Silver Bullet Talks with Jacob West

Gary McGraw | Cigital

Way back in episode 78 of this show, we discussed the arc of your career from intern to pundit, building static analysis tools, and the BSIMM. I think BSIMM4 had just come out, and now we’re on BSIMM6. This time, I’m interested in discussing the CSD work you just published. What is the CSD, and why is it important?

We founded the CSD a couple years ago to shift the focus in security from what we think the main focus has been, which is finding and fixing bugs, to looking for and actively avoiding design flaws that lead to very serious security problems. Instead of just focusing on the implementation, we’re thinking about the security implications of the design from a project’s inception to its completion.

Can you give examples of a bug and a flaw so people can understand the difference?

An example of a bug is something like cross-site scripting. Let’s say developers are trying to build a webpage and want to generate content for their users. What they’re not thinking about is an attacker supplying some malicious value. Because they don’t scrub that value, they don’t validate it before they output it in the page—that’s a bug, a mistake they made in their code. The attacker can deliver not only characters the way the programmer might have expected but also script that might run to the user’s browser and execute an attack. No one intended for the website to have that feature, but the programmers made a mistake when building the site that allows that attack to succeed. That’s a traditional security vulnerability or bug.

A flaw is a design decision—something the system intended to do but that probably isn’t a good idea from a security standpoint. One example would be thinking about authentication mechanisms and how we authenticate a user. A system might be designed to allow a simple login and password, and it might be designed to allow any arbitrary password—even very insecure ones like a dictionary word. That wasn’t a mistake a programmer made when building the site; it was part of the design. It’s a requirement that was missing from the design, which is that the password scheme requires strong passwords.

You gave a great example of cross-site scripting as a bug, but if you think about the APIs that programmers are using to get input and the frameworks they’re using, there might be ways to solve that entire class of bugs with a design tweak. In fact, Google has done that. This is one of my favorite interview questions: asking people to decide...
About Jacob West

Jacob West is the chief architect for security products at NetSuite, where he leads research and development for technology to identify and mitigate security threats. He has also served as chief technology officer for Enterprise Security Products at HP, where he founded and led HP Security Research. West coauthored the book Secure Programming with Static Analysis with Brian Chess in 2007, which is the only comprehensive guide to using static analysis to avoid the most prevalent and dangerous vulnerabilities in code. West is a founding member of the IEEE Center for Secure Design and the Application Security Advisory Council, and is a coauthor of the Building Security In Maturity Model. He is a frequent keynote speaker at industry events worldwide.

Tell me more about the CSD.

We pulled experts in software security from three main sources. The first was folks like me in the commercial world—many of us have built security products in the past or oversee security practices for our employers. Another group was from academia—folks who are responsible for teaching some of the security implications of the architecture and design principles that the CSD is so concerned about. The third was folks from government who probably have a different view of some of these systems. They might have different considerations from a process standpoint about how systems are designed, or maybe who their attackers are.

We think bringing together these three diverse groups provided a very wide-ranging set of perspectives. We actually asked people to bring real data to the first meeting. The first document we published listed the top 10 software security design flaws. We built that document out of the raw data and experience the group brought together during our first meeting.

That was published in 2014, and you recently published another report with the CSD called WearFit: Security Design Analysis of a Wearable Fitness Tracker, which is not a real product.

But it brings the top 10 flaws to a real example so people can understand what a flaw might look like in an actual system.

In the initial top 10 flaws document, we give examples wherever we can of how the general guidance for avoiding the flaw applies to different kinds of systems. With this latest report, we inverted that equation and started with a real system design. As you mentioned, WearFit isn’t an actual company; we weren’t looking to pick on anyone in particular, but we looked at wearable fitness tracking devices across the industry and tried to understand how they’re designed with hardware and protocol constraints, implementation decisions, and so on.

We used that information to design a fictitious system that closely resembles real-world systems. Once we had that design, we took the top 10 flaws from the original CSD document and applied them systematically, one by one, to complete a 10-step design review of the WearFit system we created. We talk through what would have happened in that design review, the discussions that would have occurred, and the key parts of the system design that would have been reviewed, and we try to add color about why certain system designs would have been made to achieve certain security properties in the final system.

Who were your coauthors?

I was lucky enough to work with really good friends: Yoshi Kohno from the University of Washington; David Lindsay from Synopsis; and Joe Sechman, who was a colleague of mine at HP Enterprise.

What was the most difficult flaw to work with when thinking about the design?

I think we spent the most time talking about the interplay between privacy and cryptography. This relates to a few different flaws about what’s...
important and how you protect it. I think this was particularly interesting in the fitness tracker scenario because it’s significantly hardware constrained. You have to be able to encrypt data on a very small wearable device with not a lot of power or CPU.

The data’s sensitivity also isn’t exactly clear. If the data was credit card numbers, no one would argue that it wasn’t sensitive information. If it was ambient temperature readings, you can get that information anywhere; it’s public already. But what about steps and heart rate and the other information this device is collecting about a person? How sensitive is that? How should it be protected, and how should it be shared? These were topics we spent a lot of time discussing.

Let’s talk about the importance of the process by which you find flaws like this. I would imagine you guys did it in an ad hoc manner, but systematizing an approach to architecture risk analysis and threat modeling has been a big challenge. Documents about process can be off-putting to some people. In our report, we try to walk readers through the process without really talking about the process itself. Up front we talk about the system’s design in pretty technical detail. Then we talk about the different attack categories and the kinds of threats we think the system might face. We group those into the high-level buckets—things like denial of service, compromising the device’s integrity, stealing a user’s health data—and we enumerate examples of those potential attacks.

With the combination of that design and a very ad hoc threat model in terms of what we were concerned about, we were then able to proceed through the top 10 design flaws, thinking about how the design and the threats would interact with a system that was eventually implemented. This worked really well.

**Why is design review important, who should do it, how should they do it, and should it be a process?**

Everyone should do design review, meaning every organization that’s building software. In terms of who should conduct the design review, you have to know something about design, and you have to know something about security. Somebody with architecture chops, whether that’s their title or not, is pretty important, but they have to understand the security side of it. Most likely, you need someone with experience in software security specifically.

The vast majority of what we talk about in the report and the design decisions that we believe have security implications are related to non-security functionality. It’s not the crypto or the authentication mechanism necessarily; it’s about how the system moves data around and services its users. It’s not just about the security features; it’s about the security implications of the way the rest of the features were designed.

**Are there processes for doing design reviews that are more principled than “be really smart and have a lot of security experience”?**

I think one of the best ways to learn is to work with someone who has experience. Whether it’s someone you hire from a consulting firm or someone you know, or whether you build up that capability internally, the best way to learn is to go on a ride with somebody else and see how they do it.

You mentioned the possibility of figuring out how to do architecture analysis inside your own organization. Do you think that’s possible, or do you really need to find somebody who’s done this for a while and knows how to look at these problems? Have you had any experience trying to create this capability in an organization?

I have, but it has always involved very exceptional individuals. I don’t think it’s impossible for a firm to do this on its own; it needs to be able to get the right people, which is a challenge. It’s hard to hire a good architect even independent of security. It’s not something that I think is 100 percent reproducible, which is one of the reasons we’ll continue to see outside firms provide a lot of help in this area.

One of my pet peeves, which is something I speak publicly about as often as I can, is the challenge we have in finding new security people. We have a huge talent shortage in security. Firms can’t hire enough people to solve their problems. And that doesn’t even account for tackling areas that aren’t really being looked at today, like design reviews. At the same time, the top universities and best computer science programs in the country are still doing very little to instruct undergraduates in software security and secure coding.

The industry is going to have to do everything we can, kind of guerilla style, to create these skills. But I think we should also put pressure on universities so they can start to meet this demand as well.

**What are your thoughts about the evolution of software security as a discipline since your time as a**
student at the University of California, Berkeley?
I think it hasn’t evolved nearly enough, frankly. I graduated from Berkeley in 2004 and, at the time, had really no exposure through coursework to any security topic—certainly nothing to do with robust, secure programming skills or secure design principles.

Unfortunately, more than a decade later, we haven’t seen much change there. You can graduate from any one of the top universities in the US today with a degree in computer science and really never be exposed to software security topics. That’s not to say those classes aren’t available in many schools today, but you have to hunt them down.

We need to get to the point where security is treated as a fundamental property of software and, therefore, of computer science. It’s taught as part of every discipline we teach today: OSs, networks, data structures, databases. All of these have security implications, and we need to teach these subjects in the right way.

The challenge is, of course, that real-world architecture often differs from academic project architecture. There might be security implications, but you don’t get any real experience with walls of code like you might find in a financial institution, for example.

I think we’re always going to have a delta between someone leaving a degree program and going into the workforce. There’s always going to be a gap between academic scale and commercial scale. What we can do is start to inject more software security along the way, so when graduates get to that final step of scaling up to the real thing, they’ve at least seen the important building blocks of software security. Today, security is just missing from that equation.