Edwin Jones

CS 405: Secure Coding

June 28, 2025

Journal: Portfolio Reflection

Throughout this course, one of the most consistent themes has been the importance of adopting secure coding standards early in the development lifecycle. As emphasized in both the textbook *Secure Coding in C and C++* by Robert Seacord and throughout the course assignments, waiting until the end of development to address security concerns is a critical mistake. For example, the module on buffer overflows and the accompanying coding activities made it clear that careless memory management—if not caught early—can introduce vulnerabilities that are difficult and expensive to fix later. By establishing secure coding guidelines up front and reinforcing them with practices like code reviews and static analysis, developers can build security into the software from the ground up rather than trying to patch holes later.

In assessing security decisions, it is essential to weigh the cost and benefit of each mitigation strategy. The course discussions and case studies, such as the one focused on Triple A and defense in depth, made it clear that no organization has unlimited resources. Thus, developers and managers alike must be able to identify which risks are most critical and which defenses are most effective. The use of threat modeling and risk matrices helped us consider both the likelihood and impact of different vulnerabilities, providing a structured way to prioritize remediation efforts. For instance, a buffer overflow in user input handling is typically a higher priority than a deprecated API usage warning that has minimal impact.

Another major takeaway from this course was the application of the zero-trust model. The “no one is safe” philosophy fundamentally changes how I think about software architecture and user access. Rather than assuming anything inside a network is inherently trustworthy, zero trust enforces validation at every layer, using tools like access controls, multi-factor authentication, and network segmentation. This mindset shift was clearly reflected in the final module discussion, where we considered how to persuade developers who may be skeptical of zero trust. In my own development practices, I now consider zero trust not just a network concept, but a design principle for all user interactions and data flows.

Finally, implementing security policies—such as encryption requirements, coding standards, and access control policies—helps solidify a team’s security posture. In Project One, we created a secure development policy for Green Pace that included a blend of preventive and detective controls, such as unit testing, static analysis tools like cppcheck, and enforcement of SEI CERT standards. These policies were not just theoretical—they were mapped directly to real-world vulnerabilities, reinforcing their necessity. Additionally, the journal assignments helped reflect on these principles in practice, allowing me to see how authentication, authorization, and accounting (Triple A) fit into a layered security model.

CS-405 provided both practical tools and strategic perspectives that I will continue to apply. I now view security as an intrinsic part of the development process, not an afterthought. The combination of policy, mindset (zero-trust), and actionable coding standards gives me a solid foundation for contributing to secure software development in any professional setting.