Is Java More Secure Than C?

Featuring David Svoboda as Interviewed by Suzanne Miller

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. Today's podcast is available on the SEI website at sei.cmu.edu/podcasts.

My name is Suzanne Miller. I am a principal researcher here at the SEI. Today I am very pleased to introduce you to David Svoboda, a researcher on the CERT Secure Coding team. In this podcast, we will discuss his recent analysis of the Java and C coding languages to see which one is more secure.

First, a little bit about our guest. David is a software security engineer at CERT SEI. He co-authored and contributed to four books including, *the CERT C Coding Standard* and *the CERT Oracle Secure Coding Standard for Java*. David has over 15 years of job and development experience starting with Java 2, and his Java projects include Tomcat Servlets and Eclipse plug-ins. He also maintains the SEI CERT coding standard wikis, and he has taught secure coding in C, C++, and Java all over the world to various groups in the military, government, and banking industries.

David is also involved in several ISO standards groups, including one for standardizing C and one for standardizing C++. He has been the primary developer on a diverse set of software development products at Carnegie Mellon University since 1991. His projects have ranged from hierarchical chip modeling and social organization simulation to automated machine translation known as AMT. His KANTOO AMT software, developed in 1996, is still in production use at Caterpillar Industries. Welcome, David. You are a busy guy.

David Svoboda: Hello, Suzanne.

Suzanne: You did this analysis on the security of the coding rules for C language and Java Language, and you posted that in a blog, and had more than 50 comments from Hacker News.

That is a different kind of resource than the SEI Insights blog, but that is pretty spectacular to get that kind of conversation going.

Why don’t you tell us a little bit about why you decided this analysis was needed, and why you think this has generated so much discussion within that part of the community.

David: The analysis actually started because of an argument I had with my boss. A little bit of history: we developed the first version of the C coding standard in 2008. Then we started to work on the Java standard, and that was first published in 2011. The Java standard was actually proposed to us by a student who said, *Your C standard is cool. Let’s build one for Java*. We thought, *Sure, this will be easy. Java is a secure language. You are not going to find any rules*. The student proceeded to do what students always do best, which is show us just how spectacularly wrong we were.

Suzanne: OK. From there…

From there, we wound up with about 100 rules in C and 170 rules in Java. My boss basically suggested, *Gee, Java has so many more rules than C. I thought it was a more secure language*, implying that the number of rules was inversely…

Suzanne Miller: Was directly correlated…

David: …with the actual security of the language. Now, of course, neither he nor I think that is academically rigorous, the number of rules. It is still strongly intuitive. It is still a pretty important metric because it indicates how much you have to learn in order to program securely.

Suzanne: Right. You have got more rules in Java than you do in C if you want to create secure code.

David: Well, that was the argument. The point of this analysis was to try and refute that argument that Java is not a less secure language just because it has more rules. That is why we did the analysis.

Suzanne: Is that what you found when you actually completed the analysis?

David: That is exactly what we found when I completed the analysis. First of all, as I said, if you look at this first bar graph, you can see that Java has more rules overall then C, so I decided to try focusing on the most severe rules. All of our rules have a severity metric. The highest value in the severity metric indicates a rule where, if you violate it, someone can take over your computer.

Suzanne: So, the higher the severity metric, the more important that security rule is to follow.

David: Exactly. The more critical it is to follow, the more you are going to lose if someone attacks you there.

Suzanne: If you violate it.

David: The lower security rules can only enable an attacker to crash a program, but not necessarily…

Suzanne Miller: Minor details.

David: Yes. Minor details. It is easy to restart such a program. Anyway, the point is that the red in the graph here shows the number of high-severity rules in both C and Java. As you can see, they are very close.

Suzanne: And, actually, C is a little bit higher according to this.

David: C is indeed higher, but it is only 32 versus 29 rules.

Suzanne: It is not a huge number in terms of…

David: Exactly. Not significant.

Suzanne: When you translate into practice, it is not going to make a huge difference.

David: At that point, I was saying, W*hat gives*? *We thought Java was a more secure language. Why are there just as many high severity rules for it as there are for C*? There are plenty of types of vulnerabilities in C that you do not have to worry about in Java, such as memory corruption. Buffer overflow is a problem almost exclusive to C. A couple of languages like C++ have it, too, but Java was designed to prevent buffer overflows and memory corruption. All of those C rules should have gone away and not appeared in Java. We don't see that at all in this bar graph.

Suzanne: That is not visible. So what is up with that?

David: What is up with that? That was the main focus of this analysis. The next thing I did was I simply took the high-severity rules and I said, *Why is this rule high severity?* We categorized them and they fell into four categories that you can see in the two pie charts. In the two pie charts here, I wound up with four categories for C and four categories for Java. As you can see, in C the biggest category is memory corruption, and the biggest Java category is internal privilege escalation. Again, the intuition should hold. Memory corruption is the biggest part of C, and there is no memory corruption in Java. What is going on?

A large part of the blog is spent addressing what these categories are. *What is memory corruption, privilege escalation, and so on?* The main point to note was, actually, the privilege escalation. Both C and Java have rules about privilege escalation, but privilege escalation is a different problem in both C and Java. First of all, privilege escalation itself just simply means that you have a privilege system, such as a lock on your door, and then a privilege escalation vulnerability is as if someone is able to get past your door even though the lock is supposed to keep them out.

Suzanne: Even though they don't have a key.

David: Yes. I have a key to my office door, but that doesn't let me into your office or most of the other rooms in the SEI. To have a privilege escalation vulnerability, you have to have a system of privileges. If we took all the locks off here, then my skeleton key wouldn't get me anywhere forbidden. Taking all the locks off will have other problems we don't need to worry about.

Suzanne: Other security issues.

David: Right. The difference between C and Java is that Java has its own privilege system that it uses to chaperone things, like applets. You are able to run a Java applet on your computer that… doesn't have code that you necessarily know or trust.

At the SEI here, we have used Java applets to do refresher courses on new policies, you know, recognizing terrorists, sexual harassment, that kind of thing, refresher courses that everyone here is supposed to take. Those have been using Java applets even as recently as last year. Those Java applets are kept secure by Java's internal privilege system. Now, C has no internal privilege system. There is nothing defined in the language, so if you write a C program and I run it, that program can do anything…

Suzanne: There is no chaperone as it were.

David: There is no chaperone. Exactly.

There are external privilege systems that most operating systems provide. If you have ever tried to run a program or do something and your computer pops up and says *I need an administration password to do that*, that is the…

Suzanne: We see that a lot at the SEI.

David: Sometimes, I even know the password to use [researcher being droll here]. Most of the time, I don't. Windows and Linux, other computers, will have these privilege systems, but they are not part of a programming language.

Suzanne: As a programmer, I not only have to know my programming language rules, but if I don't have rules that cover situations like privilege, I need to also know how to use the rules that are provided by the operating system.

David: Right.

Suzanne: That adds a little bit of a layer of complexity in terms of how I use that language.

David: The rules with regard to an external privilege system apply to both C and Java, but in C, they are just a small wedge. Well, we have not put them into Java simply because these rules were part of the section of a chapter in the C secure coding standard that dealt specifically with POSIX [Portable Operating System Interface]. There is a handful of rules as well for Windows, but in Java the big rules that we have here in the big wedge, the big blue wedge, deal with jobless internal privilege system.

Again, what you have is both languages have rules about privilege systems. What happens is, if you take those privilege systems away, then you wind up with the second bar chart where you still have a bunch of rules, but suddenly there are many less high-severity rules in Java than there are in C. That means that if you are not running a program that interacts with a privilege system, you do not have to worry about that category of rules.

Suzanne: You have fewer rules to worry about.

David: You have fewer high-severity rules to worry about, exactly. The upshot is if you are not working with Java's internal privilege system—by that I mean you are running a Java desktop app or mobile app, or even an applet—then you do not have to worry about all of those privilege rules and the amount of high-severity rules you have to have is this little red sliver as opposed to that big red stripe that you have if you are working in C.

Suzanne: When you were conducting this analysis, are there things that surprised you that you expected to find but were not the way you had expected in the beginning?

David: Like I said, in the first bar graph, we see that the number of high-severity rules in C and Java were about equivalent.

Suzanne: That was a surprise.

David: That was a surprise. It was counterintuitive. I expected there to be a lot less in Java because, again, we don't have that whole memory corruption section.

Suzanne: This is why we do these analyses, so that we can check on our intuition and understand what the real data is.

David: The analysis was trying to give some scientific rigor to our intuition.

Suzanne: How can developers use this system? What kinds of things are you seeing on the different sites that are discussing this? What do they see as the major takeaways from this?

David: The major take away, again, is in this third bar chart. That is, if your program does not require escalated privileges, if you are not working with less privileged code, you have less high-severity Java rules to worry about than C rules.

Suzanne Which theoretically should make it easier to write secure software from Java and those kinds of applications.

David: It should be much easier to write software that does not get pwned. That matches our intuition that Java is a more secure language than C.

Suzanne: You have other things besides these kinds of analyses you are doing. What are some other projects coming up from the Secure Coding team that people have to look for?

David: Many other things. Right now I am doing four projects. Half of my time is spent working on a project called Source Code Analysis Laboratory (SCALe). Well, it is a very simple business model. You give us some code and we will tell you if this code violates any of our secure coding rules and exactly where. That includes the C rules and the Java rules, depending on your code's language. We will basically give you a report saying, *We found the following problems*. If we don't find the problems, then we will declare this code is compliant with our standard and give you nice little certificate and put you in a registry. Hopefully, with our certificate, you can use that as marketing to say, *Our software is secure. It passed the CERT C SCALe standard*.

Suzanne: As we get into more Internet of Things and more kinds of not-as-common areas that we think about software being an impact, these kinds of certifications are one of the ways that people can gain trust that, *Even though I do not expect people who build light bulbs to understand coding, they at least are applying secure coding practices and things like that*.

David: The big problem with security is if you have to choose between two software projects for your next purchase, are you going to choose the one that is more secure or the one that is less secure? To answer that question, you have to be a security expert. Most people are not. Most people go with the one that is cheaper, the one that has more features. Being more secure is not a salable point.

Suzanne: It's not in the public eye.

David: Exactly. It's not a salable point. If the more secure product can give a certification, can say, *Hey, Here is a CERT seal of approval*, then that becomes a marketing feature.

Suzanne: That is one of only one of four projects you are on?

David: That is only one of four projects. The second project, it is a LENS [Line-funded Exploratory New Starts] project to try and help improve SCALe and software analysis by guessing which diagnostics from a tool are likely to be true and which ones are likely to be false positives.

The problem with most analysis tools that look for vulnerabilities and bugs is that they tend to report problems that are not really there. We call those false positives. We hope to be able to take a look at a code base and a set of these reports and say, *Which ones are false and which ones are true?,* which would help an auditor. It would not be as good as an auditor. A human being would still need to look at it, but they can prioritize.

Suzanne: It helps them hone in on the things.

David: They can focus in the things the machine thinks is true, and which ones the machine thinks is false, they can discard those. That is another project, and that is also taking half my time.

Suzanne: You are a classic software engineer, David.

David: The third project is to help finish the CERT Secure Coding Standard for C++. We have had a C++ standard in the works for several years now, and it is our goal to finish it. That is taking up half of my time.

Suzanne: We are up to three halves. Go ahead.

David: The last project is automated code repair. That simply takes the notion that, well, *We have a program, and we know that has a vulnerability on line 46. Is there some way we can automatically fix this without having to force a developer to go through and manually do it*?

Suzanne: Oh, that is interesting.

David: Or, suppose it has 100 vulnerabilities on lines 46, 52, 59; asking a developer to fix one vulnerabilities pretty simple. Asking them to fix 100 is going to take a long time.

Suzanne: Especially with all the interrelationships that are likely.

David: Exactly. If we can write a tool that would do this automatically or would do this thing, *Would you like to fix these five things? Just press this button*. Then that would raise the state of coding and reduce developer costs.

Suzanne: That one is going to take more than half of your time.

David: Well, it is currently slated at, I think, 30 percent of my time. That puts me at, what? 180 percent?

Suzanne: Let's not go there. Like I said, *You are like every software engineer I know*. You take on all the interesting things, but it takes time to do all of them.

Suzanne: David, I want to thank you for joining us today. This is the kind of work that the SEI is known for doing, for providing this kind of analysis and validating or contradicting our intuitions with real data. That is a great service.

If you would like to find out more about David's work in this area, insights.sei.cmu.edu is where you will find the blog that got all this discussion started.

If you want to listen to the podcast, it will be available on sei.cmu.edu/podcasts [and on Carnegie Mellon University's iTunes U site] along with the transcript. Thank you for listening. Thank you for watching. [As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu.]