8-2 Journal: Portfolio Reflection

**Adoption of a secure coding standard, and not leaving security to the end**

The adoption of a secure coding standard integrates security as a primary component throughout the software development lifecycle. It enforces rules and principles that enhance code security during development and mitigate vulnerabilities effectively. Techniques such as input validation, dynamic and static code analysis, compile checks, and multi-factor authentication are essential to reducing risks (Top 10 Secure Coding Practices - CERT Secure Coding - Confluence, n.d.). Developers can also leverage the OWASP secure coding practices and the SEI Cert C++ Coding Standard to strengthen their approach.

These frameworks emphasize the importance of applying security at appropriate intervals. The idea of shifting-left security (DORA | DevOps Capabilities: Shifting Left on Security, n.d.) highlights the benefits of embedding security into the early stages of development. Incorporating security into each phase not only reduces risks but also builds a foundation for more resilient software. Delaying security to the end of development can lead to increased vulnerabilities and require more costs to remediate. Prioritizing security from the beginning is a core principle of secure coding and is vital for protecting software systems.

**Evaluation and assessment of risk and cost benefit of mitigation**

Ensuring code security requires robust risk assessment and evaluation processes. These enable developers and operation teams to conduct thorough code reviews prior to deployment. Techniques such as threat modeling, code reviews, cost-benefit analysis (GeeksforGeeks, 2024i) play a critical role in identifying implicit vulnerabilities and ensuring the integrity of the code.

Risk assessment allows businesses to prioritize security based on the sensitivity of the code. For example, code handling sensitive financial data requires heightened security measures, which may increase costs, whereas less critical code may not demand as much scrutiny. Applying thorough assessments enables companies to allocate resources efficiently, ensuring secure coding is both effective and sustainable.

**Zero trust**

The zero-trust model is a cornerstone of secure coding practices, built on the philosophy of “never trust/always verify” (Threatpost, 2020). This approach requires every user and device requesting access to a system to be authenticated and authorized individually. Referred to as the “castle and moat” approach, it strengthens an organization’s defenses by ensuring strict access throughout the control of each level. Incorporating zero-trust principles during development helps developers design systems that maintain no inherent trust among users, both internal and external, thereby reducing the risk of security breaches. This model is particularly effective in protecting sensitive data and critical infrastructure while ensuring customer trust.

**Implementation and recommendations of security policies**

A well-defined security policy is the backbone of secure coding and serves to safeguard sensitive data and systems. It provides strategies to strengthen protection against vulnerabilities including the integration of AAA framework. By incorporating Authentication, Authorization, and Accounting (Mylonas, 2018b), developers can enforce strong password management, multi-factor authentication, and access compartmentalization, ensuring a secure environment.

To enhance security policies, I recommend adopting a comprehensive approach that combines the following elements: the Top 10 Secure Coding Practices, OWASP secure coding practices and the SEI Cert C++ Coding Standard, a zero-trust policy while also integrating the and integration of automated security tools for consistent monitoring and threat detection. By holistically implementing these components, organizations can redefine secure coding practices and create a more secure software development environment. This multi-faceted approach ensures both the integrity of the software and the protection of customer data.

**References**

*DORA | DevOps Capabilities: Shifting left on Security*. (n.d.). <https://web.archive.org/web/20240524180526/https://dora.dev/devops-capabilities/process/shifting-left-on-security/>

GeeksforGeeks. (2024i, July 29). *Secure Code Review assessment*. GeeksforGeeks. <https://www.geeksforgeeks.org/secure-code-review-assessment/>

Mylonas, L. (2018b, November 27). What is AAA security? An introduction to authentication, authorisation and accounting. Codebots. https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting

*Top 10 Secure Coding Practices - CERT Secure Coding - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices?focusedCommentId=88044413