# 8-2 Journal: Portfolio Reflection

Throughout this course, the importance of adopting a secure coding standard from the outset has become clear. Leaving security to the end of a project increases the likelihood of costly vulnerabilities and rework. Secure coding standards—like those recommended by CERT or OWASP—provide developers with clear guidelines to prevent common coding flaws such as buffer overflows, injection attacks, and insecure memory use. By integrating these standards early, teams can embed security into the development lifecycle, aligning with DevSecOps principles that emphasize “shift left” strategies. As noted in course readings, tools like static code analyzers (e.g., Cppcheck or Fortify SCA) help enforce these standards and detect violations before code reaches production.

Evaluating risk and the cost-benefit of mitigation is another key takeaway. Not all vulnerabilities carry the same weight—understanding the likelihood and impact of a threat helps prioritize limited resources. For example, a high-severity issue in a public-facing login service should be addressed before a low-risk flaw in an internal admin panel. Using a risk matrix enables a structured approach to decision-making. In our course activities, threat modeling and risk ratings helped simulate real-world tradeoffs between cost, effort, and potential damage.

Zero trust, summarized by the mantra “never trust, always verify,” challenges traditional perimeter-based thinking. In a zero trust architecture, access is granted based on continuous authentication, least privilege, and contextual data. This principle was reinforced in our modules discussing defense-in-depth and identity management. It means developers must now design systems assuming breach, validating every access attempt. As a result, security becomes more proactive and less reactive—limiting lateral movement and insider threats.

Implementing and maintaining strong security policies is the foundation for consistent practices across a team or organization. Our work on the Green Pace secure coding policy demonstrated how aligning technical controls, employee training, and threat awareness creates a culture of security. Effective policies must be adaptable, enforced through automation when possible, and regularly updated to reflect new threats and compliance requirements. Hosting regular policy reviews, as recommended in the course, ensures continued relevance and awareness among staff.

In summary, this course highlighted that secure software development is not a one-time task, but a continuous process that must be built into every phase of the development lifecycle. Through secure coding standards, risk-based prioritization, zero trust implementation, and clear policies, developers can produce software that is not only functional—but resilient.