# CS 405 Secure Coding

***8-2 Journal: Portfolio Reflection.***

Software development often prioritizes and, in some cases, almost entirely focuses on functionality of an application. “Don’t leave security to the end” emphasizes the importance of security as a best practice in secure coding, by laying the case for the adoption of secure coding standards early on (requirements stage) of the SDLC to ensure that secure mechanisms that allows for well-established secure coding practices and decrease refactoring are in place from the start.

Ensuring Defense in Depth strategies are followed, and security policies are followed and reviewed promptly can mean the difference between wasting a lot of money and time dealing with vulnerabilities, patch management, postproduction bugs, experiencing serious software compromises, and having coding principles and standards that will help locate, classify, and ultimately address threats based on their respective degree of potential impact, (ASSESS AND PLAN. Defense-in-depth is one such strategy that utilizes multiple layers or redundancies of protection in the event that one or more of the layers of defense is breached. Attack mitigation and cost saving benefits associated with defense in depth and secure coding principles and guidelines implemented early on, far outweigh the cost of leaving it to the end.

As mentioned in my post on zero trust, the number of data compromises organizations big and small have suffered as attackers increasingly up their game, the need to never trust, always verify a.k.a. zero trust is becoming more and more indispensable. The implementation of security standards (policies and principles) from a conventional identity and access management standpoint where you authenticate once, and trust indefinitely is no longer sustainable in a modern multicloud and microservice environments. Zero trust is a philosophical approach to identity and access management, establishing that no user or software action is trusted by default. In other words, authenticate everything. Authentication with zero trust occurs in real time and to a much more granular level.

Integrating security throughout the software development lifecycle (SDLC) starting as early in the process as possible, and at all relevant points including during the inception process and every design decision helps establish strong coding practices. Security practices can/should be incorporated into Static, Unit, and system testing processes in order to minimize compromising components in the application/software. In production, authentication, authorization, and accounting practices can ensure that breaches or vulnerabilities are minimized and tracked so that enhancements can be made. Implementing a DevSecOps workflow incorporates security at every step of creation and deployment. What this means is that security is not just an additional layer to add to the top of a tech stack but should be baked into every facet of the application.

One aspect in coding or software development that I know can for a fact ensure that security is addressed inherently is remaining consistent with my coding standard. This should align with the programming language I am using. Exceptions and error handling should also be handled properly, especially if it relates to a run-time environment. Test as you go, validating input data, and heeding compiler warnings, would ensure issues are mitigated and not left until an issue emerges.

Kind regards,

Denis.