**Module 6: Journal**

Zane Milo Deso

Southern New Hampshire University

CS-405: Secure Coding

**Journal**

**Introduction**

Security in software development is most effective when embedded proactively within every stage of the process, rather than addressed only after vulnerabilities emerge. As software complexity and cyber threats evolve, adopting secure coding practices early on becomes ever important. This journal explores why security should not be put off for the final stages of software development. It discusses steps to mitigate threats and highlights specific examples of integrating security intrinsically into development workflows.  
**Reflection on Secure Coding Practices**

The statement "Don’t leave security to the end" stresses integrating security considerations throughout the entire software development lifecycle, rather than treating security as an afterthought. Usually, developers might build applications primarily for functionality and performance, addressing security issues only after vulnerabilities are discovered during testing or, worse, post-deployment. Such a reactive approach may lead to tech debt. Prescriptively, secure coding best practices encourage proactive measures, from initial requirements analysis to coding, testing, and deployment. By constantly monitoring and addressing security risks, developers can significantly reduce vulnerabilities, lower remediation costs, and maintain robust software integrity. Secure software development lifecycle is surely a best practice to adopt across the board.

### **Proactive Threat Prevention Measures**

To prevent threats, it is indispensable to follow structured secure coding practices throughout development. First, the team may adopt threat modeling early on to identify and prioritize potential vulnerabilities specific to the application's architecture and functionality. Carrying out secure coding standards, such as OWASP guidelines, establishes that common pitfalls, such as injection attacks, buffer overflows, and insecure authentication methods, are mitigated during development. Meticulous input validation, sapid error handling, and proper data sanitization further minimize attack surfaces. Ongoing integration tools merged with static analysis tools, such as Cppcheck, help detect reveal before code is published. Routine code reviews and security audits clear the way for collaboration, confirming that multiple perspectives scrutinize the codebase for security oversights. Finally, educating developers and stakeholders on relevant threats and security best practices fosters a proactive security culture across teams.

### **Integrating Security Throughout Development**

In my Project Two presentation, one example I will call attention to is the enmeshment of automated static code analysis tools during the coding phase itself. Tools such as Cppcheck will be used exhaustively during each phase of development. These tools can scan code in real-time, identifying issues like potential SQL injection vulnerabilities, improper memory management, or insecure handling of user input immediately. By including this self-driving detection within our continuous integration pipeline, potential threats will be identified and handled without delay. This proactive strategy exhibits how intrinsic security practices are inherent for the daily development workflow, supporting the principle of “not leaving security to the end.”

## **Conclusion**

In summary, prioritizing security from the get-go is a keystone of endurable software development. Embedding secure coding practices with each step of the development cycle, utilizing threat modeling, secure standards, automated analysis, and continuous security reviews, developers tremendously minimize risks and protect their applications. Encompassing this approach augments software quality and user trust while also pigeon-holing efficiency by mitigating expensive post-deployment vulnerabilities.