Cisco's Adoption of CERT Secure Coding Standards
Transcript

Part 1: Enterprise-Wide Standards; Cost vs. Benefit

Julia Allen: Welcome to CERT's Podcast Series: Security for Business Leaders. The CERT program is part of the Software Engineering Institute, a federally-funded research and development center at Carnegie Mellon University in Pittsburgh, Pennsylvania. You can find out more about us at cert.org. Show notes for today's conversation are available at the podcast website.

My name is Julia Allen. I'm a principal researcher at CERT, working on operational resilience and software assurance.

Today I'm very pleased to welcome Martin Sebor, who is a technical leader at Cisco Systems. And today, Martin and I will be discussing Cisco's adoption of CERT's Secure Coding Standard.

For a little bit of background, listeners may also want to listen into a related podcast that Robert Seacord and I posted in March of 2009. Robert is the leader of CERT's Secure Coding Initiative, with whom Martin has been working.

So welcome, Martin, really glad to have you with us today.

Martin Sebor: Thank you Julia. I'm glad to be here.

Julia Allen: So what has been your role in the adoption and use of secure coding practices in general at Cisco? What have you been involved in doing with them?

Martin Sebor: At Cisco I'm a member of the Compiler Tool Chain Team. And the Tool Chain Team develops primarily features for compilers, linkers, and related tools specifically for the Cisco operating system, IOS.

And among the features that my team develops are detection and prevention mechanisms that allow us to detect defects that we tend to repeatedly find in the IOS code base. An example of some of these defects is the common buffer overflow.

My team also contributes to the development of mitigation technologies, such as the address space layout randomization. And one of the responsibilities of my team -- one of my personal responsibilities on my team -- is to interface with another team within Cisco called the Advanced Security Initiatives Group, or ASIG, on projects that are in this area, security-related projects.

ASIG is chartered with the development and deployment of various security programs within Cisco. Most of these programs are under the umbrella of the Cisco Secure Development Lifecycle Methodology, or CSDL.

The Cisco Secure Development Lifecycle was put in place about three years ago and we're still working on improving it today. One of the essential elements of CSDL that we were missing from our process, up until recently was robust coding rules that would allow us to establish a set of consistent secure coding practices across the entire organization.

Copyright 2012 by Carnegie Mellon University

And the fact that it was missing was not an oversight. We had a number of coding standards within the organization. Some were better than others. They all developed kind of organically, and the majority of them relied solely on informal enforcement via the code review process and had virtually no support for automation -- automatic detection by, say, static analysis tools.

The individual coding standards that we had in place also had fairly little visibility across business units and they lacked consistency, for obvious reasons. So in the early stages of the development of the Cisco Secure Development Lifecycle, Cisco realized that creating a successful development process was a long-term effort and decided to deploy it in phases.

And I personally became involved with the Cisco Secure Development Lifecycle at the time when both the methodology and our development tools-- the compilers and static analyzers in use at Cisco-- were ready for the formal introduction of a secure coding standard.

Thanks to my involvement with, or prior involvement, with the CERT Secure Coding Standards, and my work with Robert, and my participation in the C and C++ standardization committees, I was appointed to lead the effort at Cisco to establish the formal secure coding standards.

The new coding standards, unlike the previous informal coding standards at Cisco, are designed to apply uniformly across the whole organization. And this was an important goal for us to achieve consistency in the quality and in the security of all our software throughout the organization.

The enforcement of the new coding standard was also formalized, and it relies on a combination of peer review process, just like the previous coding standards. The peer review process was an early element of the Cisco Secure Development Lifecycle. The enforcement also relies, increasingly relies on detection by automatic tools, including our compilers and static analysis tools that Cisco uses.

Julia Allen: Okay. So more along the stage-setting side of things, before we get into your actual adoption of the standards, why do you think in general-- because this is, the podcast is sometimes for a listening audience that's just getting introduced to these ideas -- so what have you found, or why does Cisco believe that it's more cost effective to find a software security defect earlier in the lifecycle as opposed to in operations?

Obviously when you put a program in place like the one you're starting to describe, that's a big commitment and a big investment. And so why was Cisco willing to go down that path?

Martin Sebor: Well you're absolutely right. It is a big commitment and it is a big investment. In my experience, any defect, whether it's security related or not, that is allowed to make it into a production system might cause a disruption of service of the system -- potentially affects many, say, thousands or even millions of users. And for customers of service providers, it may violate their service-level agreements.

So the costs of such a defect with the users of the system and the provider of the service can be staggering. We're talking potentially millions of dollars. So finding the defect before deployment, before the software is shipped to the users, such as during system testing, avoids most of these costs. But at the same time, vigorous testing is an involved and time-consuming process. It can take weeks and maybe even months to schedule and to execute for a complex software system in a large organization like Cisco.

Copyright 2012 by Carnegie Mellon University

And when this cycle reveals a defect or a severe defect, like one that causes or exposes a security vulnerability, it means that the software has to go back to the development organization that can then fix the defect. And the software then needs to be retested. So this too can be very costly to the development and to the test organization. And it's not uncommon to see costs of hundreds of thousands of dollars; not to mention the indirect costs caused by delays in schedules.

So if we uncover the defect even earlier, during unit testing for example, we can reduce the cost even further. But defects that are uncovered during unit testing may cause cascading failures across various components, not just the component that the defect is in. And these failures can render the test results of the other components useless, and it can then take a number of engineers and a considerable amount of time, maybe days, to discover the failures across the multiple tests to decide or determine that the failures are due to the same underlying root cause.

So to reduce these costs even further, the best time to uncover a coding defect is at the time the defect is injected into the code during development. And the best or one of the mechanisms, one of the approaches to determining or detecting this defect can be a visual inspection, such as code review. But even better, it can be done by a more reliable mechanism such as an automated tool, a compiler or a static analyzer.

And ideally the defect is discovered even before the code changes in the shared coded base so that it doesn't affect other developers working on the same code base. So in such a case resolving such a defect is a simple matter of the author of the code change correcting the code, re-running the tool, the static analyzer or the compiler, to confirm its correctness and the absence of any diagnostic errors or warnings. So that's obviously much more cost-effective than any of the other approaches.

Julia Allen: Yes, that really makes a lot of sense. And going back to your earlier remarks, the whole notion of automation, and like you said static analysis and other types of automated support I suspect, really helps to both improve efficiency and effectiveness, correct?

Martin Sebor: Absolutely -- automation is essential. Not every defect can be detected automatically. Certainly there are classes of defects that don't lend themselves to diagnostic by automatic tools. But certainly those that are detectable, detecting them early by a reliable automatic tool is the way to go.

Part 2: Decision to Use CERT Standards; Steps to Deploy

Julia Allen: As you said, in the early years there were several standards that were either in full or partial use; perhaps not broadly deployed.

But as you went through your decision process, why did Cisco decide to use CERT's Secure Coding Standards in your software security development lifecycle work?

Martin Sebor: It's a good question, and I like the question. We did have a number of good, fairly good quality coding standards to choose from. The organization had a number of internal coding standards that I mentioned, that were developed largely independently by various Cisco engineering teams. And we also considered a number of external secure coding standards, or quality coding standards, developed by other organizations.

Copyright 2012 by Carnegie Mellon University

We didn't necessarily review all these coding standards but we did review and study several of them. And the reasons why we liked the CERT Secure Coding Standards were several. We liked the fact that the CERT Secure Coding Standards are general in nature; that they are independent of any particular domain or industry or any operating environment, such as Windows, Macintosh, Linux, or any other.

The CERT Secure Coding Standards are also based on the guarantees that are provided by the programming languages; either C99 or C++2003. And they rely on the guarantees provided by the runtime environment that's specified by the policy specification.

The CERT Standards also closely track the development of new versions of the relevant language standards, such as the newly ratified C11 or C++11. It's been our experience it's fairly unique among coding standards that are out there. Now at the same time, the CERT Secure Coding Standards draw on material from a number of other coding standards, such as the MISRA coding standard or the Lockheed Martin C and C++ coding standards, and other references on secure coding standards such as the CWE.

So they are generally applicable and subsume-- in our experience and in our view-- subsume most of the other coding standards that are out there, that are domain and industry independent.

Another positive about the CERT Secure Coding Standards is that they are reviewed by over 300 security and industry experts, including representatives from the static analysis community. They're also freely accessible on the wiki and make it possible for us and any other organization or individual to contribute material to them. And the wonderful thing about the CERT Secure Coding Standards is that they include examples -- examples of common coding errors as well as examples of how to fix those errors. So they provide excellent teaching material as well.

Julia Allen: Oh that's fantastic. Well thank you for citing those benefits. I really greatly appreciate it, as I know that CERT has benefited greatly from their collaboration with Cisco in moving that body of work forward.

So let's say that I'm in the position that Cisco was in three years ago or whenever this evaluation first started and I decided to follow Cisco's lead and adopt a standard like CERT's. So could you walk me through the steps involved in actually trying to deploy such a standard -- to actually put it into your defined process for developing software?

Martin Sebor: Sure. I can list the steps that we went through at Cisco. I think the steps are going to be different, maybe slightly or substantially, depending on the organization, the size of the organization that's adopting the standard, depending on the culture of the organization.

But in our view the first prerequisite step in adopting a secure coding standard is to decide on its primary goals: whether it's going to be applied to all code or to just some of it; whether it's going to be enforced, and how. Is it going to be just a set of recommended guidelines, an informative document, or is it going to be a mandate that will be enforced or strictly enforced? Will it apply to existing code or just newly developed code? Will it be automatically enforced by tools?

So with these answers -- with answers to these questions, we will be in a position to limit the set of available standards to choose from. I think starting by deciding that we'll develop a whole

Copyright 2012 by Carnegie Mellon University

new standard from scratch is -- even though it's possible, I wouldn't recommend that approach. And we certainly ruled that out early on because of the effort involved in such a project.

The next step is to choose from one of the available standards, provided you decide not to develop your own like Cisco did. It's helpful to have representatives from the engineering community within the organization participate in the decision, to give the engineers a sense of ownership.

The next step in the decision or in the adoption of a coding standard is to formulate a process for the socializing of the standards in the engineering community. It's probably not going to be enough to just send out an email with a link to the document. Engineers will need to be trained on how to use the standard effectively. So training will be essential. But training is time consuming so-- and costly-- so a good and efficient training plan will be essential as well.

We find that it's helpful to introduce the standard informally first, maybe by tech talks or brown bags. Again, some form of active participation from the engineers within the organization would be great here, to get them engaged.

And then the standard can be effectively deployed in several phases. In the first phase, the standard can be considered to be simply an informational set of guidelines, without any enforcement and compliance being voluntary -- although obviously strongly encouraged. This helps to socialize the standard and also iron out the kinks in it if there are points of contention -- for example, if not all the guidelines are applicable in the organization.

In the second phase, we can deploy, develop and deploy automatic checkers to detect non-compliance. Checkers can be developed for compilers or for static analyzers. We can put together reports, automated reports, to alert managers about products or software that fails to comply. We can put in place incentives or penalties to help drive up compliance.

And then in subsequent phases we can formalize the process by requiring software programs at inception to commit to a specific degree of compliance based on a given number of violations. We can cap, provide a cap on the maximum number of violations of a given severity. We can deploy mechanisms to prevent non-compliant code from being committed to the code base. We can perform random audits to detect non-compliant software where it's clear automatic checkers failed.

Part 3: Think Like an Attacker; Connect Vulnerabilities with Software Defects

Julia Allen: One of the things that I've always found intriguing, because I've been working in the software assurance space for some time myself, is typically software developers don't think from a security point of view. They don't know how to think like an attacker. They think about the features and the functionality and the performance. But they don't necessarily think, other than maybe in their test case design, they don't think about ways in which their software can be broken or compromised in unexpected, malicious ways.

So I'm just curious in your adoption process did you find that it was--how challenging was it to get your software professionals to begin to think like an attacker so that they could appreciate the benefit of some of the practices and guidelines you were recommending?

Martin Sebor: I think by our experience it's the same. You're absolutely right that most engineers tend to think first about the performance of the code and they do not think like attackers. It's fairly common to introduce assumptions into the code base, based on the

Copyright 2012 by Carnegie Mellon University

expected path, the happy path, through the program. And so it takes a change in the mindset of the engineer to appreciate the fact that the happy path isn't necessarily the path that's going to be taken by the attacker.

One of the effective ways to demonstrate this is by, for example, some engaging talks, the lunches I mentioned, or in labs, where examples of vulnerabilities are explored and attacks are demonstrated. That really opens up the eyes of most engineers who attend those talks, and it allows them to think like attackers more so than just like engineers who are focused on performance, as we said.

Julia Allen: One of the things that I've always speculated about but haven't had the time to go out and get any good hard data is that one of the business case arguments for adopting a secure coding standard and other software security practices is to be able to connect vulnerabilities that were found in operational systems that led to a significant compromise and impact and see if they connect back to the absence of a specific secure coding or secure development practice.

And I was just wondering if in your team you've done any of that type of analysis. I know you did mention that you use vulnerabilities to help socialize your engineers. But have you actually done case analysis on operational vulnerabilities and driven those back to a root cause defect that was injected earlier in the lifecycle?

Martin Sebor: We certainly have done that. We haven't done it to the extent of identifying violations of the coding standards as the root cause of these vulnerabilities. We're at a fairly early stage of deployment of the Secure Coding Standard at Cisco.

So while we are certainly planning on providing this type of analysis, we haven't quite gotten to the point of putting all the infrastructure in place to make it possible at this stage. But it is something that we're actively discussing and we're planning to deploy in one of the subsequent stages of the project.

Julia Allen: Well that's great because when you get to that point, I'll check back in with you and maybe we can have some discussion about that as a follow-up to today's podcast -- if that'd be alright with you?

Martin Sebor: It sounds like a good plan.

Julia Allen: Oh excellent. Well do you have some places where our listeners can learn more about this work? We'll certainly include in our show notes a link to CERT's work and the wiki and other sources you mentioned in your work with Robert Seacord. But are there some other places that you'd like to recommend that our listeners take a look?

Martin Sebor: The Cisco Secure Coding Standards are internal to Cisco. So I can't share links to those with the listeners. I think the CERT Secure Coding wikis are probably the best place to find out about the cutting edge development on those standards.

Julia Allen: Well I'm so appreciative of your time and your expertise, and all the great examples that you've provided for our listeners as they find themselves in a similar situation to Cisco. So I thank you very much, Martin, for your time today.

Martin Sebor: Thank you Julia.

Copyright 2012 by Carnegie Mellon University