Silver Bullet Talks with Fred Schneider

GARY MCGRAW
Cigital

Fred Schneider is the Samuel B. Eckert professor of computer science at Cornell University and chief scientist at the mutli-university TRUST NSF-funded Science and Technology Center. Fred is an associate editor in chief for *IEEE Security & Privacy.*

Gary McGraw: Over the years, your work has focused on building distributed and concurrent systems for mission-critical, high-integrity settings. What's the relationship between security and reliability?

Fred Schneider: They're really quite closely tied. Both of them are excuses for the system not to do what you want, and the difference is what the cause was. The reliability concern typically is about random events that one can attribute to nature, whether it's physical processes or the fact that people aren't very good at thinking about large systems, and they make programming errors. Security problems typically can be attributed to malevolence, where somebody is deliberately trying to undermine the system but in a way that either prevents the system from behaving as it should or prevents you from getting your work done.

McGraw: There are some people who claim that if we had a perfectly reliable system, that it would necessarily be secure. I assume based on your remarks that you don't really believe that.

Schneider: I don't agree with that statement, and I actually have a scientific basis for disagreeing. When you build a reliable system, if you look carefully, what you typically do is some form of replication. A simple way to replicate a system is to have multiple components running in parallel, and they all do the same thing and maybe you vote on the output. There are fancier ways to package that; instead of having multiple messages in parallel, if you add a checksum to one message, that's the equivalent of sending the message twice, but let's ignore those optimizations for a moment.

Fundamentally, reliability is obtained by doing replication, and there's an assumption made, which is the replicas fail independently. If you replicate things, then the probability that the majority of them fail is the probability that any single will fail raised to the *n*th power, where *n* is the number of boxes in the majority. If *t* is less than 1, that actually improves things. That formula only makes sense if the failures are independent. So, the assumption is, if one box fails, it's not going to be at the same time as when another box fails. If, for example, all the boxes are plugged into the same power source, and you blow a fuse, then all the boxes will fail.

There's a single event, and that formula which says replication helps actually isn't any longer sound. So, if you have separate boxes, then replication is a good thing, and it gets you a better system than running a single system by itself. But if you replicate a system, and the system you replicate had some vulnerabilities in it, and therefore was subject to attack, then all replicas are subject to the same attack. So you haven't made the system any more reliable; in fact, all you've done is given more places for the attacker to exercise his or her prowess. The independence assumption that most fault-tolerance people use—and often implicitly—is not valid if you're worried about building attack-resistant systems.

That's why security is really a different set of assumptions—change the assumptions and you're likely to get a different set of solutions!

McGraw: You've been a vocal proponent of diversity—viz your paper in the January/February issue of *S&P* called "The Monoculture Risk Put Into Context" [vol. 7, no. 1, pp. 14–17]. You talked about diversity as a useful thing there. If diversity amounts to building independence from a

About Fred Schneider

Fred B. Schneider is a professor in Cornell University's computer science department and chief scientist for the multi-university TRUST NSF-funded Science and Technology Center. His research focuses on understanding and supporting system trustworthiness. He was named Professor-at-Large at the University of Tromso (Norway) in 1996 and has a DSc honoris causa from the University of Newcastle-upon-Tyne. Schneider has a PhD from Stony Brook University. He's a fellow of the ACM, the AAAS, and the IEEE. Contact him at fbs@cs.cornell.edu.

failure perspective under attack, then we get right back around to the fault-tolerance stuff, I suppose.

Schneider: Right. So if we somehow were able to build diverse systems from the ground up, then that should increase the chances of building reliable systems because if you believe in independence, then it's not likely that there'll be an event that takes all of them out. And if you believe that attacks depend on details of the systems, no single attack is likely to take all of them out also. But the kinds of artificial diversity people are using and are practical are not completely independent. Typically, you take some program and you might generate different codes for it, say by randomizing the storage layout or using different compilers or using different instruction sets.

There, although the low-level stuff is somewhat diverse, the high-level stuff isn't. That is to say, all the replicas implement the same interface. So, if this is a program that has an interface which probably isn't called "give me the keys to the kingdom" but has that effect, then replicating it in diverse ways doesn't help, because an attacker who understands how to invoke this routine—"give me the keys to the kingdom"—ends up with the keys to the kingdom, and that's not what you want to happen.

McGraw: So that's like the level of defect. It reminds me of John Knight's attack on *n*-version systems.

Schneider: The challenge that we have is how do we build secure code? You and other people have focused quite extensively on writing secure code—those stupid things programmers can do that make it easy for an attacker to get in. And frankly, I'm mind-boggled at all the clever ways you can break badly written C code. We're now at a point where code security is a well-understood problem. It's not a soft problem in the sense that programmers don't write secure code, but we have the tools around—mental and developmental—to confront the problem.

If people stopped writing in C, if we outlawed that, we'd go very far toward a solution. We don't have a prayer of understanding, at the moment, how to design stuff at a higher level, how to pick good interfaces and know with some confidence that some combination of those interfaces couldn't be invoked by a clever attacker in the right order and get some behavior that nobody anticipated. That's going to be the next big problem we need to confront—at least those of us in the research business—hoping that those people who are actually building software will learn the lessons we have of common vulnerabilities and tools that eliminate them.

McGraw: I want to follow that line of thinking just a bit more. I heard you express this continuum of attacks and expected attacks in terms of large swaths of targets moving from configuration errors first, then to bugs—which are implementation defects we've just described—then to flaws, which is my terminology for architectural defects, and then to trust problems. So you believe that we're making some good progress on the bugs problem; we've already made some progress in the past on configuration problems; and so the age of bugs is beginning to draw to a close? Is that the gist of it?

Schneider: It depends on what telescope you're using. If you're in the research business—which is where I sit—the age of bugs is over. Spending a lot of energy in figuring out how to fix bugs is not likely to get you famous or have an impact. If you're in the system-building business, the age of bugs isn't over, because there are lots of systems out there that don't practice the lessons we've taught, and people are still building systems the old way. So, there's this huge incentive problem that needs to be addressed, and the incentive problem would cause people to practice what we have preached.

If you're in the research business, then what you term flaws or these architectural problems are the goldmine in the sense that there are lots of opportunities to make progress and give insights. The other piece of the picture that this taxonomy of attacks brings has to do with a tactical issue. If we're in the business of trying to anticipate where attackers go, then this taxonomy gives you a basis for deciding where to make investments for future research so that you will have defenses in place before the attackers have an incentive to try to address that class of flaws. As long as there's low-hanging fruit, the attackers aren't going to go for anything higher.

As we understand how to get rid of that low-hanging fruit—like, for example, the Common

Desktop Initiative in the federal government, which says everybody buys the same machine, the same software, and it comes configured by somebody who knows what they're doing, and therefore there are fewer configuration flaws—well, do that, and all of a sudden the attackers have to worry about code security. If we keep forcing the attackers down the path, then we would be wise to have positioned ourselves for the next step of the path before we force the attacker there. This taxonomy then gives you some insight into the kinds of questions people ought to be thinking about now in anticipation of dealing with attacks in five or 10 years.

McGraw: That's a beautiful line of thinking. I'd like to know a little bit more about the stuff past flaws; that is, what you've termed trust problems. Do you have a few words to say about that?

Schneider: Trust problems, as far as I'm concerned, are the hardest ones. A trust problem is when for various reasons we structure our systems in terms of enclaves, where the definition of an enclave is that systems within an enclave have stronger basis of trust than systems outside the enclave. This usually corresponds to organizational boundaries. I trust the file server in my building more than I trust the file server at the other end of the Internet that's run by some other organization. Once you trust local machines or machines within your enclave, if any one of those machines gets compromised, it becomes a launching pad for an attacker to get to any of the other machines.

Rather than getting in by breaking in, the attacker can get in just by asking for legitimate things—because, after all, it's a legitimate request. It came from something that was trusted. So this kind of attack is only addressed if you become paranoid. That is to say, because someone's trusted doesn't mean they get to ask for anything they want. That puts us into the realm of building systems, say, with least privilege. We don't know how to do that, and what we do know is if you have least privilege it means you've got a lot of privileges around because you want to know everything slightly differently. Read, write, execute—not just at the level of a file, but maybe at the level of a record or a field; not just at the level of a module, but maybe at the level of an entry point. That's an enormous bookkeeping problem. We know now that people, security officers and others, have trouble managing even the access control list associated with files. Role-based access control is partway toward a solution, but even that is hard for some people to manage.

McGraw: You've been very much involved in commercial endeavors in computer security as well as your academic endeavors, including work with Cigital's TAB and Microsoft's Trustworthy Computer Academic Advisory Board [TCAAB], and Fortify's TAB. What influence have you had on the state of the practice in the commercial world?

Schneider: The agreement usually when you get involved in one of these things is that you won't air their dirty laundry in public. And that makes it hard for me to give concrete examples of where I've said something, and it's caused somebody's product line to change.

McGraw: Well, why don't we zoom up several jillion levels to the notion of academia influencing commercial endeavors in the right way?

Schneider: Right. There certainly are various examples of academic protocols ending up in commercial deployment. Some of the work that's done in Microsoft for analyzing I/O drivers to make sure they won't lead to blue screens is based on model-checking work that had its origins in academics. If you use Kerberos for authentication, that had its origins in the Needham-Schroeder authentication protocol, but the first Kerberos actually was built as a project at MIT. The StackGuard scheme that the C compiler—one of the C compilers—uses and is fairly widely used is something Crispin Cowan did when he was at a university.

So, university stuff does end up getting out. I think more importantly, at least for big companies, is if you get the right people in the room, you get a set of perspectives that's broader than simply a technical one.

You can read more from Fred Schneider in this issue's EIC message on p. 3, and find additional podcasts in this series, including those featuring Kay Connelly, Bob Blakley, Gillian Hayes, and Christofer Hoff, at www.computer.org/security/podcasts/ or www.cigital.com/silverbullet/. ☐

Gary McGraw *is Cigital's chief technology officer. His real-world experience is grounded in years of consulting with major corporations and software producers. McGraw is the author of* Exploiting Online Games *(Addison-Wesley, 2007),* Software Security: Building Security In *(Addison-Wesley, 2006),* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001), and five other books. McGraw has 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.*