Silver Bullet Talks with the IEEE Center for Secure Design

Gary McGraw | Cigital

Hear the full podcast at www.computer.org/silverbullet. Show links, notes, and an online discussion can be found at www.cigital.com/silverbullet.



aunched in August 2014, the IEEE Center for Secure Design (CSD) gathers software security expertise from industry, academia, and government to provide guidance on recognizing design flaws and building security in. Three of the founding members of the CSD discuss the project and its future. Jim DelGrosso is the executive director of the CSD and a principal consultant at Cigital; Yoshi Kohno is an associate professor of computer science at the University of Washington; and Christoph Kern

is an information security engineer at Google.

Where did the notion of the CSD come from, and why is it important?



end of 2013, you and Kathy Clark-Fisher of the IEEE Computer Society started kickaround ing ideas

about starting a cybersecurity initiative and started thinking about some of the problems that have been lingering around for years, or quite frankly, decades. These problems haven't been solved, so they must be pretty difficult. Is there something we can do to solve those problems? The Center for Secure Design was born with the idea of ways we can try to prevent software from being built with design flaws.

We make a distinction in the CSD between implementation bugs and design flaws, both of which are software defects. Can you explain the difference with some examples?

DelGrosso: One type of a security defect is incorrectly using a security control. Maybe you're using a cryptographic primitive, but you're using the wrong primitive for what you're trying to accomplish. Imagine you're trying to prevent the tampering of data, but you're using a confidentiality control. That's a design flaw. It's a broken design that isn't going to be caught by something like static analysis. And something like this is probably very difficult to find with dynamic analysis as well. You really need to look at the design of the application to recognize that you haven't chosen the right control for the right purpose.

Please contrast that with a bug that you might normally find in a software program.

DelGrosso: Keeping with the crypto universe example, an implementation bug might involve using an incorrect key size or the classic example of any kind of buffer overflow or SQL injection, where the way a developer wrote the code created a defect, but it's really an implementation bug. The developer's choice of how to write the code created the defect.

I have a really simple example: forgot to authenticate user. When you stare at the code, you're not going to find that flaw. In fact, you have to consider the architecture in order to understand that flaw. Do you have an example of a design flaw that you think would help people figure out what we're going after with the CSD?



Kohno: Often, a flaw has to do with a failure to realize the need for something or using the wrong component. Sometimes we

see systems where people didn't realize they needed encryption or cryptography or didn't realize the need for authentication. Another example is the unfortunate practice of some servers not storing hashed passwords, but just storing passwords in the cleartext. To me, that's an example of a design flaw, because of the failure to realize the need to protect those passwords.

Do you have a favorite example of a design flaw? Maybe one that's related to bugs, so we can get some distinction between bugs and flaws.



Kern: I might make this a little bit more complicated, because I think it's important to realize that many classes of bugs are

actually in some way manifestations of design-level issues. You can say, for instance, that a buffer overflow is primarily a bug, right? There's a specific line of code that's wrong, and you can often find it through dynamic testing or static analysis. But if you look at the problem of having this class of bugs in the bigger picture, it can happen all over the place in a large piece of software. What it comes down to is that it's actually a question of language design: if you use a memory-safe language, you won't have this class of bug, but if you use C, you will.

This goes for other classes of bugs as well. Cross-site scripting [XSS] bugs, for instance, are in one way very simple and straightforward. There's a specific piece of code you can point to where the developer forgot to appropriately validate, encode, or escape a string that is somehow placed into HTML markup context. But XSS is also a design-level issue. You're not going to get rid of all possible XSS bugs unless you look at the design of the APIs and template systems that you use to produce HTML markup.

I think it's important to understand that sometimes when you work on design, you can really help to address entire swaths of bugs. Why are flaws a challenge to find and, more important, to fix as opposed to bugs?

Kohno: Oftentimes flaws arise because of the world view of the people who are designing the system. The flaw manifests because the designers didn't think to ask the right questions. I've seen a number of systems where a flaw just comes out of left field. The designers didn't think about it, and throughout the whole process of creating the system, they just weren't looking in that spot in the right way.

Kern: Flaws are more transcendental in a way. For instance, flaws can arise from a misunderstanding of the trust boundaries that are inherent in the system. A designer might make the inappropriate assumption that the system can trust values it receives from a piece of code that actually runs in an untrustworthy environment. A typical example is if you have a client/server or Web application and the server that runs in your environment assumes that the values it receives from a client (that's actually under the control of a potential attacker) are trustworthy. You have to understand the trust model and the assumptions about the trust in the various components-including where they're going to be running. As far as I've seen, there is really no practical way of formally describing and capturing such a view, and then automatically reasoning about it.

Finding flaws is difficult. One of the reasons we formed this workshop

is that the secure design problem has been around for a long time and there aren't that many people who are experts at thinking about security at the design level. Good security design analysts are usually pretty good architects in the first place meaning they're rare. In general, that means finding flaws is a challenge, and a still-open problem is that we haven't been able to scale the activity very well. Can you tell us about the first CSD workshop?

DelGrosso: There were 13 attendees at the initial workshop, from various areas of commercial and private industry. We came from a lot of different backgrounds. The homework was to bring with you the most common design flaws you find in your working environment. We started with a lot of real flaws, and came up with the 10 most common design flaws. Then we started to document what they look like, how we can identify them, and what you should be thinking about when you're designing a new application (or even as you analyze one of your existing applications). We wanted people to avoid the design problems that we've seen or have made.

Can you explain why a list like this could be helpful in your work and in other peoples' work when they're thinking about design?

Kern: If you're a small development shop and you realize you need to worry about security, you might end up looking on the Web and finding a bunch of lists of bugs. And then your mindset becomes, "I need to worry about XSS and buffer overflows." So you get down to this nitty-gritty code-level detail, but you might end up overlooking a much more important (and harder to fix) design-level problem in your application. I think having this CSD list of flaws gives you a starting point to ask the right questions. It'll hopefully make people aware of the kinds of problems that can arise so they can seek advice from somebody who can help them.

How do you think the CSD top-10 flaws list could be helpful in an academic setting?

Kohno: Part of what the CSD did is to identify the big areas where design flaws can arise. It's a great checklist for a company, large or small, to actually go through and make sure you dot your Is and cross your Ts and think about all these things. In an educational context, we're producing students who will eventually be in the workforce. Maybe they'll be managers or the people that are implementing systems, but these are the issues that the CSD recognizes as being some of the top design flaws that all kinds of technical people need to consider. I think educating people earlier on these issues is very useful, to make sure the students are prepared to think proactively about potential design flaws when they enter the workforce.

I think it helps to counter some of the misconceptions about the idea of looking for bugs only and declaring something safe when you don't find any bugs. Kern, I wanted to give you an opportunity to dig a little bit more deeply into the XSS stuff you were talking about earlier. Explain how some of your work focuses on XSS at the design level.

Kern: In itself, XSS is a fairly simple bug. The problem is that in a large Web application, there's potential for many instances of this bug. In practice, it's insufficient to go after these bugs one by one through testing or using static analysis. You're always going to have some left, and you won't have a high degree of confidence in the absence of this class of bug. What we've been trying to do at Google is design safe wrapper APIs



for the Web platform and safe libraries for HTML rendering. By design, application code that uses these safe APIs can't have XSS bugs. This effectively confines the potential for bugs into a very small portion of the overall code base—the implementation of the safe DOM wrapper APIs and template systems. Ultimately, we can restructure our application so that we really do reduce the potential for bugs. More important, we get a much higher confidence in the absence of bugs, because confining their potential into a small portion of the code allows us to more effectively reason about them.

DelGrosso: One of the interesting things I think might be happening with some of the bugs we're seeing is that frameworks are providing capabilities to developers that allow them to get a security control almost for free. But you have these protections

built into the framework that allow you as a developer to avoid certain types of bugs by using the capabilities of the framework. That seems to be the kind of model we need to try to drive toward. That is, we make some of the security choices for developers easier so they don't (and can't) shoot themselves in the foot as often. This kind of move will solve many particular instances of bugs, because developers are simply going to code things the right way. We're going to help them code things the right way by making it really hard to code the wrong way.

Kern: The tricky bit, of course, is to design these APIs so that developers actually want to use them. They have to be easy to use and not drastically alter the development flow that developers in your organization are used to. You have to work within the existing development culture



In a document called *Avoiding the Top 10 Software Security Design Flaws*, the IEEE Computer Society Center for Secure Design (CSD) identified common design flaws that can lead to security problems to help software architects learn from others' mistakes. Visit

http://cybersecurity.ieee.org/center-for-secure-design.html to read the document, and contact Kathy Clark-Fisher at kclark-fisher@computer.org to get involved. —Eds.

> and the existing software development process and add on the things you want for security. I think that's what allows you to make this actually successful in practice. If you try to completely change the paradigm, people will resist.

Can you explain very briefly the process by which we took our list of flaws and flushed out each of the top 10?

Kohno: After grouping the flaws in different areas, we broke off into groups with multiple perspectives represented. For example, cryptography would have a couple of people involved who really look at and focus on cryptographic issues. We worked through actual examples of design flaws, how to best convey them, and how we would recommend that people mitigate this potential issue. We would iterate internally (within our group) all of the important points that need to be addressed for that flaw and then actually write out a section that covers it. Then the entire group of participants went through the whole document with multiple passes, trying to comment on each other's sections and clarify things. At the end of the day, this really is a joint document produced by all of us.

What are your plans for the future of the CSD?

DelGrosso: Over the next two years, we're going to continue to have workshops and enhance the level of detail that we go into for each of

these flaws. We want to create the idea of secure building blocks. We want to create usable information so that anybody in the world can come to the CSD site, get some information about flaws, and get actionable advice on what can they do to avoid them. The list will change over time, as-heaven forbid-we start to solve some of these problems. We hope to host actionable advice where people can look at our work and say, "I hadn't really thought about that aspect of security control. Let me think about that. Let me go design my system to avoid these flaws in the future."

Do you believe we can make as much progress—or as little, if you're a pessimist—on design flaws as we have over the past decade on automating the finding of software security bugs?

DelGrosso: I think we can make more progress going forward than we have made, because I'm in the camp where I don't think we've made that much progress at all. The same flaws that we've known about for 30-plus years continue to show up at an incredibly alarming rate. So I think we can only get better at this. I'm very optimistic moving forward, but I think history reflects very badly on us.

Kohno: That's a really good question and a really tough question. I'm going to remain optimistic thinking that we can make a lot of progress. But there are a number of challenges and a lot of different environments where diverse issues arise. I think that if we consider one type of environment and can really focus deeply on design issues in that environment, we'll be able to make a lot of progress. But I do want to bounce back and say there are a lot of very diverse environments to think about, from Google Glass and the desktop computer to the toaster oven and the automobile.

Kern: Talking about design issues in the abstract is very difficult, and it takes experience to actually map the result to a real, specific design. However, applications typically fall into some class or another. Web apps usually have a lot in common in their architecture, and there's a common threat model that goes with it. The same goes for a mobile app that talks to a network-based back end. Maybe the same is true for medical devices or for automotive devices and so on. I think there might be an opportunity to map our design considerations to those specific classes of applications and then manifest them in very specific advice. I think we can make a lot of progress in that area.

he Silver Bullet Podcast with Gary McGraw is cosponsored by Cigital and this magazine and is syndicated by SearchSecurity.

Gary McGraw is Cigital's chief technology officer. He's the author of *Software Security: Building Security In* (Addison-Wesley 2006) and eight other books. McGraw received a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.

CN Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.