Reflect on and include a discussion of the following topics, using readings from throughout the course to support your views.

* Adoption of a secure coding standard, and not leaving security to the end

Throughout this course, one of the most valuable takeaways has been the importance of adopting a secure coding standard early in the development lifecycle. As emphasized in *Secure Coding in C and C++* by Robert Seacord, leaving security considerations until the end of a project creates unnecessary technical debt and opens the door to exploitable vulnerabilities. Every choice made when architecting a system is a tradeoff. Secure coding should be foundational, not reactive, which is the tradeoff.

* Evaluation and assessment of risk and cost benefit of mitigation

Another key area has been the evaluation and assessment of risk. Effective risk management isn’t about eliminating all threats, it’s about identifying which risks matter most, and which mitigations are cost-effective for the situation you are dealing with. For example, applying the NIST Cybersecurity Framework encourages teams to weigh the potential impact of a vulnerability against the cost and feasibility of mitigation. This cost-benefit lens is critical when working with limited budgets or complex systems where trade-offs are necessary.

* Zero trust

Learning of the concept of *Zero Trust* has also played a major role in shaping my perspective on modern security practices in enterprise environments. Zero Trust, a security framework outlined in the Google BeyondCorp whitepaper, is all about constant verification, making sure the user, device, and request are all trustworthy no matter where they’re coming from. This approach makes a lot of sense, especially with how common insider threats, compromised credentials, and lateral movement have become potential routes of attack. Adopting this mindset helps build systems that are more resilient from the ground up, not just protected at the edges.

* Implementation and recommendations of security policies

Finally, putting strong security policies into place and keeping them up to date are what turn high-level strategies into consistent, actionable practices across the organization. These policies establish clear expectations and provide a foundation for accountability. As highlighted in the course readings, good policies should be easy to understand, reviewed regularly, and supported by both training and automation. A strong example of implementing effective security policies can be seen in organizations adopting DevSecOps practices. Rather than treating security as a final step in development, DevSecOps brings security into every stage of the software development lifecycle. This approach supports the idea that policies should not only set expectations but also influence day-to-day behaviors within fast-paced workflows like CI/CD. For example, a well-crafted secure coding policy in a DevSecOps environment would require developers to use automated static and dynamic application security testing during development instead of waiting until just before a release. These tools, when defined and enforced through policy, reduce the risk of vulnerabilities making it into production.

To support this kind of policy, automation has become essential. Manually enforcing security in environments that deploy code frequently is not practical. That is why policies should require automated security scanning and dependency checks, which ensure that known vulnerabilities in code or libraries are caught early. Security policies should also include training requirements so that developers understand how to respond to findings and use security tools effectively. In this way, security policies serve not just as guidelines but as working systems that align development speed with security outcomes.