Raising the Bar: Mainstreaming CERT C Secure Coding Rules
Transcript

Part 1: How the Specification Came to be and Its Structure

Julia Allen: Welcome to CERT's Podcast Series: Security for Business Leaders. The CERT Program is part of the Software Engineering Institute. We are 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 conversations are available on our podcast website.

My name is Julia Allen. I'm a principal researcher at CERT working on operational resilience. I'm very pleased today to welcome back my colleague, Robert Seacord. Robert leads CERT's Secure Coding Initiative. And today we have some really exciting news to talk with you about, which is the release of a new International Standards Organization technical specification, which is based on the work of Robert and his team.

And I'll just give you the full title. It's "Information technology -- Programming languages, their environments and system software interfaces – C secure coding rules". It's a bit of a mouthful but it tells it all.

And for our listeners' benefit there are also two previous podcasts on Robert's work, and I'll include links to these in the show notes in case you want to go back and learn a little bit more about the background.

So with no further ado, welcome back to the podcast series, Robert. Glad to have you today.

Robert Seacord: Thank you.

Julia Allen: So just so summarize, in your past podcasts we've talked about making the business case for secure coding practices, how to get these into active use. So today, I'd really like to focus our conversation on how you move this incredible body of work into an ISO technical specification, which makes it more accessible, obviously, to the international community.

So could you give just to kind of high-spot some of the background and history that got to this milestone?

Robert Seacord: Sure. So we were working on the *Secure Coding in C and C++* back in 2005 and I was writing some material about the _S functions, which were developed by Microsoft and promoted to, it turns out, the C standards committee. So I wound up getting involved with C standardization in ISO/IEC back in 2005.

In 2006, we went to a meeting in Berlin and a member of the committee, Dr. Thomas Plum, approached me with the idea of CERT creating secure coding standards for the C language. So that seemed like a great idea so we went off and did that. We came back to the committee I believe it was 2009 in Delft and suggested that they publish the guidelines that we developed.

So their reaction was that the committee generally wasn't in the habit of endorsing guidelines for end users. So we licked our wounds a little bit, thought about it, and then came back and suggested that we develop requirements for conforming analyzers. And the response we got at that time was, “Well, that's exactly the sort of thing that the C standards committee does.”

So we formed a study group and that study group lasted about three years. We decided to create a work item and then we spent another maybe year, year and a half and just in November, we finally successfully balloted and published a technical specification.

Julia Allen: Well it sounds like, I mean I know you're just summarizing an incredible amount of time and effort but it sounds like this process is not for the faint of heart, correct?

Robert Seacord: Yeah that's true. You have to get a lot of different people involved with a lot of different perspectives and, typically speaking, you know when you're done when no one is happy with the document.

Julia Allen: Yes, the definition of compromise, right, and consensus.

Robert Seacord: Right.

Julia Allen: So just for clarification because I'm not real familiar with this, how does a technical specification as a category differ from a standard in ISO speak?

Robert Seacord: Yeah, so different committees have different sort of criteria for what they're comfortable publishing. When you do work in the languages committee, it all has a lot of consequence. Any standard or technical specification you publish is going to require millions of dollars in investment to comply.

So the C standards committee tends to be pretty conservative. So what they like to do is initially publish documents as a technical specification and that basically signals the industry that this is the direction that they're moving in. And it gives a chance for vendors to start to work on it and conform with the technical specification, potentially identifying any problems that might exist. And then eventually the hope is that it would move into an actual international standard.

Julia Allen: Great. That makes a lot of sense -- give the community a chance to reconcile their actions and prove it and make it more robust before it becomes even more cast in concrete, correct?

Robert Seacord: Exactly.

Julia Allen: So, tell us a little bit about the specification, its scope, how it's structured, maybe a little bit about the contents and then further on in our conversation we'll actually talk about some of the coding rules. So can you give folks a feel for the landscape?

Robert Seacord: Sure, so it's mostly a collection of coding rules. That's really the normative content of the document. So it lists 46 coding rules and these are all violations for which a conforming analyzer would have to issue a diagnostic.

If you look at the C standard, there really is very little requirement for them to diagnose anything. So any warning messages or error messages you see from a compiler are really at the discretion of the compiler vendor and not required to be a conforming C compiler.

So this is really to simultaneously raise the bar as to what any conforming compiler or analysis tool will have to do, but it's also at the same time it's meant to establish a baseline for which you can be guaranteed that if you were to purchase and use one of these conforming tools, you would know for certain that it was at least finding and diagnosing some class of problem.

Part 2: Vendor Conformance; Selecting and Updating Coding Rules

Julia Allen: Interesting. If we're going to cover this later, tell me. So, if the developer of a compiler, developer of an analyzer decides to take this on where they want to be recognized as a conforming organization, do they have to go through some type of formal process to be so designated?

Robert Seacord: This is sort of an interesting part of the standards process. So, ISO doesn't create any conformance test suites so that's really left to the market. And in most cases, it's really left to the vendors themselves to claim conformance or not claim conformance.

We developed a conformance test suite called the Secure Coding Validation Suite and we have put that up on GitHub (https://github.com/SEI-CERT/scvs) so vendors can take that and test their analyzer or compiler to see whether or not it can successfully diagnose the rule violations that we have in the test suite. And that's distributed with the BSD-style license so anyone can pick it up and use it and extend it and so forth.

Julia Allen: That's great, and then once they've subjected their software to that type of testing then they can make their own claim of compliance, correct?

Robert Seacord: Right, and it's really just market forces that take over from then. But in the case of C compilers, it's worked very well and there haven't been any issues really with compiler vendors claiming conformance when they are not conforming because that would create a lot of bad will in the market place and they'd lose market share and so forth.

Julia Allen: Great, well that's probably a much more powerful force than any kind of law or regulation, right?

Robert Seacord: Yeah.

Julia Allen: So let's talk a little bit more, get into the meat of the specification itself. Can you say a bit about how the 46 coding rules that ended up in the specification were actually selected? Because I'm sure you had many others and I know as you briefly describe in your background, you had quite a community of folks involved in the evolution of this body of work.

So how did you actually decide which rules would be in the specifications?

Robert Seacord: Right, well so we started with the rules in the CERT C secure coding standard. And those are available up on our secure coding Wiki and available in a book we publish with Addison-Wesley.

So we went through those and we did an initial assessment as to which of these were analyzable. And we submitted this -- we contributed that document to ISO for use in the standards process.

And then the remainder of the process -- this is really at the beginning of the study group. So we spent three years where I would say mostly the analyzer vendors tried to argue to eliminate all the rules, and the people who were there from the user perspective or security perspective argued to keep rules. So the criteria was really, first, what does this have to do with security?

So the key there is that if the rule -- we would keep rules that would check against programming flaws that have been known to result in vulnerabilities in ordinary non-security critical code. So what we mean by that is if you have security-critical code, say an encryption algorithm, any defect in that code constitutes a security violation.

But that same defect just in ordinary code might not constitute that high of a risk. So the really the gating factor is problems that are so severe that if they occur just an ordinary code for example, buffer overflow, that would be considered a vulnerability.

The second criteria was the how effectively the rule could be implemented. So many of the analyzer vendors involved in the process have great concerns over high false positive rates because users, developers, their customer base tends to reject tools if they're very visibly wrong most of the time. So one good metric was 20 percent, looking for rules that would have false-positive rates at or below 20 percent or generally viewed to be acceptable to everyone.

Julia Allen: Okay, so you actually subjected each of the candidate coding rules to this kind of rigorous examination and it sounds like you had obviously two competing objectives between the security research community and user community and the vendor and analyzer community. So once a rule passed those two gates, it was a candidate for inclusion. Do I have that right?

Robert Seacord: Yeah that's fairly accurate. It's definitely people who are more concerned with finding every violation and those are people, say, in the software assurance community. And then there are software developers who are more concerned about schedule and getting product released.

And so for the most part they want to really only be informed of issues that are real problems so that they don't spend a lot of time chasing down false positives that don't help them improve the quality of their software.

Julia Allen: Right and as you said, “The real test is nobody is happy with the result,” right?

Robert Seacord: Right, right. Everyone is willing to accept it but no one is happy.

Julia Allen: So you talked about this future path of the specification being used and reviewed and beat on pretty hard. Do you anticipate or is there a part of this process that allows you to take rules out of the specification and maybe add new rules? How does change management happen on the specification?

Robert Seacord: Well there are a couple options and we haven't really made decisions about how to go forward with this yet. We are still taking our victory lap from getting this thing done. But one possibility is that we’ll look to produce a second edition of the technical specification.

Another possibility is that we'll just wait for the analyzer vendors to try to implement the technical specification. We'll get feedback from them as to which rules turn out to be viable, which rules turned out to be maybe too much of a stretch. And then we could make revisions to the document and then we have a lot of options.

If we're pretty confident in what we've discovered we can just move directly to make that an international standard. Or if we still have doubts because we've discovered that we weren't as close as we thought, then we might decide to have a second edition of a technical specification and give the industry again more time to validate that it's the right document to standardize on.

Part 3: Example Coding Rules and Resources for Addressing Violations

Julia Allen: Great. Okay so let's launch into a discussion of a couple of the rules and I'll just warn our listeners that this is actually best done as a visual exercise, a reading of the specification and some of the sources that Robert mentioned, but we're going to give this a try in audio.

And have, if you could Robert, just pick a few of your favorites that are fairly easy to describe in this medium and then hopefully if folks are interested they can go take a look at your resources and the specification. Start out with one of your favorites. Which one would you like to pick?

Robert Seacord: If you want me to start with my favorite, it's got to be overflowing signed integers. So signed integer overflow is actually undefined behavior in the C language and people frequently misunderstand the consequences of that.

If there's undefined behavior anywhere in your program, a conforming compiler basically has complete latitude to do whatever it wants at that point. And so some of these undefined behaviors are more colloquially known as "buffer overflows" and various sorts of exploits.

So the signed integer overflow was a really contentious kind of rule because it's really endemic in most C code that gets written today. Programmers really don't have a very good understanding of integral behavior. So we wound up making this a rule that depends upon tainted input. So we'll only diagnose signed integer operations that might trap as the result of one or more tainted inputs meaning inputs that are derived from untrusted external sources.

So in fact, this is sort of a subset of the actual problems that could plague your code. And normally you would want to eliminate all of these integer overflows. But the standard, the technical specification, only requires that a subset of these things be diagnosed to make it a more manageable problem.

Some other rules we have -- we have one called "invalid pointer.” This requires that the forming or using of out-of-bounds pointers or array subscripts be diagnosed. And so this is not just the actually then reading or writing memory at an invalid pointer -- just the formation of the pointer itself is undefined behavior in the C language. So we require that to be diagnosed but the obvious risk here, again, is buffer overflows if you’re writing to memory that's outside the bounds of an array object.

Julia Allen: So in terms of the rules that you’ve described so far, the compiler or the static code analyzer or other type of analyzer, if it is conforming with this technical specification, will actually detect these conditions and report them to the developer in some fashion, right?

Robert Seacord: That's right. They would be required to diagnose code that violates these rules. The core C standard does not require that any undefined behavior be diagnosed. One of the reasons behaviors are classified as undefined by the C standard is because they can be difficult to diagnose. So consequently the C standards are written by C compiler vendors for C compiler vendors so they frequently don't like to make things really difficult on themselves.

Julia Allen: So let's say I'm a developer and I'm developing in C and I am familiar with this body of work and I have access to tools that would help me address these coding rules, are there -- assuming that such conditions are diagnosed, what do I do next? Are there actually in your set of resources or other community resources, are there actually solutions to these kinds of issues?

Robert Seacord: Yeah, so we have our own coding standard, which is the CERT C coding standard and that's currently being formalized and published as a second edition with Addison-Wesley so that will probably be available early in the spring (2014).

So the technical specification is written for analyzers. So we'll have examples of code that needs to be diagnosed and we'll have examples of code that does not need to be diagnosed. But in the second case, those examples are really meant to illustrate kind of the edge conditions.

So very narrowly this sort of code needs to be diagnosed even though it's close to correct and this other type of code doesn't need to be diagnosed even though it's close to wrong. So really it's to illustrate what is the precise boundary of code that needs to be diagnosed to conform to the technical specifications?

So in the CERT C coding standard, which is really targeted towards developers, we've made some effort to make these two documents consistent. And there we'll have the corresponding rule. You'll have the noncompliant example, which now is just there to show "here's how not to write code." And then we'll have compliant solutions and those compliant solutions are meant to be very safe, secure implementations of the same functionality that you could theoretically just cut and paste and plug into your system.

Julia Allen: Oh fantastic, okay. Thank you, that's a great explanation and hopefully provides some comfort to those that are going to be faced with trying to make this all work in their software and systems.

So thank you for those examples. I just have a couple more quick questions for you. You did talk a little bit about what's going to happen next for the vendor community. Did you want to elaborate on that at all or have we pretty much covered that?

Robert Seacord: I think we've covered it pretty well. We are still in contact with everyone from the study group and we are getting incremental updates as to what, how far along they are in their support.

So hopefully the next set of announcements you'll see is vendors claiming either full or some limited compliance to the technical specifications.

Julia Allen: Are you an active, I mean you said you've pretty much stayed in contact with the study group members. So would you envision that your team will continue to interact with the vendor and compiler community as they start to adopt these rules?

Robert Seacord: Yeah, we are still very much involved. One of the gentlemen who chaired the study group for this is David Keaton. He's a member of my team. He's also the current PL 2211 Chair so he basically heads the U.S. delegation to the C standards committee. So we are very active in this community and we're going to continue to be involved in all aspects of security and in C and other programming languages.

Julia Allen: Great, great. Well, Robert, you mentioned throughout our conversation today quite a few sources for folks to refer to if they want to find out more. Are there any others that you'd like to highlight?

Robert Seacord: The technical specification's available online from the ISO store. We have the existing CERT C Secure Coding Standards book. There's a second edition, again, that will be out in the spring and has been updated for the C 11 major revision of the C standard as well as for consistency with the TS 17961.

We have the secure coding website and Wikis and for more, just less of a reference material, more of a pick it up read front to end, you can pick up a copy of *Secure Coding in C and C++.* There's also a second edition of that book which was just published in January of this year (2013).

Julia Allen: Well Robert, hats off to you and your team. This is a fantastic accomplishment that as you said it's been a seven-year journey and it sounds like the journey continues. But I really want to congratulate you and your team, and I thank you so very much for your time and preparation for our conversation today.

Robert Seacord: Thank you.