Andres Trujillo

02-25-2025

CS 405 Prof Olga Mills

There are multiple implementation details and recommendations for secure coding and we must always be aware of the tools and resources. The SEI CERT coding standard is invaluable for examples. We can use static analysis tools such as an IDE with verified plugins such as CPPCheck and clang-tidy for CLI tools. Configuring projects to use such tools is highly important to implement coding standards.

These tools provide code quality and compliance checks. Not recommending them or not using them is an unsafe practice towards detecting potential defects. Sometimes when programs aren't properly secure the user can hurt the computing resources with denial of service attacks, via memor finding y leaks and other denial of service attacks where a server can become unavailable. Not leaving this detail until the end of the development lifecycle makes us aware of the security implementations and architecture of our code to exhibit zero trust computing.

Adopting these coding standards is where we can apply mitigations across various software roles, such as project managers, testers, and programmers. There is a need for security requirements, that assess risks. We move towards design where we analyze the attack surface by establishing design requirements. Implementations must use approved tools, and deprecate unsafe functions. Aftwards, verifying through dynamic analysis and fuzz testing where we purposely try erroneous input while the program is running to discover bugs or unspecified behavior. Then we can release the program by crafting the incident response plan and creating a final security review and then archiving it in order to pinpoint where problems start. This

security development lifecycle ends with a response where we execute the incident response plan of unsafe functions, exceptions not being caught or an abruptly terminating program.