**8-2 Journal**

Mason Shaner

Southern New Hampshire University

CS-405 Secure Coding

Dr. Mimi Tam

December 22nd 2024

**8-2 Journal**

Security must be a top priority for any business engaged in the development and maintenance of custom software. To effectively mitigate vulnerabilities throughout all phases of the software development lifecycle (SDLC), organizations should implement secure coding standards and best practices, such as the zero trust model. While integrating these new techniques and practices may present significant challenges when modifying existing systems, the advantages they bring far surpass the costs and efforts required for their adoption.

Part of ensuring that security is prioritized is to not leave it as an afterthought. "Don't leave security to the end" refers to the idea that security and secure coding practices should be considered and implemented in all stages of the software development lifecycle (SDLC). Many development projects focus on creating the minimum viable product as quickly as possible, and often, this urgency to deliver working software comes at the cost of thorough testing and security measures. However, it is not always possible to retroactively integrate security measures into a project once the development phase is over without significant cost, which highlights the importance of practicing secure coding at all times. It is much easier for a developer to know they are writing potentially unsafe code, such as collecting user input, and to implement the security policy or mechanism required at that time than it is later. As such, developers must write secure code and make security-minded decisions at every phase of the SDLC to avoid missing potential vulnerabilities or having to rewrite large portions of code.

One way to ensure that security is not left to the end is to adopt a secure coding standard that can serve as a guide to developers throughout the SDLC. A secure coding standard establishes a set of guidelines for developers to adhere to when writing code. For instance, a secure coding standard may prohibit implicit conversions between data types, which is a process where a value is automatically converted from one data type to another. Another example of a secure coding standard may be to ensure that integer overflow or wrapping is prevented. By adopting a standard that enforces the use of best practices, developers will write code that is more secure and correct than if they were left on their own, where mistakes and oversights often result in vulnerabilities.

Another method for ensuring security at all stages of a system's development and operation is to adopt a zero-trust security policy. Zero trust is an approach where "Instead of assuming everything behind the corporate firewall is safe, the Zero Trust model assumes breach and verifies each request as though it originates from an open network. Every access request is fully authenticated, authorized, and encrypted before granting access" (*Zero Trust Model - Modern Security Architecture | Microsoft Security*, n.d.). This type of approach prevents the possibility of an attacker who breaches a system gaining trusted access. Instead, every user within the system is continuously verified, and their permissions are context-dependent on many different factors, including what device they are on, where they are located, their role in the organization, and the sensitivity of the data they are trying to access.

Any major changes to a system incur some risks in addition to benefits. Some of the risks of adopting a secure coding standard and a zero trust approach include refactoring, and thus potentially breaking, legacy code, increasing the complexity of the current system resulting in it taking more time to develop and deploy, and the greater level of skill and knowledge required from developers to successfully implement these policies and standards in practice. This includes understanding the principles of secure coding, being able to identify potential vulnerabilities, and having the expertise to configure and manage a zero-trust security model. While these are real risks and certainly will cause some additional difficulty and cost to address, they are far outweighed by the benefits of moving to a more secure approach.

The benefits of adopting a more proactive security approach include preventing vulnerabilities and, therefore, greatly reducing the likelihood of the system being attacked or exploited. This proactive approach to security not only saves the tremendous cost and time that can occur if a vulnerability is exploited but also provides a sense of reassurance. Many companies have suffered catastrophic setbacks as a result of their systems being compromised. For example, 7-Eleven suffered a SQL injection attack that exposed 160 million credit and debit card numbers, which caused thousands of individuals to lose their life savings (*SQL Injection Used in Heartland, 7-Eleven and Hannaford Breaches | AON*, 2023). The monetary and reputational damage of this type of attack can be unsalvageable. Thus, the benefits of adopting a more robust approach to security that utilizes techniques like zero trust and secure coding standards outweigh the risks.

**References**

*SQL injection used in Heartland, 7-Eleven and Hannaford breaches | AON*. (2023, March 1). Aon. <https://www.aon.com/cyber-solutions/aon_cyber_labs/sql-injection-used-in-heartland-7-eleven-and-hannaford-breaches/>

*Zero Trust Model - Modern Security Architecture | Microsoft Security*. (n.d.). <https://www.microsoft.com/en-us/security/business/zero-trust#:~:text=A%20Zero%20Trust%20network%20fully,to%20anomalies%20in%20real%20time>.