CERT'S PODCASTS: SECURITY FOR BUSINESS LEADERS: SHOW NOTES

Cisco's Adoption of CERT Secure Coding Standards

Key Message: Implementing secure coding standards to reduce the number of vulnerabilities that can escape into operational systems is a sound business decision.

Executive Summary

Any software defect, whether security related or not, that is allowed to escape into a production system can cause a disruption of service affecting thousands to millions of users. Costs to both providers and users can be staggering – up to millions of dollars. About 3 years ago, Cisco decided to develop and implement a Secure Development Lifecycle Methodology (CSDL). The CSDL includes secure coding guidelines based on the CERT Secure Coding Standards. While local solutions had been in use for some time, a key goal of this effort has been to ensure a coding standard that applies uniformly across the entire organization to achieve consistency in Cisco's software quality and security. The standard is enforced and has automated support through the use of, for example, compilers and static analysis tools.

In this podcast, Martin Sebor, a technical leader at Cisco Systems, discusses Cisco's adoption of the CERT Secure Coding Standards and why the use of such standards is an essential part of Cisco's software development process.

PART 1: ENTERPRISE-WIDE STANDARDS; COST VS. BENEFIT

Cisco's Compiler Tool Chain Team

Martin is a member of Cisco's Compiler Tool Chain team, responsible for developing:

- compilers, linkers, and related tools for Cisco's operating system, IOS
- mechanisms for detecting and preventing defects in the IOS code base, such as buffer overflows.
- mitigation technologies such as address space layout randomization

Cisco's Secure Development Lifecycle Methodology

Martin interfaces with the Advanced Security Initiatives Group (ASIG) on security-related projects. The ASIG supports Cisco's Secure Development Lifecycle Methodology (CSDL), which was put in place about 3 years ago.

Until recently, the CSDL was missing a set of consistent security coding practices across the entire enterprise. Up to this point, there were localized solutions, the majority of which relied on informal enforcement via the code review process, with little automated support.

Cisco decided on a phased approach to integrate security into their development process. Martin became involved when the CSDL methodology and development tools – compilers and static analyzers – were ready for the formal introduction of a secure coding standard.

Martin had been involved with the CERT Secure Coding Standards project and participated in the C and C++ international standardization committees, so was appointed to lead Cisco's efforts in this area.

A key goal was to ensure that the Cisco standard would apply uniformly across the entire organization to achieve consistency in their software quality and security.

Cost vs. Benefit

Any defect, whether security related or not, that is allowed to escape into a production system can cause a disruption of service affecting thousands to millions of users. Such a defect may also violate service level agreements. So costs to both providers and users can be staggering – up to millions of dollars.

Finding Defects During System Testing

Finding a defect during system testing certainly helps. However, vigorous testing is an involved and time-consuming process, taking weeks and possibly months for a complex system.

When a vulnerability-inducing defect is discovered during system testing, the software development team has to fix the defect. And then the software must be re-tested. This is costly to both the development and test teams.

It is not uncommon to incur costs of hundreds of thousands of dollars as well as incur costs resulting from schedule delays.

Finding Defects During Unit Testing

Finding a defect during unit testing is somewhat better. But such defects may cause cascading failures across interfacing components. Such failures can render component test results useless. It is often challenging to find the root cause of the underlying defect.

Finding Defects During Coding: Automation Is Essential

The best time to uncover a coding defect is when it is injected into the code during development. Visual inspections such as code reviews is one effective mechanism. However, a more reliable approach is the use of automated tools such as compilers and static analyzers.

Ideally the defect is discovered before the code changes in the shared code base so it doesn't affect other developers.

When this happens, the code author simply corrects the code and reruns the tool that revealed the defect to confirm it has been corrected. This is much more cost-effective.

PART 2:DECISION TO USE CERT STANDARDS; STEPS TO DEPLOY

Why Use CERT Standards?

Cisco considered a number of internal coding standards as well as external standards developed by other organizations.

They chose the CERT Secure Coding Standards for the following reasons:

- They are general in nature, independent of any particular domain, industry, or operating environment such as Windows, Macintosh, or Linux.
- They are based on the guarantees that are provided by the programming languages they support: C99 and C++2003.
- They rely on the guarantees provided by the runtime environment specified in the language policy specification.
- They closely track the development of new versions of the relevant language standards, such as C11 and C++11. This is a fairly unique feature in comparison to other coding standards.
- They draw from a number of other coding standards such as MISRA, Lockheed Martin's, and CWE.

  [Note: The CERT Secure Coding Standards Bibliography provides many additional references.]

- They are reviewed by over 300 security and industry experts.
- They are freely accessible on the CERT wiki and contributions are encouraged.
- They include examples of common coding errors as well as fixes for those errors, which serve as excellent

teaching material.

Deploying the CERT Standard at Cisco

This is the approach that was used at Cisco. It may vary slightly, or substantially, depending on the organization, its size, and its culture:

1. Decide on the primary goals of adopting a secure coding standard.
    a. Will it apply to all code, some code, newly developed code, etc.?
    b. Will it be enforced and how? Is it going to be a set of recommended guidelines or a mandate that will be strictly enforced? By automated tools?
2. Are we going to develop a new standard from scratch (not recommended based on the effort involved) or use an existing one?
3. Chose from the set of available standards. Involve representatives from the engineering community. Their participation is critical to success.
4. Formulate a process for socializing the standards in the engineering community. Email is not sufficient. Training is essential so develop a training plan.
5. Introduce the standard informally, through tech talks and brown bag lunch discussions.
6. Deploy the standard in phases:
    a. Phase 1: as an informational set of guidelines, without enforcement and with voluntary compliance
    b. Phase 2: develop and deploy automated checkers (compilers, static analyzers). Alert responsible managers about non-compliance. Create incentives and penalties to help drive up compliance.
    c. Later phases: require software programs to commit to a specific degree or level of compliance based on a maximum number of violations of a given severity. Prevent non-compliant code from being added to the code base. Perform random audits.

PART 3: THINK LIKE AN ATTACKER; CONNECT VULNERABILITIES WITH SOFTWARE DEFECTS

Software Engineers Typically Don’t Think Like Attackers

Software engineers tend to think first about functionality and performance. They introduce assumptions into the code base based on the "happy path” – the expected path through their programs.

It takes a change in mindset to appreciate that the happy path is likely not the path that will be taken by an attacker. This change can be accomplished by demonstrating examples of vulnerabilities and the attacks that exploit them, during tech talks or brown bag lunch meetings.

Link Exploited Vulnerabilities to Ineffective Secure Development Practices

A promising direction for future work is to examine the extent to which exploited vulnerabilities result from absent or ineffective secure coding practices, as well as violations of in-place coding standards as a root cause contributor.

Resources

CERT’s Secure Coding Standards website and wikis

Cisco’s Secure Development Lifecycle Methodology (CSDL)

CERT podcast: Mainstreaming Secure Coding Practices, March 2009.