Before moving over to Google, Halvar Flake ran Zynamics, a company specializing in reverse engineering tools, as an independent researcher and consultant focused on reverse engineering and vulnerability discovery. Flake has spoken at RSA, Black Hat, CanSecWest, and other venues; he has also taught classes on code analysis, reverse engineering, and vulnerability analysis for independent software vendors and governments. Hear the full podcast at www.computer.org/silverbullet or www.cigital.com/silverbullet.

Gary McGraw: Congratulations on the purchase of Zynamics by Google! Can you describe how Zynamics began and evolved, and what led to Google's interest?

Halvar Flake: Many years ago, Microsoft started putting out patches without disclosing the full details of what they were doing. I was interested in multiple security vulnerabilities—more specifically, patches—so I started experimenting with executables, comparing not the bytes or byte sequence but the flow graph structure. I wrote the first prototype for BinDiff in 2004, I think, and started selling it. Truth be told, if you sell a piece of software and you’re just one person, people want source code escrow, but if you’re an incorporated company, people don’t ask the same sort of questions, so I started a small company that originally operated under the name of Sabre Security. After BinDiff, we launched a second product not much later called BinNavi, and it took off from there. Eventually, we got into a bit of a trademark dispute with a large travel agency about the use of the word “sabre” for delivering services over the Internet. We decided that we didn’t want to, well, compete with somebody whose legal department was 50 to 60 times our company size…

McGraw: So you switched your name.

Flake: Right. We picked Zynamics, which goes to show that not all democratic ways of choosing names end up in a good pick.

McGraw: BinDiff is an important product. Could you explain a little more about how it does what it does?

Flake: The real problem when you compare two executables is that although they might have been derived from the same source code, they’re usually created in vastly different build environments, which means you’ll see different compiler versions, slightly different versions of the source code, and so forth. All these things have a cascading effect, to where the final binary looks quite different from the other binary, all because of the build environment.

When it comes to patch differencing, at least in Microsoft’s case, it’s not that bad today because build environments don’t change much between two particular updates. Back when we wrote BinDiff, release binaries from Microsoft were optimized in a particular way, and this screwed up a lot of the standard ways of comparing executables.

So, with BinDiff, we disregard the actual byte code for as long as possible; we create flow graphs for every function in the executable and a big graph called the call graph, which relates all the functions in the executable to each other. Then we compare the executables based on these graphs, which means we’re trying to compute an approximate maximum graph homomorphism, which essentially says “let’s try to find those chunks of the graph that are structurally identical and then just map them to each other.” The advantage is that a lot of the small changes introduced by compiler optimizations and so forth disappear, so you have a lot less noise...
Interview

and you can associate pieces of code that don’t immediately look obviously similar through their structural properties.

**McGraw:** This allows you to identify places where the patch has changed the executable in interesting ways and then focus in on the original vulnerability?

**Flake:** Correct.

It also allows you to do other things—for example, if you’re reverse-engineering a piece of software and you see a string that indicates a particular version of some large open source library has been linked into the binary, you can then use BinDiff’s algorithms to compare your executable against a version of the library that has been compiled with debug symbols and pull that information from the version with debug symbols into your disassembly. It saves countless hours of work, because, well, you get all those names more or less automatically.

Another advantage is that you can pull information from one disassembly into another disassembly. Let’s say you’re analyzing two variants of the same malware—you can disassemble one malware, comment it all, and reuse the results on the new version of the malware.

**McGraw:** I assume, based on this conversation, that you must be very psyched to be getting back into more hands-on engineering and less startup tending.

**Flake:** Oh, most definitely. One of the many things that Google offers is relieving me of the business side of things, allowing me to focus on what I do well, which is engineering and research, and I’m really happy about that.

**McGraw:** Do you believe, as I do, that security researchers and bad guys are making better use of classic code—understanding tools, like decompilers and disassemblers—than people who build code for a living?

**Flake:** I’m not sure whether I would necessarily agree—the focus of these tools is quite different. Realistically, if you look at the state of reverse-engineering tools and the state of engineering tools, I think the engineering tools are far more advanced than the reverse-engineering tools; simple economics dictates this, because many people write code for money. A comparatively small number of people reverse-engineer code for money, and as such, it’s much easier to get resources for developing tools than for reverse-engineering them.

**McGraw:** I agree with that, but some of these reverse-engineering tools are very useful for code understanding in the first place, and yet the people who build code for a living don’t, generally speaking, make use of that technology.

**Flake:** At Zynamics, we had both development and code analysis, so we were developing tools for people who need to understand software. One of the devs on my team always joked about security review as the job that nobody who actually likes programming wants to do, because if you like programming, you like creating stuff, and you like seeing stuff run, whereas security review looks at those few corner cases where things don’t work. The security reviewer doesn’t really care about the software per se, but about the moment where he subverts the software.

**McGraw:** When it comes to reverse engineering, in your mind, what’s more powerful, static analysis or dynamic analysis?

**Flake:** I don’t think there’s a good answer for that—it depends, really, on the situation.

**McGraw:** It’s actually a trick question.

**Flake:** When I was much younger, I used to play basketball, and the coach would always correct people’s shot techniques, but he had one golden rule, which was, whoever scores is right. I think that very much applies to code analysis in all sorts of ways. One thing that I do appreciate about the security community is that we’ve got a very, very simple truth test: does it find relevant bugs or not? This is what keeps research honest, because you can’t argue your way out of trouble with, “Oh, my stuff is good, but it doesn’t find X, Y, and Z...” So from a technical point of view, dynamic analysis often makes a lot of things much easier, but you pay a price, which is vastly reduced coverage. I’m not a fan of an either/or distinction on dynamic or static, but I’m a huge fan of using dynamic wherever possible to augment or improve on static and vice versa. It’s not a binary choice.

**McGraw:** Much of your work focuses on finding and leveraging bugs and implementation defects. What do you think we can do to find flaws or architectural defects?

**Flake:** I don’t think we have many good ways of doing so, because in order to find architectural defects, we would first have to have a clear understanding of what we want from an architecture. Textbook software engineering offers a full list of requirements, but that’s not reality. You might have a Porsche in your garage, and it might run beautifully, but unless you work in car engineering, you won’t realize all the engineering trade-offs that had to be made to get that thing to run at that price, and you won’t
realize how messy a process even engineering a car is. Software isn’t much better, so trying to understand or having tools to improve the finding of architectural flaws, I mean, it boils down to understanding again, right? And the sad truth of the matter is that people want to avoid understanding at all cost. And when we’re speaking about tools that help uncover architectural flaws, we’re really speaking about tools that make the process of understanding something large and complicated less painful and more time efficient.

**McGraw:** I agree 100 percent with you that the disclosure debate is really stupid, but I wonder what your feelings are about alerting vendors and others when a new vulnerability is discovered.

**Flake:** Before Google bought Zynamics, I would occasionally discover defects. I wouldn’t report them to vendors for one reason, which is that I don’t have the time to deal with vendors. My experience might have been forged in a different time, back in the early 2000s, but now when you find a bug and notify the vendor, the first thing you get is a reply from the legal department, threatening you. If you’re lucky, eventually somebody with development experience will want you to spend a couple of days of unpaid time explaining the bug to them.

**McGraw:** Free consulting.

**Flake:** So, in the optimum case, you’re going to spend three days of your life helping somebody else do their job and probably not even get as much as a handshake for it. In the worst-case scenario, people will start throwing the legal textbook at you or have their lawyers write threatening letters. From my perspective, there was no point or incentive ever to do this.

**McGraw:** I think things have improved somewhat for some of the vendors. We find problems all the time in the field, and we send them directly to the product groups, and they do a pretty decent job of fixing them. Sometimes they ask for more info, but often, when we find a problem, we’re working for a customer, so we’ve already written the stuff up in a pretty easy-to-package way. I guess that makes it a little bit easier.

**Flake:** Oh, definitely, and I think there’s been a big change in attitude as well. To a certain extent, you can argue that all these bug bounty programs at least create some incentive to sit down and document them.

**McGraw:** I still think they’re a poor excuse for not paying a real quality assurance staff.

**Flake:** Yes, but at least what I can say about my current employer is that even though we have a bug bounty program for Chrome, it’s not like we don’t have internal review. I guess my point is that if you make the process of submitting issues convenient and fun, then you’ve got a decent shot at getting people to submit stuff to you. If you make it inconvenient and a hassle, you can’t complain about not getting anything.

**McGraw:** You’ve spent some time poking around Zeus and its huge pile of variants. In your view, is malware getting more complex, or is it about the same as it ever was?

**Flake:** It’s definitely not the same as it ever was. If we go back 10 years in time, we’re speaking about a significant growth in the magnitude and sophistication of malware.

**McGraw:** Slammer, Code Red, and Nimda were pretty easy.

**Flake:** We’re speaking about file infectors, which I have to admit had very, very sophisticated polymorphic engines, but malware has moved from what used to be interesting assembly experiments to writing good code to proper software. If you read the code, it’s like, okay, they have an architectural diagram on the wall somewhere.

**McGraw:** Somebody actually designed it—it wasn’t just whacked together.

**Flake:** Right. With a release schedule and a feature roadmap—it’s literally just like software development.

**McGraw:** Let’s talk about Stuxnet a little bit. It shows a fair amount of engineering preparation.

**Flake:** Well, both on the good and the bad side. There were parts of Stuxnet where you’re like, okay, now clearly they had one person with a lot of clue drafting the design, and then he delegated some
of the code to people with somewhat less of a clue.

**McGraw:** I think there was a pretty big discrepancy between the delivery vehicle and the payload in Stuxnet in particular.

**Flake:** Well, that’s what you get when you’ve got a large organization, and Stuxnet was built by a nontrivial-sized team. It’s difficult to get 10 insanely bright people to work on one thing, so what you usually do is you get three or four of them, and then you fill up with slightly less bright people.

**McGraw:** Now for some politics. You were denied entry into the US in 2007 when you were supposed to teach at Black Hat, which was just silly. Are you still on some kind of government list? How has that unfolded over the past four years?

**Flake:** What happened back then was a huge misunderstanding about my immigration status. The immigration person at the border decided that my going to Black Hat and Black Hat being a for-profit organization meant I needed an H1B visa, and they rejected my entry based on this. We can argue back and forth about what exactly would’ve been the right way, but, to a certain extent, there’s an interpretation of the regulation of the law where what they did was completely the right thing.

**McGraw:** It’s just a faceless bureaucracy you grind up against.

**Flake:** Whenever I cross a border, they still pull up my data and ask me about what happened back then. For about a year and a half or two years afterward, I got a lot of in-depth questions, and I would be sent to the back of the line. But after a certain number of situations where they’ve asked me questions and decided that I’m not lying, they now wave me through fairly quickly. It’s gotten to be a very pain-free process.

**McGraw:** It’s good to hear that bureaucracy can self-correct eventually.

**Flake:** I think it might be economics. They’ve got limited resources, and they can’t check everybody.

**McGraw:** People like Fred Schneider in previous iterations of this podcast have talked about the future of security issues writ large. Fred’s belief, and I think I agree with him, is that once we get done dealing with architectural flaws, the next thing will be trust enclaves, and trust transitivity will be the core of what we have to figure out.

**Flake:** The question is how quickly you end up putting restrictions on innovation. There are good reasons why you usually don’t have capital control when you run markets—you don’t necessarily want to restrict the flow of money across borders—but you can make arguments for restricting the flow of money in some situations. Similarly, with information, if you restrict it too much, you run the risk of, well, you can deny an attacker your information, but your organization is going to wither and die anyway, because you’re restraining yourself. It’s going to be an interesting and then very hard problem to deal with. How do you restrict access to information while, at the same time, providing information? The US intelligence community is probably running into this exact issue.

**McGraw:** I think about the US obsession with cyberwar over there on your side of the pond?

**Flake:** No. I’m not sure what sort of hype is being blown these days, but generally speaking, computer attacks are a valid component of espionage, which is just a reality, and it’s very difficult to deny that computers and espionage go together, or intelligence and computers, whatever you want to call it. I think it’s a good thing that these things are discussed openly. It’s much healthier if military officials go on record and say, “Yes, we do this,” instead of standing up and denying it. In a healthy democracy, the public is informed about what the military is doing.

**McGraw:** What kind of music are you listening to these days, and what’s your favorite?

**Flake:** Generally, my taste in music is, according to my girlfriend, quite horrible. It ranges from German rap from the mid ’90s to Argentinean tango to jazz from the ’30s. A band that I very much like is Jancee Pornick Casino. They mix heavy and surf punk with Russian folk music.

Show links, notes, and an online discussion can be found on the Silver Bullet webpage at www.cigital.com/silverbullet.

Gary McGraw is Cigital’s chief technology officer. He’s the author of Exploiting Online Games (Addison-Wesley, 2007), Software Security: Building Security In (Addison-Wesley, 2006), and seven 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.