Journal: Portfolio Reflection

Security is not a final checkbox that is added at the end of a project; instead, security must be front-loaded with secure coding standards incorporated early in the software development lifecycle to identify vulnerabilities sooner and fix them. These formal standards provide guidelines to avoid common programming errors and integrate security into the software's design (*What Are Secure Coding Standards? CERT, OWASP, and Compliance Explained | Kiuwan*, 2025). Secure coding standards are invaluable when memory-unsafe languages like C++ and C still dominate the market (Gaynor, 2018).

Mitigating vulnerabilities early in the software development lifecycle is cost-effective, as the price and complexity compound as the development cycle continues. There are frameworks like NIST SSDF (SP800-218) that focus on risk management in each stage of the SDLC lifecycle, defining security requirements and CI practices (Souppaya et al., 2022).

One principle of security is the zero-trust mindset, which assumes no device or user is trusted by default. Trust is perpetually verified with the least privilege granted at each step (*What Is Zero Trust?*, 2019). The creation of zero-trust security was in response to the growing prevalence of cloud-based environments, remote work, and mobile devices, which have rendered traditional security models obsolete (*Zero Trust Maturity Model*, 2025).

The implementation of security policies determines the success or failure of the project. Using comprehensive security-by-design principles like defense-in-depth, threat modeling is a first step. Strict adherence to these principles is important, but not comprehensive enough. NIST hosts a list of tools that analyze code for security vulnerabilities. The list is comprehensive, including Java, JavaScript, C, C++, Bash, Ruby, C#, and far more. C and C++ analyzers include CodeCenter, Astrée, C/C++test, CodeValor, cppCheck, FlawFinder, and more (*Source Code Security Analyzers*, 2021).

Do these "by-the-book" guidelines still have any real benefits, or are they just a checkbox to affirm a job for a security developer? Several case examples exist of catastrophes from a lack of security-focused design. In 2018, British Airways failed to keep a JavaScript library updated, exposing them to a severe vulnerability. When exploited, there payment page was redirected to a malicious domain that logged credit details. The result was over 400,000 customer’s cards were compromised, leading to a £20 million fine (*Breach of British Airways’s Customer Data*, 2020). A full continuous improvement cycle should have caught this, discovered the CVE, and urged the developers to fix it. An example within the United States was the 2019 Captial One Breach which was caused by poor cloud infrastructure, monitoring, and internal controls, which allowed a disgruntled AWS employee to access over 100 million individual data. The result was the bank was fined $ 80 million dollars, with an additional $ 70 million from legal fees (Berman, 2020). Zero-trust along access should have prevented this user from accessing the companies database after release but Captial One had not instituted anything of the sort. There are numerous more examples like Interserve’s 2021 ransomware attack that led to a £4.4 million fine, Equifax’s 2017 data breach leading to legal fines and compensation of $700 million, and many more. Security by design is not just a box to check nor is it a choice, but a necessity to keep your customers and user’s information safe and your company afloat.

References

Berman, M. (2020, August 11). *Board & Management Fail: Operational Risk Management Weaknesses Leads to $80 Million Fine*. Ncontracts. https://www.ncontracts.com/nsight-blog/operational-risk-management-weaknesses-80-milliion-fine

*Breach of British Airways’s customer data*. (2020, May 20). Wikipedia.org; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/British\_Airways\_data\_breach

Gaynor, A. (2018, November 15). *The Internet Has a Huge C/C++ Problem and Developers Don’t Want to Deal With It*. VICE. https://www.vice.com/en/article/the-internet-has-a-huge-cc-problem-and-developers-dont-want-to-deal-with-it/

*Shifting left on security*. (2024). Archive.org. https://web.archive.org/web/20240524180526/https://dora.dev/devops-capabilities/process/shifting-left-on-security/

Souppaya, M., Scarfone, K., & Dodson, D. (2022). Secure software development framework (SSDF) version 1.1. *NIST Special Publication 800-218*, *1*(1). https://doi.org/10.6028/nist.sp.800-218

*Source Code Security Analyzers*. (2021, March 23). NIST. https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers

*What Are Secure Coding Standards? CERT, OWASP, and Compliance Explained*. (2025, August 21). Kiuwan. https://www.kiuwan.com/blog/secure-coding-guidelines-2/

*What is Zero Trust?* (2019). Crowdstrike.com. https://www.crowdstrike.com/en-us/cybersecurity-101/zero-trust-security/

*Zero Trust Maturity Model*. (2025). Https://Www.cisa.gov/Zero-Trust-Maturity-Model; U.S. Department of Homeland Security. https://www.cisa.gov/zero-trust-maturity-model