**6-1 Journal**

Aaron Ciminelli

SNHU

CS-410-R1913 Software Reverse Engineering 23EW1

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

October 3rd, 2023

* **Define: What is a security vulnerability?**
* **Purpose: Why would you be looking for vulnerabilities during legacy to C++ conversion rather than during testing?**
* **Solutions: How do you determine the appropriate fix to a security vulnerability?**

Security vulnerabilities refer to weaknesses or flaws in a system's design, implementation, or operation that could give unauthorized access, manipulate data, or cause system failures (NSIT, n.d.). If a vulnerability is present, an attacker could gain unauthorized access to data or system and even crash it. Several ways a C++ vulnerability could occur include incorrect handling of user input, misuse of pointers, or improper use of libraries and functions. Hence, software developers must be vigilant and look out for security vulnerabilities throughout their coding process.

During the conversion process, it's especially important to look for vulnerabilities. The term legacy software refers to old software that's hard to update with today's technology. This code often contains vulnerabilities and weaknesses that can easily be exploited by attackers. In addition, it may be incompatible with modern technology and systems due to its age and lack of maintenance. Over time, cyber threats have become more sophisticated, so vulnerabilities that might have been overlooked in the past could now be a big deal.

According to Higgins (2020), it is important to take proactive measures to address vulnerabilities when converting legacy code to modern C++. This helps prevent any security issues from the old code from carrying over to the updated system. To ensure the new system is strong against potential security threats, I use both static analysis tools and manual code reviews to identify any lurking vulnerabilities within the legacy code. Static analysis tools scan the codebase for known vulnerabilities and coding standard violations, while manual code reviews provide a deep-dive analysis by experienced developers who can spot complex vulnerabilities that automated tools may miss. This blended approach fosters a culture of security awareness among the development team and ensures a thorough examination of potential security risks. By taking these measures, we can guarantee that the modernized code not only functions well, but is also secure enough to withstand contemporary security challenges.

When I review legacy code for security vulnerabilities, I focus on several key areas. Firstly, I pay close attention to pointer usage to identify memory leaks, buffer overflows, and dangling pointers. Additionally, I follow the recommendations outlined in OWSAP's 2021 documentation on "Vulnerable and Outdated Components," including validating all user input before incorporating it into the application (VanDerStock et al., A06:2021 – vulnerable and outdated components 2021). This helps prevent injection attacks and other input-related vulnerabilities. Finally, I make sure to use libraries and functions appropriately and carefully avoid any deprecated or inherently insecure code.

As soon as I have identified a vulnerability, I take action to resolve it. My approach to fixing vulnerabilities depends on the nature of the vulnerability, but I always test the fix thoroughly before deploying it.

Additionally, I evaluate the overall security of the system when fixing a vulnerability. If, for example, a vulnerability is fixed in one part of a system, but it can still be exploited in another part, then the overall security of the system has not improved.

When converting code to C++, I can address security vulnerabilities, ensuring that the new system is secure. Taking this step is essential in order to protect systems and data from unauthorized access.

**Reference:**

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/

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.

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/