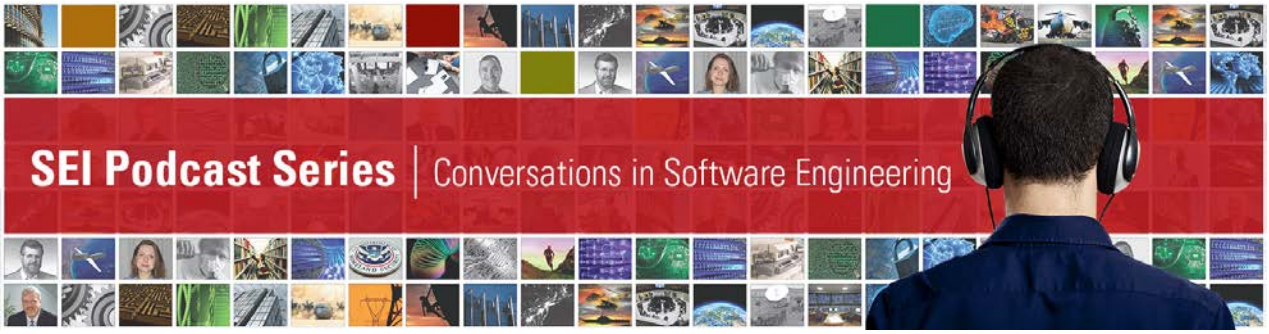

Data Driven Software Assurance

featuring Mike Konrad and Art Manion as Interviewed by Suzanne Miller

---

Suzanne Miller: Software vulnerabilities are difficult to eradicate and often have devastating consequences. While coding-related vulnerabilities are preventable and detectable, until recently little attention has been paid to vulnerabilities arising from flaws in requirements and design defects. In today's podcast we will talk to two researchers who are exploring the impact of design-related vulnerabilities and their impact on system costs and quality. Welcome to the SEI Podcast Series, a production of the Carnegie Mellon Software Engineering Institute.

The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense (DoD) and operated by Carnegie Mellon University. A transcript of today's podcast is posted 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 Mike Konrad and Art Manion. Before we delve into their research into data-driven software assurance, let's first tell you a little bit about our guests.

Mike Konrad is a principle researcher who has been with the Software Engineering Institute since 1988. I have known him since 1992. Up until 2013, he was involved with Software CMM and CMMI models development. Since then, he has been the technical leader for three research efforts that investigate the early software development life cycle, particularly requirements and design, in challenging environments. The first is eliciting unstated requirements at scale. Also data-driven software assurance, which we are here to talk about today, as well as concurrent deliberation of requirements and analysis of socio technical ecosystem infrastructure improvement. We'll save that one for another time.

Art Manion is a senior member of the vulnerability analysis team in the CERT Division of the SEI. Since joining CERT in 2001, Manion has studied vulnerabilities, coordinated disclosure efforts, and published advisories, alerts, and vulnerability notes for CERT's Coordination Center and U.S. CERT.

Manion currently focuses on vulnerability discovery and other areas of applied research, including ways to automate and improve operational vulnerability response. Prior to joining the SEI, he was the director of network infrastructure at Juniata College. Welcome, Mike and Art.

Art Manion: Thank you, Suzanne. Great to be here.

Suzanne: In 2014, the IEEE Computer Society Center for Secure Design was established. That is a new entity for them, to "shift some of the focus in security from finding bugs to identifying common design flaws," all in the hopes that software architects can learn from others' mistakes. I would like to start by having you explain for our listeners why this is such a big deal for the IEEE. What is the current state of practice, and why this sudden emphasis or seemingly sudden emphasis on design?

Mike Konrad: I would start by pointing out that there is a lot of code-centric tools and emphasis, and this has been very good in many ways. What it fails to do is acknowledge adequately the bigger picture, the longer-term costs and impacts of decisions made in the early software development lifecycle, of the cost to patch, and damage to or harm to a mission that comes operationally once that software is delivered and deployed in the end-user or operational environment. What we had hoped to do with this project was take this broader view not just of the code-centric practices but the context that led to the development of software, taking into account the impacts, operationally, that vulnerabilities might have.

Suzanne: So, we talk about the *-ilities* or we talk about quality attributes, security, obviously, being one of them. And, vulnerabilities are things that are malicious or not malicious that lead to, frequently, security as well as other kinds of operational breeches.

Mike: That is right.

Suzanne: So, this is very consistent with other research at the SEI, research in architecture, research in static analysis, other kinds of techniques for addressing not just security but the other kinds of quality attributes. What is special? What is unique about dealing with vulnerabilities in that requirements and design space?

Mike: A subset of the errors that are committed during software development can leave the resulting software product, depending in part on its end use and other matters, prone to a vulnerability that may result in a malicious attack, loss of information, and, as I mentioned earlier, potential danger to the mission or to those engaged in the mission.

So, it is a subset of the human errors that are created. It is difficult to know in advance and it depends on the use case, the operational environment, which errors may actually become vulnerabilities that release information or control even of the end product or its system to some

outside party. It could also lie dormant for a period of time, and the error may not be, or defect might not be recognized. At some later date, the operational conditions could change, the use case may change, and what was a benign, relatively benign, defect suddenly becomes something that can be exploited.

Suzanne: So, in contrast to, say, user experience, UX-kinds of defects which may be annoyances, these are defects that, as you say, can be prone to lay dormant and can have pretty devastating effects in terms of making public information that shouldn't be public, other kinds of mission kinds of impacts.

Mike: Exactly. [Defects] can give a malicious individual, intending individual, access to information that the program is storing or perhaps even to the operating system underneath and access to files or control of the system, which is perhaps the ultimate goal. As we know today from lots of news headlines, there has also been a lot of value attached to extracting information: credit card numbers, private information.

Suzanne: Email addresses.

Mike: Even emails, yes.

Suzanne: How is your research trying to address this? You talk about getting into the design and requirements aspects of vulnerabilities, how did you approach that? What does your research help designers deal with?

Mike: Well, I think it is important here to acknowledge that this project was really two different groups at the SEI, two different research groups broadly speaking at the SEI. One that has been associated more with project management, process, and measurement, and the other, which is much more related to software vulnerabilities and their detection and research around how to mitigate them and alert the community and guide the community in dealing with them. Art represents the latter. I represent the former.

This was, for us, a bit of an experiment as far as typical SEI projects go, in that we tried to draw from expertise from two different parts of the SEI organization. I have to say, it's been not only a pleasure working with Art Manion and Julia Mullaney and others, I learned a lot from those folks. They have insight into what happens after the requirements and design. My interest has been requirements and design. Art, maybe you'd like to speak to that.

Art: Thank you for the kind words there, Mike. To your point, it was a very nice collaboration in that sense. If you look at a software development lifecycle [SDL], my area and my team are dealing with bugs that are now security bugs. These vulnerabilities, they allow the consequences Mike was talking about. We are dealing with after it has been baked in, shipped, deployed, used,

discovered, and now we are playing a clean-up game. Then, Mike's area is looking at coming at it from an earlier part of the SDL where, *Can we do things at design architecture time that maybe lower the number of these things or reduce the harm of them that get out the door?*

Suzanne: So, you have got sort of the catalog of things that we see in operations, things that we see in test. Mike is really saying, *OK, out of these things I can identify that this set over here are things that might be from a requirements flaw or might be from a design flaw*. Is that how that collaboration worked?

Art: Yes. That is generally how it started out. One of our starting points was to look at our catalog of operational vulnerabilities, things we had found. It was largely a bit of manual inspection. At least in my world, we look at things—we typically say an implementation flaw, so, a coding mistake—and that can be fixed by hopefully secure coding practices or the coding tools Mike was talking about might detect those, code-focused tools. So, those were fairly easy to identify out of a set, and we looked at what was left. With a little bit of manual reading into the case and remembering what happened at that point three years ago, we were able to pretty well determine that those were likely to be a design-time issue or architecture-time issue and not just a mistake in writing a line of code somewhere.

Suzanne: And, those typically are going to have farther reaching effects than where they are found also. Right? Because when I have an element of the architecture—messaging, for example--is going to affect a whole range of components in a system. So, if I have an architecture defect that I found in the messaging component in one place, chances are I am going to have to look everywhere else to see, *Is that same flaw something that has been seen in the other components?*

The effect of these architecture and requirements flaws can be much more far reaching than often the, *I turned a plus into a minus when I am doing my coding*. How did you deal with that piece of it? How did you deal with figuring out where to look to see the effects of these design flaws?

Mike: So, that is kind of getting to the design of the project. Initially, our thought was that if we looked at a variety of root cause analysis type techniques, we might be able to find one that we could use for some of these design-related flaws that Art was just speaking to.

We looked at a couple different techniques. Our colleague, Bill Nichols, led that particular effort. We looked at IBM's Orthogonal Defect Classification. We looked at HP [Hewlett Packard]. They looked somewhat at different parts of the life cycle, and we tried to pick from the best of those for our particular purposes. That did lead to a kind of taxonomy that we then used in reviewing the history and provenance of particular design-related vulnerabilities to understand how they came to be, maybe what is the nature of the decision making that led to them.

I should quickly mention here that we mean *design* in an expansive sense here. It is often in the broader context of the particular context in which the product was envisioned, that architecture was envisioned, that resulted in an oversight or maybe more narrower focus than was warranted, considering the long-term reuse and particular context envisioned for a particular component changes over time.

Suzanne: Sure. So, lack of consideration of kind of the evolution of the product can be one of the sources of these kinds of vulnerabilities.

Mike: Well said. A second component of the project design here was we did a mapping study. Again, Bill Nichols led in that. So, the question we wanted to answer is, *What is already known about what practices might lead to design vulnerabilities? Can we even predict design vulnerabilities?*

There we found the research literature to be actually, frankly, kind of sparse. It had some useful material, but for the most part what was focused on was more predicting faults or errors, defects in code, some of which, as we mentioned, might become vulnerabilities and trying to predict, maybe, which modules would have them. We concluded that that really wasn't quite so adequate for our purposes.

Then we began what we call these deep dives, and we kind of teamed up and paired off, and we looked at some of these vulnerabilities that Art mentioned earlier. Trying to understand, again, under what was the particular flaw? What decisions led to that flaw being there? They were often conscious. They were often decisions that were made, occasionally, even out of spite, perhaps at a competitor. It was intentionally, maybe, thought of as a benign decision initially, but as the product got more and more widely used it also planted the seeds for much great unintended impact. So, we tried to follow the timeline for these. We learned that some very specific vulnerabilities did that. Then our hope was to also capitalize [on] a model—from Bill Nichols and my part of the SEI, which is today called Software Solutions Division—and, we hoped to capitalize on data with the Team Software Process or TSP and use that to inform a model, an economic model [whose development was led by Andy Moore], that would relate certain decisions for how software was developed to potential economic impact and operations.

So, that is kind of the broad sweep of the project. We tried to do a little bit in every area and ideally everything would lockstep with each other. It didn't quite [work] as well as we ambitiously had hoped, but truth is we got some, I thought, some interesting results.

First, the sense that vulnerabilities—both design-related vulnerabilities and requirements-related vulnerabilities—can be horrific in their impact. In some cases, you literally cannot remove the vulnerability. You have to live with it.

That was one of the things I learned and did not know and [learned] from Art and Michael Orlando, who was another project team member at the time.

One of the others was that, under some very modest assumptions, and this speaks to the economic model, there is a win-win-win story in here. A little more attention to design practices and identifying early design defects makes developers happier because they can get to code release faster; makes the customer happier because there is less disruption from faults as well as our concern, vulnerabilities, in the field; and the software is more ready and available and operational. So, we saw a win-win-win here also for the acquirer, who has to worry about the long-term costs.

Suzanne: And sustainment.

Mike: And sustainment. Right. Thank you. Of this [software] in the field. So, we saw a win-win-win from some modest assumptions of the economic model. But, to use that, unfortunately, you would have to tailor the economic model and calibrate it for your organizational history.

Suzanne: An interesting aspect of this for me is that it is sort of like "Back to the Future" because the SEI has been preaching, *Pay attention to design*. *Pay attention to requirements*. *Pay attention to these things early and in multiple dimensions*. *Don't just look at security. Don't just look at usability. Look across these. Find out where the conflicts are, deal with those early on.* We have been talking about this for 20 years. What do you guys think are some of the barriers to actually getting people to adopt practices like this that would have huge benefit but require them to change the way they approach design or require them to look at requirements in a different way? Do you have any sense of that? Or, have I just gone over the border there?

Art: Well, the software that I see most often would probably be called COTs [commercial off-the-shelf software]. It is internet-facing or widely used.

Suzanne: Commercial off-the-shelf software.

Art: Sure. Commercial off-the-shelf software. The point, there, is there is custom software development or more bespoke development, which may have a different lifespan and a different process around it. But, the speed of the internet is, *A new version. A new technology came out. A new way to share your information came out.*

Suzanne: *A new competitor.*

Art: *A new competitor came out. Time to market*. Those competitive forces and the compressed time…My guess, and this is just an educated guess here, is that the SDL time is squeezed, and it is probably fairly easy to grab some components quickly and assemble them and ship something without doing what we are talking about, some threat modeling, architecture review, considering

design carefully up front. Even though that may pay off in the long run, there is just not time to do it. Developers feel that at least.

Suzanne: So, there is what I would call implicit acceptance of risk instead of explicit addressing of risk.

Art: Also that. A lot of the privacy work, you will see people say, *My information should be private*. Yet, in practice they will participate in all sorts of information-sharing activities. *Free stuff? Yes, I will give you my information*. Again, all of that compression of time and acceptance of possible risk, I think that is probably driving the lack of attention up front in the SDL.

Mike: Excellent. If I could build on that, I think Art has touched on what, for me, are two reasons or two phenomenon. Art's point was that a lot of software development today is developed to different business values. As you said, Suzanne, maybe to a diminished perception of risk associated with quality deficiencies.

Suzanne: *These components have worked over here, why shouldn't they work for me?*

Mike: Exactly. The second thing is that I think a lot more software development today does a lot of COTS-type integration, reuse, calibrating software that is already there. We end up with these very layered stacks. So, I think compared to 30 years ago, there is simply a lot more layers of software for what we develop.

Suzanne: And, less knowledge about the internals of the software that we are using in composing these systems.

Mike: We were also dealing with a lot more products and services and infrastructures that didn't exist 30 years ago where, to some degree, there is a little bit less fragility around, say, avionics-type software.

Suzanne: As Art said, the custom software has a lot more scrutiny, it has a whole lifecycle of assurance kinds-of-activities that you don't know.

Mike: Well said. Yes.

Suzanne: They might have been done in the COTS product, but you don't know that.

Mike: Right. I think software has become a spectrum of different worlds or universes where there is still heavy, and it has to be heavy, attention to the quality of software. On the other hand, I'm calibrating a website that has security built in or building on a platform such as IOS that has security built in to a large degree, and so I don't have to worry so much about that. The worst

thing I will do is just anger or inconvenience my user. I am inheriting the value that has been built in the infrastructure, assuming that it has been done correctly.

Suzanne: That's the big assumption. How can organizations in the commercial space as well as in the more custom space use the research that you have done to help them create products with fewer vulnerabilities? What are a couple of practical things that they could use if they wanted to?

Art: Mike had mentioned an economic model that is in that work, and without delving deeply into it—and this is also, this is long standing SEI research—that investing design and development time up front earlier in the cycle pays off. Fewer bugs, fewer defects, cheaper to fix. More robust software.

The economic model was an attempt to, sort of, find some of the pieces to that argument or that equation, or see if you could turn up a variable that implied more design time, and what does that do to your defect rate. That sort of thing is behind the model. One thing a developing organization could do would be something similar. Try to assess, *If we shift our investment in the SDL to earlier, is that a win for us? Is it feasible in our market, and does it result in fewer defects, less-harmful defects, more robust software?*

Something that, at least in my area—again, it is sort of an educated guess, but we think [it] is perhaps another quick way a software developer could get into some of these things—is threat modeling. It is one thing to say, *I'm going to apply the security broadly at the design time*. If you are building software you know is going to exchange messages on the internet or be part of a web service or something, there are a handful of attacks you can very well predict. They are going [on] all the time in the background. So, at the very least, you identify where sensitive data is. What are the interfaces with the internet, the network, and their systems? What are common types for attacks that might happen? At the very least, enumerate those and figure out how your design, you know, deals with those types of events.

Suzanne: I will tell you, that is an area that I was surprised…I have done some work evaluating technical teams. I was on one of these evaluations about a year and a half ago. That whole concept of threat modeling…. They had a threat model that was an enterprise-level threat model, but they were kind of off to the side doing something a little different, and they didn't see any need for having their own threat model. That really surprised me. I mean I was like, *Really? You guys aren't doing exactly what the rest of them are doing. Don't you want to think about what your threats are?* So, that seems to be an area that is not as well understood or well accepted in the software architecture and design area as some other practices might be. I second that.

Mike: I third that.

Art: When I look at threat modeling, or attempted to look into it more formally, I've seen NASA work and the safety software world is paying attention and does hazard analysis and things. Some stripped-down version of that I think would be applicable to internet software and COTS-speed software, maybe not very much thorough analysis but parts of it should apply. A lot of the mistakes we see coming out just seem to be, *We didn't think about this. It wasn't even considered, and now we have got a mess.*

Mike: Unfortunately, I think that the trend that you two have been both speaking to is going to continue. Things are going to become more interconnected. These two worlds that one might have thought of as separate and amenable to different concepts of what best practices are, perhaps, and what techniques to employ, I am not so sure that that holds up as we increasingly intermix the artifacts and components from both worlds and become dependent on both.

Suzanne: There are architectural patterns, service-oriented architectures make assumptions about interconnection and things like that that, if they are held, if those assumptions hold, everything is great. But, if they don't hold, if the COTS infrastructure didn't pay attention to some of the vulnerability possibilities and things like that, your service is interacting with something unsafe. That aspect of it is another architectural aspect that architects just can't look at their own services; they have to look across the whole span of the system, or the systems that they are interacting with, to be more assured of safety. We are never going to be completely safe.

Mike: I hold the view that I have heard others speak to, of there being an attention economy. In a certain sense, the amount of context we need to keep up with as we try to develop code I believe used to be much more narrow and reduced. I don't know that our brains are built well to handle lots of possible targets, and also moving to a situation where you have a weakest-link-in-the-chain-type phenomenon going on.

I think one good thing about our project that might be a takeaway is to try to make more visible, more salient, more compelling, this overall sense or measure of what the value [risk] of the weakest link is and help encourage attention.

Suzanne: And, the impact of the weakest link.

Mike: Exactly, and drive attention to that. So, that is the potential value of things like enterprise models, looking from a very broad picture of where is our weakest link, are there particular parts of the organization that are not focused on this broader picture that we need to bring in, such as the product development group.

Suzanne: Excellent. Well, I am excited about this research. I think this is an area that connects very well with other work that we have done at the SEI but gives a different spin and focus on it that I think is very timely for the software industry.

I thank you both for joining us today. For a deeper dive into the work that Mike and Art and their colleagues have been doing, we welcome you to visit the SEI library where you can download a copy of their technical report by going to resources.sei.cmu.edu/library. In the keyword field, type the name of their report, which is Data Driven Software Assurance. Or, you can search on Art Manion and Mike Konrad. Although if you search on Mike Konrad, you will find lots of other stuff too, interesting but not all about data assurance.

This podcast is available on the SEI website at sei.cmu.edu/podcasts and on Carnegie Mellon University's iTunes U site. As always, if you have any questions, please don't hesitate to email us at info@sei.cmu.edu. Thank you.