Software Security in Rust

featuring Joe Sible and David Svoboda as interviewed by Suzanne Miller

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Suzanne Miller: Welcome to the SEI Podcast Series. My name is Suzanne Miller. I am a principal researcher in the Software Solutions Division. Today, I am very pleased to have my colleagues Joe Sible and David Svoboda from the CERT Division here to talk about a language that has some interesting security implications called Rust. Before we get into that, however, I know David has been with us before, but Joe, you have not. So I am going to ask each of you to just give a little bit of a bio and tell our audience what it is that is so cool about working at the SEI. Joe?

Joe Sible: My name is Joe Sible. I am a software engineer on the Applied Systems team. My duties bounce around a lot—sometimes it is just writing code, sometimes it is helping fix our servers. I enjoy working here a lot. It is a great place to be.

Suzanne: What did you do before you came to us?

Joe: This was my first job right out of college.

Suzanne: So whatever we tell you is the way the world is.

Joe: Basically.

Suzanne: I like that. David, you have been around a little bit longer than that. Tell us a little bit about yourself and your background.

David Svoboda: Just a little bit. I actually joined back in 2007 or so when the Secure Coding group was just getting started. They built the secure coding standards for C and Java and Perl and C++. One of the nice things I like about the SEI is that I am a pretty smart guy, but there are lots of people smarter than me around that can help me learn and adjust. Hi Joe!

Suzanne: Speaking of smart people. We are talking about a language today, and there are probably hundreds of languages out now that have various functions. The one we are talking today is called Rust. Why is Rust important, and why is it a catalyst for more security-related work in comparison to some other languages? David, do you want to start us off with that?

David: Yes. In the big picture, there has been a lot of…. Well, I tend to think of languages as competing with each other on the world stage, and lots of languages have kind of competed with C, with C++ being the first contender. What we discovered over the years is that most of these languages did not really compete with C in the embedded world where you have limited memory and your program needs to run really, really fast. Lots of languages, if you do not really need speed and performance, then they are easier. Python and Java are two good examples. So in the embedded world, C is almost completely dominant. C++ has a small niche role but the other languages are pretty much absent in the embedded space. Rust is the first language that really looks like it can possibly compete with C in the embedded space.

Suzanne: So what you are telling me is we are done with Ada? For our audience that is younger than I am, Ada was a language that was promoted for embedded systems in the 80s, and it turned out to not be as good an idea as we thought it was.

David: It was a good idea in theory. In fact, I did apply for a couple of jobs in Ada back in the late 80s. They never managed to get C-level performance out of Ada.

Suzanne: That is a good way to say it.

David: You might say the safety guarantees that Ada offered killed its possibility of performance.

Suzanne: So we have a competitor to C after 30 years?

David: After 30 years.

Suzanne: After 30 years. So what is it about Rust that makes it a competitor to C?

Joe: In the old days, you had C, which is dangerous but fast. Then you had a bunch of garbage-collected and safer languages like Python and Java that were slow. Rust is one of the first languages and the first one that is gaining significant traction, where it does a lot of checks at compile time to make sure things are safe. So at run time, it is as fast as C, but it still gives you the same safety guarantees or better that the other languages have given you in the past.

Suzanne: That is important.

Joe: And the main way it has done that is a thing called the borrow checker. That is basically the brilliant thing it has brought to the table. And even other languages like Ada are looking at picking it up because it is what has given the ability for all the new safety…everything stems back to that.

David: Yes, perhaps if Ada had had a borrow checker, it might have succeeded.

Suzanne: Tell us a little bit about what that is, if we have people that aren't familiar with that concept.

David: Do you want to take this one, Joe? You can probably explain it better than I do.

Joe: The borrow checker basically enforces two rules. One rule is, make sure programs do not try to use any memory after it is freed. That is a big source of vulnerabilities in C. And the second is, make sure there are never two different parts of the program that can edit something at the same time.

Suzanne: OK. Alright. Those are things that are, I mean, secure coding practices. Do not use the memory after it is freed. Do not write to the same…I mean, you are basically giving tools that allow secure coding practices to be

implemented even if the programmer is not as aware as they should be of those practices.

David: Right. In C, we have a secure coding rule saying, *Don't do this.* There are various analysis tools that will try and detect if you are doing that, but they do not always get it right. And so it is easy, it is possible and unfortunately, and very easy for you to ignore the rules, and then you are in dangerous territory. And Rust, as Joe explained, tries to prevent that all at compile time, which means that at run time, you can just worry about performance.

Suzanne: Yes. I love that and it.... To my mind, it is like, why do more languages not do things at compile time because that is when you have the time, if you want to think about it that way.

David: C++ has been trying to do lots of things at compile time, but not security.

Suzanne: Let's move to talking about security specifically, because the blog post that you wrote recently is really about what are some of the security promises of the Rust programming language. What kinds of issues did you explore? I know Joe, you were involved in this piece, in particular.

Joe: The big piece of the blog post I went into was giving examples of C and C++ code with the kind of common mistakes that lead to security vulnerabilities, then giving a transliteration of that code in Rust and showing how the compiler will not even let you compile it until you fix them.

Suzanne: Nice. I like it. I like it a lot. I think I like this language. You also looked at security limitations, because no language can do everything. What are some of the things you ran into in terms of limitations to how Rust can help you with security issues?

David: There are several ways of looking at that. First of all, of course Rust has this notion of safety. And like Joe said, the safety prevents you from using memory after it has been freed and it prevents you from race conditions [**NOTE**: Rust protects you from data races but not other kinds of race conditions. A data race is a type of race condition. It is a race condition where the shared object is one or more memory addresses. Race conditions can exist over other shared resources, such as files. So all data races are race conditions, but not all race conditions are data races.], which is two separate parts editing the same memory. However, there are many other things that

people tend to think of as safety issues, such as it not depleting your bank account to zero dollars when it says, if you like have $1,000 in there, or such as say if you do some floating-point arithmetic and you wound up with the answer of 4.9999999 and say, *Is that equal to 5?* Well…

Suzanne: It depends on the context.

David: Right. That safety might be critical if you are talking about, if you are trying to figure out if your rocket should cut power. So Rust's notion of safety, it is internally quite consistent, but it does not necessarily match with your or my definitions of safety.

Suzanne: Right. And so any user of Rust needs to look at the context of use. And especially for embedded, a lot of our embedded software systems are very safety and security focused, so we have to be careful that we are not ignoring the vulnerabilities that Rust still has.

David: Right.

Joe: One rule of thumb for what it fixes and what it does not, the kind of vulnerabilities that you only tend to see in C programs, the memory safety and memory corruption, it will take care of. Other vulnerabilities like forgetting an "if the person's an admin" check it won't; that can happen in any language.

Suzanne: Sure, sure.

David: Yes, in fact, one thing I note is the top 25 CWEs is a set of safety or security problems, and Rust only covers 7 of those top 25. The remaining 18 is still up to you. A lot of them are web-based, so they are not the kind of thing that you encounter in C, and that is why it is so low; it would certainly be higher if it was not focused on web issues.

Suzanne: And just for our audience that do not know what CWEs are, explain that acronym for them.

David: Sorry. CWE is common weakness enumeration. It is a way of partitioning the set of all vulnerabilities into subcategories, so there is one CWE for use-after-free, and there is one for say dereferencing null pointers, there are several for buffer overflows. There are a whole bunch of category CWEs, like here is one for just general code-quality issues. So it is a common way, it is a lot like the CERT secure coding standards of identifying

vulnerabilities. However, CWEs, unlike CERT standards, have become widely known in the DoD.

Suzanne: That is something that we can reference in the transcript if people need more information about those. Are there other things about Rust that you want potential users to know about—some things that are really cool about Rust or some things that you go, *And oh, by the way, do not ever try and use it in this context*? Things like that that would be interesting for our audience.

Joe: To build on what David was saying about null pointer dereferences as a CWE, the concept of null—these days it is called the billion-dollar mistake. That refers to, there is an estimated over a billion dollars' worth of productivity loss and problems caused by the fact that null is a thing that exists.

David: In C and Java and Python and so many other languages.

Joe: Rust is fixing the billion-dollar mistake. They do not have a general concept of null that anything can be, unlike C where any pointer can be null or Java where any object can be null. In Rust, by default, everything has to have some valid value of the type in it. If you want something to be optional, you have to wrap that thing in an Option, then it can either be *None*, the equivalent of null, or *Some* if it has something in it. And the compiler enforces for types that are wrapped in Option, that you check it everywhere before you use it. And for types that are not, that there is always something in it.

Suzanne: So we are trying to take the billion-dollar question down to maybe a few cents.

Joe: Yes.

David: Yes.

Suzanne: I like it.

David: There are maintenance costs that you have to take to get your program to work right, but then you are not going to have to worry about null pointer dereferences or anything like that again.

Suzanne: OK. Alright. That could be useful in some settings. Are there settings where you would recommend against using Rust?

David: There are. As we have said, Rust is a very performant language and uses little memory. But if you are, say, using Java, and you are happy with it, you are happy with the slower performance and the greater memory usage, then you would probably not be happy working with Rust. Rust does impose a cost on you, a maintenance cost, of making sure that your memory is free when done, and you do not have race conditions and use-after-free problems. This does impose some maintenance overhead, which Java and Python do not impose. So if maintenance is the most important thing for you, more than security or performance, then you would not be happy with Rust.

Suzanne: Thinking about new users to Rust, what kinds of things that we have done at the SEI or that you want to reference externally would help them to get familiar with Rust and help them to understand if Rust is relevant to their situation? I know we are going to be doing some other things with vulnerabilities on another podcast related to this, so really just focusing on what are the things that users have available to them to become familiar with using Rust productively?

David: What is it, rust-lang.org website?

Joe: Yes, rust-lang.org.

David: That will be the place to start that has an extensive reference manual. I enjoy the rustlings course, which is actually a project on GitHub with about 100 small Rust programs. Each one has some failure, and your job is to find it, to fix the failure. It has a server that simply detects when you change the file, so it tries to run it and say, *This program is still not working* or *This program is working, congratulations. You can move on to the next program*.

Suzanne: So a little gamification going on.

David: Yes, it gamifies the learning process.

Joe: And the Rust documentation is excellent. So rust-lang.org, as David said, it is not just a reference. There is also a Rust book, which is an introduction. It is almost like an introductory course you walk through. It has live code examples in the book you can run in place and follow along with. Also, Rust by Example, which is a little bit more practical and hands-on, teaches you the same thing. Both are very much worth going through. That is all it took to teach me Rust. I didn't need any formal course beyond that.

Suzanne: We will make sure that all of those are referenced in the transcript so that people do not have to take notes while they are…

David: Our blog post did cite rust-lang.org.

Suzanne: Good, good, good. So Rust is a new language. You have been working with it. What else are you researching? What is next for each of you in terms of either Rust-related work or other work in your security areas that you are both working in?

David: For me, this is not exactly Rust-related, but my main project now is [an] automated repair system that can say, *Hey, you have a null-pointer vulnerability in line 10. Let's just fix this automatically so that the program will never complain of a null pointer again*. If you have like 1,000 null-pointer vulnerabilities, it fixes all of them at the time. This is a two-year MTP project that I am working on. Hopefully in two years, we will have a tool that can do that and that will be useful to the DoD.

Suzanne: In particular, yes. Although that is still useful to a lot of other people.

David: Yes, but the SEI's main, one of the SEI's main missions is to serve the Department of Defense, so that is why I am focusing on them.

Suzanne: And just for those that do not know, an MTP [medium term project] is a particular type of research project here at the SEI. Joe, what about you? What are you working on?

Joe: My next security topic that I might do a blog post or podcast about, also not really related to Rust, is looking at ways to keep network middleboxes from telling what websites people are going to. And the upcoming keywords for that, the technologies in use, DNS over HTTPS and encrypted client hello.

Suzanne: Encrypted client hello, I like that.

David: Not encrypted client hello world.

Joe: Both of which Cloudflare is kind of the driving force behind.

Suzanne: So we have something to look forward to. I really want to have a podcast that is titled Encrypted Hello. I want to thank both of you for joining us today and talking about this. I think that understanding that there are

alternatives, especially to some of our standard, embedded-software language C, is important to people who are trying to make more secure and safer code, so I think there will be a lot of interest in this out in our community. I will look forward to future times when I will be able to talk to you about other kinds of things, your automated repair and the kinds of Cloudflare research things that you are getting into Joe, so thank you very much.

I want to remind our audience that you can get this podcast anywhere that you get your other podcasts. You can get it on SoundCloud. You can get it on Apple. You can get it on, my personal favorite, the SEI YouTube channel. I hope that you will access this work and that you will get benefit from the work that these guys are doing. And with that, I will say thank you.

Thanks for joining us. This episode is available where you download podcasts, including SoundCloud, Stitcher, TuneIn Radio, Google Podcasts, and Apple Podcasts. It is also available on the SEI website at sei.cmu.edu/podcasts and the SEI's YouTube channel. This copyrighted work is made available through the Software Engineering Institute, a federally funded research and development center sponsored by the U.S. Department of Defense. For more information about the SEI and this work, please visit www.sei.cmu.edu. As always, if you have any questions, please do not hesitate to email us at info@sei.cmu.edu. Thank you.