Merrik Wright   
CS405

Adopting a secure coding standard is one of the most important practices in modern software development. Secure coding standards provide developers with a clear framework for writing safe, reliable, and maintainable code. Rather than waiting until the testing or deployment phase to address vulnerabilities, developers should build security into their code from the very beginning. This proactive approach not only prevents common issues like buffer overflows, input validation errors, and injection flaws, but also reduces the time and cost of fixing problems later. Throughout the course, we’ve seen how vulnerabilities often stem from poor coding decisions early in the lifecycle, reinforcing why security must be intrinsic, not reactive.

Evaluating and assessing risks also plays a critical role in secure development. Not all vulnerabilities present equal levels of threat, and it’s important to weigh the likelihood and impact of each issue against the cost and effort of fixing it. For example, mitigating a low-risk bug that would require a full system rewrite might not be practical, while addressing a high-risk SQL injection with a simple prepared statement would be an obvious priority. The ability to perform a cost-benefit analysis ensures that resources are focused where they matter most, which ultimately leads to more secure and efficient software delivery.

Zero trust has fundamentally changed how I view system security. The principle of "never trust, always verify" forces developers and system architects to assume that every user, device, or process could be compromised. This mindset encourages robust authentication, access controls, encryption, and logging at every layer of an application. Rather than relying on perimeter defenses alone, zero trust promotes internal compartmentalization and strict privilege management. As a result, even if one part of a system is breached, the attacker cannot easily move laterally. Incorporating zero trust into both coding and system design adds a vital layer of defense.

Implementing security policies based on best practices and tailored to the specific needs of an organization is another crucial element of a secure development environment. Policies should clearly define roles, responsibilities, and expectations for developers, testers, and system administrators. They should also include standards for code reviews, version control, and incident response procedures. When backed by training and reinforced through regular audits, security policies provide the structure needed to maintain consistency and accountability across teams. Recommendations for policy improvements should be based on real-world lessons, such as breaches or audit findings, to ensure policies stay relevant and effective.

In reflecting on these topics, I’ve realized that secure coding is not just about technical skill, it’s also about discipline, mindset, and culture. The course has emphasized that integrating security from the start, performing ongoing risk assessments, and applying models like zero trust all work together to create resilient systems. Moving forward, I will prioritize writing secure code, encouraging peer review, and aligning with organizational policies that support long-term security goals. Ultimately, secure development is a shared responsibility that must be embedded at every stage of the software lifecycle. This was an incredible course!