Silver Bullet Talks with Gary McGraw

Marcus Ranum | Tenable Network Security, Inc.

**Over the past decade, Gary McGraw has interviewed some of the security industry’s most influential gurus for his monthly Silver Bullet security podcast. A globally recognized authority on security and software, McGraw is the chief technical officer of Cigital, a software security consulting firm with headquarters in the Washington, DC, area and 13 offices throughout the world, and the author of eight bestselling books on software security. He also serves on the Dean’s Advisory Council for the School of Informatics at Indiana University, where he earned a dual PhDs in cognitive science and computer science. For the 120th episode of Silver Bullet, we flipped the tables—I took the role of host and interviewed McGraw. You can watch this special video edition at www.youtube.com/watch?v=LIvYFaNIcI.

You’re one of the founders of the software security field. How did you get into that?

I was a programming languages weenie, besides being a cognitive science guy, when I was in grad school. When Java came out, I was excited that there was a programming language for the Web. It turned out to be much more like C++ than a functional programming language like Scheme. They were making all these claims about security, so we started breaking Java. I wished it was better, and wondered why the guys who built it made some of the decisions they made from a programming languages perspective.

**Did you believe the marketing hype that Java was a secure programming language?**

It was called Oak in the early days. It was actually built for set-top box cable systems and was very closely based on P-code from the 1970s. When it came out and they were saying "everything you build in Java is secure," I wondered what the heck they were talking about. I did a lot of work with Ed Felten from Princeton and his colleagues, who were in grad school at the time. It made me wonder why really good people screwed this up when it came to software security. If Bill Joy and Guy Steele can’t get it right, what chance do mere mortals have?

The way the software industry is structured right now, it seems like it’s vastly more rewarding for people who are part of the problem rather than people who are part of the solution. If you try to produce good software, you’re going to be later in the release cycle, and the other guys are going to have five million customers before you release anything.

There’s that, and there’s also the idea of being incentivized to put bugs in that you later take out.
If Netscape had waited three releases to release the browser, it would have never been what it was. Where were you when Netscape went public?

I was still at Indiana University when Netscape went public. You know, Netscape really didn’t have Java in the beginning. It isn’t a Java problem. Kent Landfield [of Intel] and I were in the data-center at Sterling Software the day the Netscape IPO happened, and we watched that ticker go through the roof. It was the beginning of the 20-year beta test. You knew the world was going to change. Now we have DevOps, where we can make that cycle even tighter.

You’ve been on advisory boards for many companies. What do you do? You’re supposed to be a third set of eyes and kind of a “person from Mars” perspective for the company you’re giving advice to. Usually, they’ll put on a formal presentation about a new design, product idea, or strategic move they’re going to make as a company. The advisory board weighs in on that, helps get the flaws out of the design, and talks to the engineering team directly. Sometimes it even takes a look at code, which we did in the very early days at Fortify. It generally provides outside council that’s highly trustworthy.

I think it’s really interesting, when you’re in the sausage factory and you’re actually packing the sausage and someone comes in from the outside and says, “Did you think about doing this?” It really happens when you’re coding. You have this bug, and you can’t figure it out. Your friends call you because you’re supposed to meet them for pizza, and they’re like, “Where the heck are you?” They finally come by your office and you’re still working on this bug, and it’s driving you crazy. One guy just says, “Oh, why’d you do that?” And you say, “That was the bug! That was it.”

The Internet of Half-Baked Things is upon us. The other day I found this wonderful item on Amazon— the Bluetooth 7000 Dental Professional Toothbrush. Bluetooth isn’t a problem—it’s a local network. But how does the brush get data to the cloud? The base of this toothbrush has an IP address, and it’s DHCPing [Dynamic Host Configuration Protocol] something on your network and going to some website, and personal information about how often you brush your teeth—but more likely your YouTube password—is going to be on that site. People look at this and say, “There’s no way a toothbrush can actually hurt people, so we don’t have to worry about securing a toothbrush.” But they don’t realize that it’s using a finite resource. And that’s the key thing to think about when you’re designing these things.

It also means that if I can get access to your toothbrush records, I can tell when you’re on vacation because you’re not at home brushing your teeth. And I know it’s time to break into your house and steal your toothbrush. How do you see this playing out? Are you going to be doing application whitelisting on your toothbrush? What if somebody gets a buffer overrun on your toothbrush? Or more to the point, what if you’re running a website and you find yourself under a distributed denial-of-service attack from 10,000 toothbrushes?

Software updates for everything! I think there’s some possibility that because these devices are so small, we might be able to write better code because it’s going to be smaller, tighter code. But that might be truly insane.

Software updates for everything! I think there’s some possibility that because these devices are so small, we might be able to write better code because it’s going to be smaller, tighter code. But that might be truly insane.
these embedded systems, including some car chips. The difference between a Ford 350 and a Ford 150 is a bunch of EEPROM [Electrically Erasable Programmable Read-Only Memory] settings. You might get a cheap chip because there are billions of 8088 chips sitting around, for example, in a warehouse. It’s cheaper to put that into your dishwasher than it is to store it in a warehouse. So you end up having way too much computing capacity in your toothbrush, in your dishwasher, and everywhere else.

The other problem with this is that when you’re dealing with a physical device that gets shipped through Amazon, you have to get the software right the first time because you can’t destock it and reship a million light bulbs.

The same thing doesn’t work too well in the mobile space. If you think about Sony or Sony Mobile, who has to have all these devices going back to five years ago and support them all, you know that the kernel and the code that’s associated with Android isn’t being updated to keep things normal. As a consumer company, it’s on the spot to make sure that even a five-year-old phone still works.

Without naming any names, what are some of the most screwed up things you’ve seen in the Internet of Half-Baked Things?

I worry when the code really matters, like control code for a vehicle, airplane, or nuclear power plant. You can’t screw that up. You can’t say “Oops, I’ve made a mistake; let me just DevOps that later,” because you might create a 30,000-year problem if you screw up the code in a nuclear situation.

We’ve learned a lot of lessons from “high-assurance land,” some of which we’ve been borrowing over the years into “nonsense software land” (where we exist—banks and consumer devices and everything else). Now it’s trickling down to even sillier things like toothbrushes. But we can still borrow good ideas from high-assurance software and apply some of them while we’re building these things.

What do you think about Charles Perrow’s normal accident theory? Does it apply when you start talking about a toothbrush? Is there a situation where your toothbrushes could exhaust your DHCP address and cause a reactor failure somewhere else? Is that our future?

It might be, because everything is ridiculously interconnected. If you’re dumping noise from all of your toothbrushes onto the Internet and a really important message needs to get by but the toothbrush traffic is too heavy, what are you going to do?

When you’re in computer science school, on the first day of class, you don’t get the bejesus scared out of you. If you’re in a mechanical engineering class, you watch a bridge shake itself to death, and you’re 20 and saying, “I don’t want to make a bridge that falls into the ocean.” In computer science school, it’s kind of like, “you can build anything you want.”

I was once on a panel with Larry Wall [creator of the Perl language], who said, “Perl is fantastic because there are six ways to do everything.” I spoke right after him and said, “Perl really sucks from a security perspective because there are six ways to do everything and you only thought of blocking four of them.”

Years ago, I said the only thing that’s going to get the world to take this problem seriously is a “software Chernobyl.” It seems like some disaster is inevitable.

There have been some: the Ariane 5 rocket [which exploded 40 seconds after liftoff in 1996 due to software design and specification errors] and the Therac-25 [a radiation therapy machine] that burned a bunch of people to death in the 1980s with radiation [due to programming errors]. Life-critical systems have to do better. I think we’ve actually made progress with building consumer-grade commercial systems over the past 20 years. Although, you might argue that we’re growing so fast that security will never catch up.

I’m not sure I’m concerned with the growth factor as much as the governments uncorking this new genie: now we can screw you up by causing you economic damage. By making your new reactor no longer function, it’s going to cost you millions of dollars.

Or slow down your nuclear weapons program by making your centrifuges fail to work properly.

Which, of course, is a violation of the Geneva Conventions because you’re not supposed to mess with nuclear power. That’s where I always like to introduce the concept of a weapon of privilege in cyberwar—one that I can use against you, but if you ever dream about using it against me, you’d better wake up and apologize. There are many of those because we have a nuclear deterrent in the US as well, so you have to factor that in. Something could go kinetic.

An important thing to realize about cyberwarfare is that the best way to avoid it is to engineer stuff that’s much better and more expensive than anyone else’s—basically outspending our adversaries by building things properly. The deterrent is that our engineering is great and theirs isn’t so great.

You could deny your enemy a seat at the battlefield. You make it so much more expensive for them, which is basically what’s happened with nuclear weapons around the world—the cost alone ought to deter rational people.
I think cyberwar is complicated economically because of the low cost of building something to disrupt some important piece of life like water or power control systems. Developing something to screw that up might cost US$10 million, not $10 billion. And the number of countries out there that have even a $6 billion budget for defense is pretty big.

At Source Boston, one of the keynote speakers was talking about evolutionary models in security. My friend stood up at the end and said, “I am an actual evolutionary biologist and I have to tell you, for evolutionary algorithms to work, you have to be willing to suffer billions of casualties.” We’re not prepared to suffer that in computing. What happens when an entire operating system’s tree dies? Or is that what is happening to Java? That branch of programming is dead. At this point, you’d be crazy to invest in writing more Java code. There are an awful lot of people still writing enterprise Java code to do things. I don’t think old code ever dies. That’s one of the really terrible things about programming—if you wrote it 20 years ago, it’s still somewhere and somebody’s going to find it and run it. And it has your name on it!

All these vendors creating security mechanisms with millions of lines of new code think they’re building more security to put in front of the broken stuff, but they’re actually increasing both the pile of broken stuff and the attack surface. So, if you realize that the cloud is somebody else’s computer, and you realize that the security mechanisms are somebody else’s code, you think twice about the way we’re doing some parts of computer security.

It seems to me the US federal government is outsourcing a lot of stuff to beltway bandits that write the code. They don’t know anything about software security. Think about the vast middle market, and these people that are system administrators for mom and pop shops or cousin Joe who knows something about Windows. Those people are probably better replaced by the engineers at Google, Amazon, or Microsoft. So, in some sense, movement to the cloud can be a good thing from a security perspective, even though it can be a bad thing for some people who do know what they’re doing. It depends on where you sit and how you want to operate your enterprise.

You play violin, you mix drinks, you raise goats, you live on a farm where birds poop on your interviewers. What else do you do? What are some other fun things you do to keep busy?

I read a lot. I like to read fiction. I just read a new book by a woman who just graduated from New York University named Julia Pierpont [Among the Ten Thousand Things]. It’s fantastic to see a young person writing with such insight about human existence. It refreshes my feeling about humanity not being a terrible, horrible disaster when you read the thinking of some of these people. That makes me happy.