Undiscovered Vulnerabilities: Not Just for Critical Software

Featuring Jonathan Spring 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, and I am a principal researcher in the SEI Software Solutions Division. Today, I am joined by my friend and colleague, Dr. Jonathan Spring. Dr. Spring, known to all of us as Jono, is a senior vulnerability researcher in the SEI CERT Division. Today, he is here to talk about a recent paper he published analyzing the number of undiscovered vulnerabilities that remain in information systems. He has also been a frequent guest on our show to talk about his other work here at the SEI.

Welcome, Jono. It's good to see you again.

Jonathan Spring: Thanks, Suz. Nice to see you.

Suzanne: For those of our audience who aren't familiar with your work, why don't you tell us a little bit about yourself, particularly what brought you to the SEI and the work that you do here?

Jonathan: Sure. I started at the SEI as an intern while I was studying at Pitt [University of Pittsburgh]. One of the adjunct professors there was teaching the security-management class, and I ended up working for him over the summer, and then I didn't leave. I have done network traffic analysis focusing on DNS [Domain Name System], with the SiLK tool suite stuff. I like to say that I basically started asking *Why* too many times, like, *Why does this work?* or *Why doesn't this work?* or *Why do we do it this way?* Eventually, in order to answer that question, I went and did a Ph.D. in essentially philosophy of science and information-security stuff.

Suzanne: Which I find to be a really interesting pairing. Because security and philosophy of science have a little bit different focus. Why don't you speak to that for just a second?

Jonathan: Oh, sure. Security has a different practical focus, but also the network traffic analysis stuff, we often talked about the Internet ecosystem, or the Domain Name System, the DNS ecosystem or whatever. We had a series of papers about the blocklist ecosystem. We are clearly using life-sciences metaphors when we are trying to understand how the Internet works, because even though we built it, no one person can be asked what they did to make it a certain way, and so it ends up functioning, I think, a lot like life sciences, and philosophy of life sciences [has] a lot to do with how we ask questions, or not how we ask questions, but what the right questions to ask are, so that we can learn some general predictable things.

While I was at UCL [University College London], I also sat in the program verification and logic group, by happenstance. It turns out that a lot of practical logicians are in computer science and not in philosophy anymore. That is where I learned some of the formal mathematical modeling stuff that is related to the paper we are going to talk about today. It was actually through the program-verification side of computer science, which is obviously also related to security, but is also different goals, I think. I don't want to get into the safety, security.

Suzanne: Ah, yes. There is another pairing there that has interesting conflicts as well as interesting synergies. I will put it that way.

Jonathan: Yes, I would love to talk to you about that another day.

Suzanne: We have many days ahead of us where we can talk about these things. All right, but for today, in the introduction of your recent paper, you state that you are going to bring computing theory and security operations into the conversation to answer the question—and it is a very important question, in my mind—how many undiscovered vulnerabilities are there in a piece of software? Tell us about the catalyst for this. This is not a *why* question, exactly, but I think there is a *why* question behind it, so tell us a little bit more about this.

Jonathan: It matters because you want to fund and resource your security teams in the right places. If the answer was *We can make it so that there aren't vulnerabilities*, we should spend all of our money on that. We don't have to spend money on vulnerability-management anymore. But the CMMI stuff that I think you are much more familiar with than I am, is related to a bigger set of this, right? But if you could do a process and guarantee that at the end of it, your software wouldn't have bugs, whether that is through OCTAVE or through CMMI or whatever other process, then you would do that. Whether it is program verification, you can automate it. If you can't do that, then you need to invest in a robust and quick vulnerability-management capability.

Suzanne: And giving yourself some confidence that…The *how many undiscovered* is a way of helping people to understand that *I need to manage what I don't know, as well as what I do know*

is kind of how I look at it. If you don't manage what we don't know, then we are really just focusing on a small part of the problem if we only manage what we know. Is that fair?

Jonathan: Yes, that is definitely right, and I think that you see some regulations or policies written around, *Don't deploy software with known vulnerabilities*, which is great. You definitely shouldn't do that. The other side of that question is, *Well, how many unknown vulnerabilities are there? What are acceptable risks around what you don't know about the potential vulnerabilities in the software you are going to be using?*

Suzanne: This leads us into some of the concepts that you talk about in the paper. We are not going to ask you to get into the algorithm level of this. I know there is a lot of that really cool stuff in the paper for those that want to get deep into it. But just at the concept level, you talk about dense or sparse vulnerabilities, and that was a new concept for me. Why don't you explain to us what is the significance, one, that there are dense or sparse, and then what are they?

Jonathan: Some folks in the security community have used *dense* to mean there are always more undiscovered vulnerabilities and *sparse* to mean there are few enough of them that we could potentially find them all.

Suzanne: In a piece of software, there may be aspects of the software that we believe have lots of vulnerabilities, and those would be these aspects of the software that we consider to be dense, and then other places that we think we have got them all or we can identify what is there if we just spend the time. Is it a difference between *I think I can actually address the whole space with this* versus *I don't even know what the whole space is*?

Jonathan: I think it would be the first one. I think that you said this, and I want to emphasize it. This is talking about one version of one piece of software. Once we get into the *Oh, and your software changes every day* discussion, we have to have this conversation every day, which is fine. We're going to focus just for scoping on *OK, one document processor, the one version and the one operating system*, like the one thing. So, yes, your summary about that is correct. Sparse would mean if we put enough effort in, we could find them all, and dense would mean, there is no such thing. The metaphor here, which doesn't hold up exactly, but the metaphor is with numbers. Dense means there is always another number in between two numbers. Fractions are dense. One-half is in between zero and one. A third is in between zero and a half, a quarter, a fifth, a sixth. You can always find another one in between. The integers are not. There is not an integer in between zero and one, because the integers are just one, two, three, four, five, six. This is a useful concept in some mathematical things, but I found that it is not actually super useful for vulnerabilities because vulnerabilities aren't ordered like that.

Suzanne: Well, to a certain extent, people who create intentional vulnerabilities, they are trying to find those interstices. The interstices are where—I love that word—are where they are trying to find opportunity. Now, the unintentional vulnerabilities [are] in some ways even harder, because you don't even know that you found an interstices. We don't even know that it's there. Now we have created an opening for others to access, so…

Jonathan: Yes. That is exactly the right way to think about it. If you are looking for whether there is an interspace that causes some unexpected behavior, if there are infinitely many interspaces and you can't tell me anything about the properties of them, one of them is going to be a bug.

Suzanne: At least one.

Jonathan: Yes.

Suzanne: So understanding dense and sparse helps us to understand if we have a space where we have no hope of ever understanding everything about the vulnerability space, and sparse vulnerabilities, we have a hope for that. But you need a different lens on top of that, to be able to say, *Now where am I going to focus?* In your paper, you talk about understanding the halting problem, which is fundamental computer science theory, and how you can use that to help you with that. Do you want to explain a little bit about how that works in your head?

Jonathan: Sure. The halting problem is about whether a computer will finish calculating something or not. That is the basic question. That seems really simple. You should be able to know ahead of time whether some piece of software, given a specific input, will produce an output or not, not even saying what the output will be, just that it will produce an output. But it turns out that for a lot of reasons that are well connected to other important things that we understand to be true, that you can't do that. Part of the reason is that in order to check whether a piece of software is going to produce an output or not, whether it will hold. So it will stop on an output, that is the output. Part of the reason that we can't know that is that we have to compute something about the program, or we have to compute the property of the program. But it turns out that you can't use computers to compute general-purpose properties of computers.

Suzanne: It's kind of a recursion problem.

Jonathan: Yes.

Suzanne: Right? And a little bit of Schrodinger's cat. It is like you are trying to observe something, and yet in observing it, you are changing it, and so you can't be confident that what you are observing is actually what you are observing. So I get that, and then that then means that I can't trust am I going to halt, and if I do halt, is that going to open up a vulnerability space?

Jonathan: Right.

Suzanne: So it sounds like that is really where that particular problem fits in trying to help us understand these undiscovered vulnerabilities. So if we can't do computational analysis of those kinds of computers, what do we do?

Jonathan: We can ask smaller questions. Instead of asking *Will this program be secure or not,* we can ask, *Are there any memory-safety vulnerabilities in the way that this program handles pointers*? That doesn't say that there are no vulnerabilities in the software, but we might be able to say that it doesn't have buffer overflows in the way that it handles pointers. You can write a bunch of different smaller security policies, which are super useful. We definitely should be doing that. It doesn't mean that things are hopeless, but it also means that if you, say, try to enumerate all of the ways that the computer could cause the vulnerability by making a mistake somewhere or that the software engineer has made a mistake or whatever, you will never enumerate all of the ways that mistakes could be made.

Suzanne: Got it.

Jonathan: So we can use program verification to check for stuff. We do that. It is important. We should keep doing it. We can randomly check software for mishandling of input. This is called fuzzing. We should also do that. But because there are, generally speaking, infinitely many possible PDF files you can produce for a PDF reader, you can't check all of the PDF files, because it would take literally forever. You can't be sure that the PDF reader doesn't have any problems, because you can check for some specific problems, like does it handle pointers appropriately to prevent memory corruption. And you can check some subset of files with the fuzzer, and the fuzzers can be pretty smart about where they are messing with stuff based on statistics of past vulnerabilities to check for similar problems, but you will never check enough stuff or enough properties to be sure that your program doesn't have any vulnerabilities.

Suzanne: To a certain extent, and one of the messages I took away was, especially when it comes to this dense set of vulnerabilities—some of it is architectural, some of it is programming practice. There are some things we can do that allow us to use these smaller questions productively, and there are things we can do that make it hard to use those questions productively. So we want to favor the architectural principles, the programming practices that are going to allow us to use these smaller questions so that we can make the space more sparse is I guess the way I would say it. Is that fair?

Jonathan: You bet. I think that that is a really common framing, and I think that that is actually the thing I'm trying to push back against. So I am glad that you framed it that way. If you cut a dense space in half, is it still dense?

Suzanne: OK. Yes, based on your definition. Yes.

Jonathan: OK. So, if you cut it in half again, is it still dense?

Suzanne: OK, yes.

Jonathan: How many numbers are in the dense space?

Suzanne: Oh, no. OK. All right, I get it.

Jonathan: Can you cut a dense space in half enough times that it becomes sparse? No.

Suzanne: No, unfortunately, not, but I can eliminate some of the…Can I eliminate some of the sources of a vulnerability that would make that a dense space?

Jonathan: Yes. There are two responses here. One is, there are going to be vuls. Have a patching procedure that is pretty good. Don't ignore that. Be able to prioritize the vuls that do get found, that are found. That is one separate side of things. The other side of the question you asked, *What do we do actually? What do we pay attention to?* I think that becomes an economics problem.

Suzanne: OK.

Jonathan: You are trying to make it more expensive for the attacker to find vuls. You should do that. You shouldn't make it free for them. If there is some open-source software that provides some sort of program verification or fuzzing technique, you should assume the attacker will use it and find any vuls that you don't find by not using it. If there is some open-source program-verification or fuzzing tool, you should use it. If you don't, you are making it pretty cheap for the attacker to find bugs in your software. If everyone does that, the cost, the value to an attacker of a vulnerability will go up, because there will be less of them. This is the basic supply-and-demand economics. If supply goes down, and demand is constant because the same number of people want to break into things, the cost will go up.

Suzanne: The other side of that economic framework though is that it costs me time, mostly time, in performing iteratively all of those kinds of analyses, whether it is a vulnerability, whether it is fuzzing, or any other framework that I am using. So that is the balance you are trying to strike is, how much it is costing me versus how much it is costing them, but then there is back to OCTAVE, you mentioned at the very beginning, which is a framework, risk framework that talks about low-probability, but very high-impact kinds of risks. In some systems that we work with, it doesn't really matter how much time I have. I need to spend enough time to really reduce the impact of what could happen in terms of vulnerabilities, not just the probability. That is where I think some of the attention to this really needs to come into play, because it is

easy for us to think about just lowering the probability, but what you are really trying to do is lower the impact to the systems that are critical. Yes?

Jonathan: Yes, and for critical systems, what this means is that for every critical software system that is written on a regular chip—I can talk about that qualifier later. So for every critical system that is written on and runs on a regular chip in a regular operating system, there are vulnerabilities we don't know about. If that is the huge problem, there should be lots of things that are not based on computers that are stopping that bad thing from happening.

Suzanne: Got it. So part of the point of all this is we get so centered on the computing analysis, the computational analysis, that we forget about physical security. We forget about some of the other things that are necessary, insider threat, that are necessary to use beyond the computational because we know that the computational analysis is flawed.

Jonathan: Yes, and I think that several of the communities where this is a real big problem are cognizant of this. I know that the people that write software for nuclear power reactors have a bunch of things that are not based on computers that will stop the reactor from going critical. Yes, I am very happy that they put all the work into that. It sounds like it is a very difficult job to maintain the software in that environment, because there are so many things that they have to go through. I am very happy that those people are doing that, and I hope that they continue. Some of us work in less controlled environments that are becoming software-based. I don't know that everyone understands the risk that they are taking on.

Suzanne: Fair. If I were to take an example of healthcare, the risk of releasing medical records may not seem like much, but for individuals, it can be very impacting, and even for groups, being able to have vulnerabilities that allow people to analyze the health trends of a particular population can be harmful as well. That is not a natural place that you think being a high-risk computing environment. Yet it is, when you start thinking about it in terms of what is the impact of having this data available to people that do not have our best interests at heart.

Jonathan: Yes. For sure. There are some just overlapping computer controls—not having things be directly connected to the Internet, that helps, but it doesn't stop everything. Different layers of authentication help, but also don't stop everything because both the network control and the authentication are also computers and also might have problems. You mentioned insider threat. Obviously, that is also potentially a way to bypass those things. We saw the executive order last year in May about critical software. One of the things that did was say that critical software is going to have to have vulnerability management. That is in line with this. Some people didn't have vulnerability management, even when they were doing critical software. I think everyone should be doing vulnerability management, even if it's not critical software, but you talked about the cost to the software developer and the cost to the victim of the attack, the cost to the attacker.

I guess the victim of the attack is the other entity in this economic system. One of the difficulties in balancing this out is that the software developer doesn't often actually bear the cost of their software being exploited. It falls on the user of their software.

Suzanne: Right.

Jonathan: One question is, if you are buying something, and you know it is going to have some flaws in it you don't know about, because it's a computer, is your contract structured in a way that you can get fixes for the lifetime of the product. And, is the person you are buying it from committing to providing those in a timely way? There are parts of the SEI that talk about acquisitions, which I am not super connected to, but these things are filtering up to that level.

Suzanne: Yes, it is a concept called software bill of materials, which is basically just trying to, at the start, just trying to identify what software do I have that is critical to me being able to continue to provide the capability? Then the next question is, *How long am I going to have access to it, fixes etc.*? I resonate very, very strongly with that perspective.

Jonathan: In some sense this paper is saying nothing revolutionary. The computing theory that I am talking about is from the 1930s. Some of the regulations and rules and best practices are catching up to everyone needing vulnerability management. They might not realize how deeply that is a requirement. It is not because we have done something wrong somewhere in the software engineering pipeline that you need vulnerability management. It is because we are using computers. A lot of people have recognized this. We are moving towards a world where more people are doing the things that are recommended by the conclusions already. People recognize this, people aren't dumb about this. It is not some entirely new thing that I have landed here. But I wanted to remind folks that the vulnerability-management stuff is here to stay as long as we are using computers.

Suzanne: I think that part of that message is that with the critical-software executive order, be mindful of whether you actually fit into that. Because one, you have got a compliance issue if you fit into the definition and you are not paying attention to vulnerability. Even if you don't, what I am hearing from you is, take vulnerability-management seriously, all the way down to the individual level, back to the *Is the password on your router still 1, 2, 3, 4?* because you could have impact. Me sitting in my home office can have impact if I am not paying attention to these things. I use a computer; therefore, I am subject to vulnerabilities. We all are. That is not something that, certainly my generation. I am in my 60s, that is not how I grew up. That may be how somebody in their 20s grew up, but a big part of our population is not accustomed to thinking this way. A lot of our viewers are going to appreciate not being able to sleep tonight.

Jonathan: Well, there are lots of places that can help with setting up vulnerability-management stuff. Whatever country you are in probably has a national computer security incident response team that you can talk to and get information about stuff in your country from them. The CERT/CC and SEI have a bunch of materials on doing coordinated vulnerability disclosure and all of that stuff. So again, it's not hopeless, it's one of these like *I have good news and bad news. You have to do a thing that you're not doing, bad news. Good news, other people have done it, and we have some advice*. But I know, so the software engineering folks, I think, it is not totally integrated. The change management is not totally integrated with the whole process all the time. Is that right? There is still work to be done there I think.

Suzanne: Oh, yes I know of cases where we have wonderful DevSecOps pipelines that have all of these static analyzers, and even sometimes in dynamic ones, and the first thing that happened is the software engineers turn some of them off because they are inconvenient. We still have some education to do.

Jonathan: Yes, and we have some usability stuff. I mean usable security often focuses on end users of software, but development pipelines and usability of program verification and fuzzing tools for engineers is, I think, an important area that some people have started to work in. Like you said, a lot of people turn it off because it is not convenient. That means that the development pipeline integration with the tools that do help can be improved.

Suzanne: I think you would get a lot of software engineers to agree with that statement. All right you have already talked about some of the transition aspects of this. Use your CERT, your CSIRT, kinds of resources and as a way of getting more visibility into this, I know we have vulnerability management sections on our website that have educational materials and things like that. As always, we will have some resources in the transcript that relate to this topic in addition to the paper and all the citations that it gives. This is in some ways a public-service message from you that is reflective of your research, but tell me, what is next for you in terms of where either this work is heading, or other work that you are looking at as being future work that needs to be done?

Jonathan: Yes, related to usability and related to the fact that there will always be more vulnerabilities, we are working on better prioritization of vulnerabilities once they are discovered. I would like to see everyone resourced sufficiently that they can just fix everything the day they learn about it, but that is not what the world we live in is like.

Suzanne: It is not practical.

Jonathan: Well, it is not economical at least.

Suzanne: Fair enough.

Jonathan: I don't know whether it's practical or not, but no one is going to pay for it, or no one is currently paying for it. So we need to be able to prioritize stuff. I work with the common vulnerability scoring system to try to make that standard better. I also have some strong feelings that it should be better in some very big ways, so we have been working on a stakeholder-specific vulnerability categorization, that is SVC, for the last three years, that provides a more consistent alternative that is a little bit more focused on providing communication between leadership and analysts of vulnerabilities. Sorry, providing communication might be wrong. Ensuring that they're both talking about the same things and making the same decisions.

Suzanne: So looking for alignment.

Jonathan: Yes.

Suzanne: OK, that's fair.

Jonathan: I get asked about vulnerabilities in machine-learning systems and stuff like that. The answers are pretty much all the same, and we're not doing the best job with commodity systems, so I think that we have a long row to hoe on smoothing a lot of that out. The bug bounties stuff, I think is interesting. I am not doing as much work on that right now. Bug bounties are an important facet of getting vulnerability reports in. Folks should probably have their own internal systems before they start paying people to run a fuzzer on their software and report a bunch of problems. That is a pretty useful marketplace solution for getting reports in, but you have to have a change-management function that is working before you can start taking in reports.

Suzanne: OK, so this just came into my head. It may not be something that that you have been studying, but the bug bounty made me think about it. Are there some gamification strategies for getting at least part of the community to engage more actively in this aspect of their work? Is that something that you are aware of?

Jonathan: I haven't thought about that too much, except for in the context of we are literally paying people to do the thing. That is not gamified in the sense that you mean, although it is using the results of some economics and game theory to get people to report to the people who make the software rather than using it in attacks.

Suzanne: Maybe that's your next LENS project. Everybody needs a new LENS project, right?

Jonathan: There's already some fairly good work on this that happens at WEIS, the Workshop on the Economics of Information Security. My effort at understanding how to gamify vulnerability management will probably be to read the forthcoming WEIS papers in two months, and hope that there is one about that.

Suzanne: That way, you don't have to write the LENS project on it.

Jonathan: Yes.

Suzanne: Jono, I want to thank you for talking with us today. I always enjoy expanding my brain in directions I hadn't been thinking about. I hope that we were able to do that for some of our audience as well. I want to remind our audience that our podcasts are available on SoundCloud, Stitcher, Apple Podcasts, Google Podcasts, and of course, the SEI's YouTube channel, my favorite. If you like what you hear and see today, feel free to give us a thumbs up. We always love that. I want to thank you again for joining us, and I want to thank our viewers as well.

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 don't hesitate to email us at info@sei.cmu.edu. Thank you.