Courtney Warner

CS 405

Module 6 Journal

**“Don’t Leave Security to the End” – What This Means**

In secure coding, the phrase “Don’t leave security to the end” is a critical best practice that emphasizes the importance of incorporating security at every stage of the software development lifecycle, not just during testing or deployment. Waiting until the final phases of a project to address security often leads to vulnerabilities being missed or hastily patched. Instead, security must be designed into the system from the beginning, just like functionality or performance.

This mindset ensures that secure architecture decisions are made upfront, that developers consistently write secure code, and that the application is resilient to threats before it goes live. By taking a proactive rather than reactive approach, development teams reduce technical debt, avoid costly late-stage rewrites, and improve long-term maintainability and compliance.

**Steps to Prevent Threats**

To prevent threats effectively during development, I follow several best practices:

**1.** Validate all input data - Input validation prevents common attacks such as SQL injection, buffer overflows, and XSS.

**2.** Follow secure coding standards - I refer to SEI CERT C/C++ rules to ensure consistency and prevent known vulnerabilities.

**3**. Use the principle of least privilege - Limit access rights to users, systems, and code segments only to what is necessary.

**4.** Apply defense in depth - Implement multiple layers of security so that if one fails, others still protect the system.

**5.** Use static code analysis tools - Tools like SonarQube or Cppcheck can detect vulnerabilities during coding.

**6.** Perform frequent code reviews - Peer reviews often uncover logic errors or security risks missed by the author.

**7.** Conduct threat modeling during design - Identify potential attack vectors before any code is written.

**Project Two: Example of Proactive Security**

In Project Two, I am including unit tests for vulnerable code paths to ensure early detection of issues and to prevent regressions. For example, I wrote custom unit tests to check input validation for dangerous function parameters. This ensures that if someone modifies the function later, the test will fail if the validation is bypassed.

Additionally, I integrated static analysis into the DevSecOps pipeline using SonarQube. This allows vulnerabilities to be flagged as soon as the code is committed, well before the final testing phase. These measures demonstrate that I am building security into the workflow, rather than treating it as a last-minute fix.