Amanda Stone

The textbook showed that in C and C++ simple string handling errors can lead to buffer overflows that attackers exploit (Seacord, 2013). If developers wait until the end to deal with security, then those errors are already built deep into the code base. The same goes for formatted output, where format string vulnerabilities can expose memory. When secure standards are used from the start, like safer string functions and runtime protections, the system becomes more stable and cheaper to maintain. It is easier to build secure functions than to fix unsafe ones later. This connects with the idea of shifting security left, making it part of development instead of just a late test (DORA, 2024).

Evaluation of risk and cost benefit is another key theme. Developers and managers need to weigh how much a vulnerability could cost if exploited versus the cost to fix it. The AES defense-in-depth article made this point by showing how layers of encryption improve reliability against attacks even though they add complexity (Mahmud et al., 2021). The OWASP risk frameworks explain that ignoring a vulnerability may seem cheap now, but it usually becomes more expensive after a breach. Even exception handling in C++ shows this tradeoff. Using exceptions correctly prevents unstable states, but mishandling them can introduce more risk (ACoder’sJourney, 2018). The cost of proper handling is so small compared to the possible impact of a whole crash or data leak.

Zero trust was highlighted a lot as well. Traditional perimeter models assume trust once inside, but the articles and videos showed why this is outdated. Zero trust requires constant identity checks, access based on least privilege, and validation at every step (Digital Guardian, n.d.; VMware Carbon Black, 2020). I liked how the Threatpost guide broke it into pillars: identity, devices, applications, networks, and data (Cimpanu, 2020). The videos reinforced that zero trust is not just for big companies, but essential in a world where data moves across devices and cloud systems. This model means developers and admins cannot assume safety, they must design code and systems that verify every request.

Finally, the role of policies ties all these ideas together. Secure coding policies enforce practices like memory management rules, safe exception handling, authentication requirements, and static analysis audits. Chapter 9 of the textbook made clear that compiler security features and defense in depth are only effective if they are applied consistently (Seacord, 2013). Readings on AAA security (authentication, authorization, and accounting) showed how policies make sure access is enforced in practice, not just in theory (Codebots, 2023). Policies also must adapt. Best practices like DevSecOps emphasize automating security checks early and often so that policies are not ignored (TechBeacon, 2024). If an organization builds policies around secure coding standards and zero trust, it creates a culture where developers naturally think of security as part of the job.

I think the main lesson is that secure coding is both technical and organizational; it is about using safe coding practices in C and C++, evaluating risks in a realistic way, adopting zero trust principles, and enforcing policies that support all of these. When security is integrated from the beginning and reinforced through culture and policies, the code and the systems it runs on become much more reliable.

**References**

ACoder’sJourney. (2018). *Top 15 C++ exception handling mistakes and how to avoid them*. <https://www.acodersjourney.com/top-15-c-exception-handling-mistakes-avoid>

Cimpanu, C. (2020, February 18). *A practical guide to zero trust security*. Threatpost. <https://threatpost.com/practical-guide-zero-trust-security/151912/>

Codebots. (2023). *AAA security: An introduction to authentication, authorisation, accounting*. <https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting>

Digital Guardian. (n.d.). *Understanding the zero trust security model to safeguard digital infrastructure*. <https://www.digitalguardian.com/blog/understanding-zero-trust-security-model-safeguard-digital-infrastructure>

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

Mahmud, S., Khan, M. I., Al-Asaduzzaman, M., & Al-Amin, M. (2021). Defense in depth approach on AES cryptographic decryption core to enhance reliability. *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*. IEEE.

Seacord, R. C. (2013). *Secure coding in C and C++* (2nd ed.). Addison-Wesley.

TechBeacon. (2024). *6 DevSecOps best practices: Automate early and often*. <https://web.archive.org/web/20240117143629/https://techbeacon.com/security/6-devsecops-best-practices-automate-early-often>

VMware Carbon Black. (2020, April 28). *Why Zero Trust is essential for organizations today* [Video]. YouTube. <https://www.youtube.com/watch?v=1D5mg9an19o>

Veritas Technologies. (2019, July 25). *What is Zero Trust?* [Video]. YouTube. <https://www.youtube.com/watch?v=d-Jk3GzjwlM>