allocated or deleted, but their corresponding pointer is never changed. Also, not handling errors or ignoring them can present security issues, such as causing a program to terminate abruptly, or any other type of undefined or unexpected behavior. The reason why these can be identified in C++ code is because they are all vulnerabilities that can be introduced because they are written by developers. If they neglect to pay attention to how well they are writing their code and keeping security in mind, they will likely introduce one or more of the previously mentioned security vulnerabilities.

One reason why someone would be looking for vulnerabilities during legacy to C++ conversions rather than during testing is if a program or a portion of a program has been discovered to contain one or more security vulnerabilities that have been exploited, and the original source code or software documentation no longer exists. By performing the legacy to C++ conversion, specific patterns or behaviors that occur during a program’s execution can be uncovered to help identify the vulnerabilities. Also, not all testing tools can detect all vulnerabilities, thereby making the conversion process from legacy to C++ code essential in detecting hidden issues or security flaws that might not always be evident when performing regular testing. Furthermore, finding and fixing vulnerabilities before testing phases can lead to cost reduction and better utilization of resources by finding potential vulnerabilities early on.

For someone to determine the appropriate fix to a security vulnerability, a number of steps should be taken. Essentially, it is the process known as vulnerability remediation and includes steps, such as discovery, prioritization, remediation, and monitoring (HackerOne, 2021). First, the vulnerability must be clearly identified to know what approach should be taken to address it. Vulnerabilities should be assessed to determine their severity, especially when multiple vulnerabilities might exist. This helps ensure that vulnerabilities that pose the most significant risk are dealt with first, on a priority basis. Research should be conducted to help determine the appropriate solutions for fixing the vulnerability, including fixes that may already exist from software vendors, security companies, or other trusted and reputable sources. Next, the solution for the vulnerability should be developed if not already, and implemented, which might include re-writing the software code that may have introduced the vulnerability, updating existing configurations, and ensuring security controls are in place and also properly configured. Lastly, the solution should be tested to ensure the vulnerability has been mitigated before being re-deployed into a live environment, and once it is, continuous monitoring should be conducted.

**References**

HackerOne. (2021, September 30). *Remediation | A Step-by-Step Guide*.

<https://www.hackerone.com/vulnerability-remediation-step-step-guide>

Ikhlaq, D. (2023, July 9). *Demystifying C/C++ Vulnerabilities: A Comprehensive*

*Guide*. Vulert. <https://vulert.com/blog/demystifying-c-c-vulnerabilities-a-guide>

Snyk. (2024). *Security Vulnerability: Types and Remediation*.

<https://snyk.io/learn/security-vulnerability-exploits-threats/>