Aspen Blue Koreny-Crawford

Southern New Hampshire University

Secure Coding

20 August 2025

Portfolio Reflection

Throughout this course, I gained a deeper understanding of secure coding practices and their role in the software development lifecycle. Key lessons emphasized the importance of adopting secure coding standards, such as the SEI CERT C++, which help prevent vulnerabilities by integrating security into every stage of development, including design, testing, and deployment. This proactive approach is more cost-effective than addressing issues at the last minute.

I have been privileged to learn how to evaluate risk and balance mitigation against cost using tools like threat matrices. While some vulnerabilities, such as SQL injection, require immediate attention, minor issues can be deprioritized. The cost of prevention is significantly lower than the cost of responding to breaches.

A critical concept that reshaped my perspective is Zero Trust. Traditional security models often relied on a “castle-and-moat” approach, where anyone inside the network perimeter was trusted. However, as Kueh (2020) explains, this model is obsolete in today’s IT environment, where users access systems from multiple devices and locations. Zero Trust instead operates on a “never trust, always verify” model. For example, device trust ensures that only approved and compliant hardware is allowed to connect to the network, while user trust requires robust methods such as multifactor authentication and conditional access policies. At Green Pace, adopting Zero Trust could mean requiring password-less logins or biometric authentication for developers accessing sensitive code repositories. Similarly, transport/session trust through least-privilege access ensures that a developer working on one subsystem cannot automatically access all others. These measures directly reduce the risk of insider threats and misconfigurations while keeping the developer experience practical.

Throughout the Green Pace projects, for example, I also observed how these policies and frameworks, such as Triple-A (authentication, authorization, and accounting), provide a structure for secure development. Authentication verifies identity, authorization limits actions, and accounting ensures all activity is logged for auditing. These align naturally with Zero Trust principles by reinforcing verification at every layer. To remain effective, I recommend that these policies be actively maintained and enforced through automation, regular audits, and ongoing developer training so that they evolve alongside emerging threats.

In conclusion, this course, Secure Coding, has enhanced my knowledge and ability to prioritize security in software development. By adopting secure coding standards, evaluating risks carefully, applying Zero Trust principles, and implementing stronger policies, developers can create software that is resilient and trustworthy. Security should not be left to the end but must remain a continuous focus across the lifecycle, ensuring both protection and long-term trust in our systems.

References

Kueh, T. (2020, January 17). *A practical guide to zero-trust security*. Threatpost English Global threatpostcom. https://threatpost.com/practical-guide-zero-trust-security/151912/

Seacord, R. C. (2013). Secure Coding in C and C++ (2nd ed.). Pearson Technology Group. https://mbsdirect.vitalsource.com/books/9780132981972