This class has taught me a lot, but one of lessons that will stick with me is “don’t leave security until the end”. Security should be considered at every stage of development from start to finish because it’s much harder to implement after the fact. It will cost significant time and money to rework the code with secure principles, not to mention the delays that could happen and the vulnerabilities that will exist until it can be fixed.

In Project Two, we really focused on early security development using the SEI CERT coding standards and tests to catch issues at the start. This will catch errors with intentionally written tests, which I was able to use to test memory and critical methods. Tools like Cppcheck and Clang++ are very helpful in finding vulnerabilities that wouldn’t be obvious in a code review. The automation tools are important in software development today and enable security without a cost to time.

There is also a balance between risk and cost, which was made very clear when we developed our threat matrix where we had to determine the highest risks that should be handled first. In the evaluation of the likelihood, severity, and ease of remediation to prioritize the most likely to leave the code open to threats. I found that STD-008-CPP Never Hard Code Secrets was the highest priority for my security police as it could expose secure credentials and sensitive data. If that vulnerability is fixed early on in the process, the cost will be lower to development which increases that amount of security. The threat matrix is a good representation of what a security expert in real-life would have to consider and determine which vulnerability presents the highest risk and should be dealt with first.

Zero trust architecture assumes that everything inside a network is a risk and nothing is safe. While a negative way to view security, it does emphasize identity validation and the principle of least privilege which provinces better security. In this project, this principle was applied in my recommendation for multi-factor authentication, role-based access control, and tracking all user activities. These suggestions build on the NIST guidelines that have been laid out, and supported by our readings this term which emphasizes protecting boundaries.

Developing and implementing security policies was a huge focus in my final project particularly in encryption, AAA policies, and secure logging. My recommendations tried to look ahead of the current policy and focus on the growth and scalability of the project. I suggested using AddressSanitizer to improve memory, and OWASP dependency-check to scan third-party libraries since they are prone to attacks. This will ensure that security can continue to grow and is proactive rather than reactive. Looking ahead, I will incorporate these principles into all my future projects. Secure coding is important regardless of the goal of the application and should never be ignored.

References

OWASP Foundation. (n.d.). Secure coding practices quick reference guide: Checklist. OWASP. https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/02-checklist/05-checklist

Terry, R. (2025, March 13). What is Zero Trust? – Guide to Zero Trust Security. CrowdStrike. Retrieved August 12, 2025, from https://www.crowdstrike.com/en-us/cybersecurity-101/zero-trust-security/

Thurmond, T. (2023, December 27). 8 best secure coding practices learned from OWASP. KirkpatrickPrice. https://kirkpatrickprice.com/blog/secure-coding-best-practices