Control-Flow Enforcement Technology (CET)

Description: Father Robert and I begin by catching up with a week of mostly clickbait stories and case studies of real-world insecurity. Then we take a very deep dive into the operation of Intel's forthcoming anti-hacking chip enhancement known as "Control-Flow Enforcement Technology."

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-565.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-565-lq.mp3

SHOW TEASE: It's time for Security Now! with Steve Gibson. And, boy, do we have a propeller-head episode for you. Palantir got owned, but in a good way. Two-factor is no factor with SMS. Your cameras are probably spying on you. And Steve pushes, pokes, and pops with stacks. Security Now! is next.

FATHER ROBERT BALLECER: This is Security Now! with Steve Gibson, Episode 565, recorded June 21st, 2016: Control-Flow Enforcement Technology.

It's time for Security Now!. It's the part of the Internet where we only trust you if you trust no one. And the man who trusts the fewest people is none other than Steve Gibson from GRC, Gibson Research. Steve is my personal security guru and the big brain behind Gibson Research, SpinRite, and of course ShieldsUP! and the coming non-password overlords. Mr. Gibson, it's so good to see you.

Steve Gibson: Well, it's great to see you for a change. We have Leo on vacation with his family, actually somewhere around here in Southern California. I think they're doing the various amusement venues, and also being in a boat in Newport Beach or something. So you're, I guess, filling in for at least this podcast. I don't know if you're going to be doing any others, but we're delighted to have you. We always get a lot of great feedback, Father, when you're my co-host. So I'm glad for it.

PADRE: Oh, that's fantastic. I always enjoy it. I will say, I like it that Leo's getting out. He should enjoy life. He should get out there.

Steve: We want to keep him fresh.

PADRE: Keep him fresh. But I do enjoy when he goes away because it means I get to chat with you for a few hours.

Steve: Well, and coincidentally, this ended up being a perfect one because, as I

promised our listeners last week, this will be a wind up your propeller beanie and concentrate because what I want to talk about is for our main topic, after we cover the week's news, is some forthcoming hardware technology which Intel has added, or will be adding, to their processors to finally, hopefully, meaningfully help with all of these, like many - well, not all. But, well, maybe all of the buffer overrun problems, and many of the ways that hackers have found to cleverly exploit systems using vulnerabilities in code in order to essentially arrange to get their own stuff executed. Intel calls this - abbreviates it CET, which is their abbreviation for Control-Flow Enforcement Technology.

So in order to talk about this, we're down where I live, down at the machine level, with registers and the stack and instructions and things. But it's understandable. So I think everyone's going to enjoy spending some time, not up in fluff land, but a serious technology podcast this week. And you're a perfect co-host for it. So it's perfect.

PADRE: Well, I mean, it is an interesting list of topics. It's basically everything is busted. Our CPUs, our SMS, our data analytics, our Internet of Things, pretty much everything we use today is going to be kind of beat up today.

Steve: So there was a story about Palantir, the very well-known sort of security company that was actually founded with some CIA dollars, getting owned. Some confirmation of something that I had said a couple week ago when I was choosing what second factor to use, a reason why SMS is really not something we can trust as much as we would like to. A frightening true life experience with the Internet of Things, perfectly following on the last two weeks of topics. Or I guess actually three weeks ago was we did a pair of episodes that I called "IoT Infancy," talking about the fact that the Internet of Things world is still trying to get itself going.

Also GoToMyPC had a massive password reset, but their coverage of their own problem was inconsistent and odd. And then of course something that was probably the most tweeted thing to me was the story of a hidden rootkit computer underneath our main computer. And so we're going to cover that and then get into a little bit of miscellany, but not much, and then a great topic. So I think we have a great podcast for everyone once again.

PADRE: Indeed we do. And this will be the no-fluff, no-fat podcast, only the meat. Only the best of the marrow is what we're going to be bringing you. Steve, tell me about Palantir.

Steve: Well, you know, I got all worked up thinking, wow, how hired hackers got complete control of Palantir. And so I dug into the story thinking that there would be a lot of meat there. And I came away thinking that, okay, this is just clickbait. First of all, Palantir hired a group of hackers and then deliberately let them in. So it was like, wait a minute. Okay. So what they did was they simulated somebody within the company, well, the simulation was of their making a mistake, clicking on a piece of phishing email. But everyone knew that was positioned to allow this so-called "red team" of hackers to establish a foothold within Palantir.

What was annoying was that this whole process, I mean, this sort of internal security review is something that responsible companies do all the time. And I was annoyed because Palantir ended up with egg on their face, where the way the article came off was that they were blundering in some way, that they had made a mistake, like hackers got complete control. And the real story here is that somehow the report that the security company generated under Palantir's pay and permission escaped to the press because that's what normally doesn't happen. No, these reports never looked good. They're not something that you want the public to see. They're meant for internal purposes only.

And so I salute Palantir for being proactive about their security. They simulated a very possible real-world event of somebody in the company falling for phishing email. And then the test was, once people got in, what could they do? And very much as we saw in the case of the Sony attack, any network which is sufficiently large and has been around for a while, stuff has been bolted onto the sides because, oh, we didn't realize we need another database over here; and oh, look, we need another, bring another website over there. These networks, in large corporations especially, just sort of grow organically. And what was originally well planned ends up just naturally evolving into sort of a Rube Goldberg contraption, which has all kinds of exceptions and corners that haven't been visited recently, and ends up not really being designed, but evolved.

And so, yes, the bad guys were able in a really large company to, once they got in, with Palantir's permission, to set up shop and find credentials and use some credentials to elevate their privileges and basically take over the network. But again, the report they produced provided Palantir with really useful feedback that allowed them to then respond by fixing all of the problems that the employed benign hackers were permitted to come in to find. So it was a little disturbing to see Palantir's PR branch trying to explain that, like, okay, this is the way the industry operates. This is how a security-conscious company functions is you hire a red team to come in and see what they can do. So, like, okay, on one hand, yes, these people got complete control. But it was intended. And Palantir benefited, and all of Palantir's customers benefit hugely.

PADRE: That was the disturbing part of the story for me because, like you, I got a lot of people tweeting me links to this. And all of the links were - they were link bait. It was data analysis company that does work for the government is breached. And that wasn't the story. This is actually a security firm that did it right, that hired an outside consultation firm to come in and test their security. They let them in. So none of the initial borderline security was in play. This was only what would happen if one of our employees accidentally let you into the network. And a lot of the - I read a more detailed whitepaper on some of the hacks, and it was a lot of pivoting. So that's one of the most difficult things to account for right now.

But if someone takes over a computer and then is able to pivot it to be the attacker inside, but they're smart about it, so that it doesn't always throw off attack traffic, there's no network I know that can survive that. It doesn't matter how segmented your network is. Once they start pivoting on the inside, they're going to get access to credentials. They're going to get access to certificates. But what Palantir did was they were very open about where the security flaws are, and now they can fix them. This is what we want to encourage in the industry; right, Steve?

Steve: Right, right. And so that's exactly right. I think no one should be put off by this, the idea that they're being proactive. I mean, for example, they're in a much better position today, except from a PR standpoint, than they were - I think this was done, I think it was in October and September, so the end of September and into October of last year. And Palantir's customers have a stronger company they can rely on as a consequence, not a weaker company.

PADRE: Right. In fact, I'd go one step further. I'd say if you're looking at trusting your data to a data analytics firm, don't trust them unless they release the results of their latest breach test. I mean, that's what I'm going to be looking for. I'm going to be looking for a company that is forthcoming about any potential hazards to my data while I have control of it. So I want our audience to get out there and tell people, no, no, no, no, you've got that story all wrong. The fact that you're hearing about it is good.

In fact, Dan Geer at Black Hat two years ago, his keynote address was all about transparent failure. The goal of our industry should be 100% transparent failure. When something happens, you should not feel ashamed. You should not feel reluctant to release that information because that's how the security industry grows. So, yeah, I'm with you. I think this is something that [crosstalk].

Steve: Yeah, I guess the only modification I would make to that I that they never intended that internal report, which they purchased, essentially, from these hackers, I mean, the question is, how did this leak to the press? Because this was juicy for the press. And unfortunately the spin that got put on it, as you also saw, was a negative spin. Instead, it was intended to be internal and to stay internal. And so Palantir would say to their clients, either current or future clients, we hire red teams to attack our networks on a semi-regular basis so you can be assured that we're taking our internal security seriously. And it just shouldn't have gotten loose because the value is to Palantir and to their customers, not to the public at large.

PADRE: Right, right. And hopefully I hear more. I'd love to hear more companies release the findings of their penetration tests.

Steve: Yeah.

PADRE: All right. What about - on my device right now I have a super secure - I'm sorry, did you want more on the Palantir story?

Steve: No, no, no. That was perfect. I was just going to say that a couple weeks ago I was setting something up, and I don't remember now what it was, but I told our podcast listeners and Leo that I had a choice between having an SMS message sent for second-factor authentication, or using a one-time password, I mean, sorry, using a time-based, like a Google Authenticator or Authy-style six-digit temporal-based token. And I didn't hesitate to choose the time-based solution. And I said at the time that the reason I did that was that it's much more secure.

The idea there is that one time over a secure TLS connection to whatever the server was I was establishing this with, I don't remember now what it was, it provided me with the cryptographic information required, basically the key to key the standard time-based protocol, TOTP, which would then generate a series of six-digit numbers, given an authentication application. And back then I said I'm not using SMS because the problem with, as we know, with our cell phone system is that it's not secure. SMS messages are not secure.

And the way, with the choice I made during a truly secure point-to-point connection, I obtained one time a secret from that server which would then, for the rest of time, drive this time-based token generator. And so never again would they need to be providing me with a secret on the fly, which of course is exactly what simple messaging system (SMS) messages does. It's like every time you're logging in, send me a message to my phone, which I then look up and enter into the web page.

So anyway, this all kind of came back, and I got a kick out of this because a security company, Positive Technologies, put out a press release that popped up on my radar. They wrote, and this is in press release format, I'll just read the first couple paragraphs: "Positive Technologies researchers able to compromise many popular social media sites by hacking the SS7 network, intercepting a one-time password, resetting passwords, and taking ownership of accounts."

So they wrote: "Positive Technologies, a leading provider of vulnerability assessment, compliance management, and threat analysis solutions, today confirmed its researchers have exploited a flaw in the SS7 protocol to intercept one-time passcodes used by many online services to reset passwords. Facebook, WhatsApp, Telegram, Twitter and many other online services offer password resets via SMS message. But instead of strengthening security, this ability actually introduces a vulnerability hackers can, and will, exploit. Positive Technologies researchers recorded themselves demonstrating the hack against Facebook and WhatsApp accounts with the owner's permission, proving the dangers of this authentication method."

And so we had been talking about the SS7 system and how old it is and creaky it is and how, unfortunately, it's another one of these things where it was never intended to be secure. It was intended to - it was originated to glue together separate cellular networks to provide like a call-forwarding technology from one network to the next. And among many things it lacks is strong authentication. There just isn't strong authentication as part of the SS7 protocol. And we all know that, if you can't be sure who you're talking to, then you can't trust anything they tell you. And you don't want to be giving them any secrets. So we've heard anecdotal reports of SS7 protocol compromise.

But I love that here was an affirmative, yes, just the other day we did this. The system is still broken. And again, it sort of puts a point, a little bit of punctuation on what I was saying a couple weeks before when I said I chose a time-based authenticator rather than SMS for exactly this reason. I don't want something secret constantly going through the cellular network every time I need to authenticate. That's just not - now, okay. Using it for password reset, there your window of vulnerability is going to be smaller. But on the other hand, if you've got somebody who's set up looking to suck all of the SMS messages that go by in, and they're seeing six-digit codes or something obviously that is meant to be secure, well, then that creates an opportunity for them to grab it, so not good.

PADRE: And you know, Steve, I've been telling my users for the longest time to use multifactor authentication. It's the standard model - something you know, something you have, something you are. And most of them have tried using some sort of, well, I get texted a password. I get texted a unique number that I can enter in, and that's my two-factor. And they get confused when I've told them don't do that because they think that's the something you have. You have your phone. But it's something you have that's receiving a transmission, therefore it kind of invalidates that part of the multifactor authentication equation. I have seen a couple of apps, though, that will use secure communications, non-text, over TCP/IP, encrypted, that I do trust. Would you trust that more than like a time-based RSA token? Or are you still going with a token?

Steve: It just occurred to me that one way you can explain this is this is not - if you use SMS, it's not something you have, it's something you blab.

PADRE: Right, exactly, yeah.

Steve: So not what you want. So to answer your question, the issue is always authentication. It always comes down to that. And this is something that Leo and I go around and around about because like he was all excited about Telegram for a while. And I said, well, yeah, but there's no authentication. He says, well, but it's secure. It's like, yeah, maybe. But if you don't know who you're talking to, if you don't know who it is secure to, then it could be a man in the middle.

And frankly, I'm even skeptical of systems like iMessage. I've talked about that. The problem is they're handling the keys for you. Yes, it's hugely convenient. But the tradeoff for that convenience is that, since Apple is providing the keys under which the messages

you send out are being encrypted, they could toss in an NSA key, and who would know? I'm not saying they are, but they could.

So messaging apps, for example, have to have, I mean, if you really care - and there's a spectrum of how much this really matters because remember, even if you have super-secure messaging with authentication and encryption, if you're using a phone which has been rooted so that there's something in there watching you send stuff before it's encrypted and capturing it after it's decrypted, outside of the point-to-point encryption, then you still don't have security.

So one of the things, sort of one of the concepts we've been looking at more in depth recently is sort of the fallacy of absolute security. So, yes, we want security. But unless we went out onto the beach and got some sand and extracted the silicon and then designed our own chip and wrote all of our own software, that to the degree that we're trusting anyone else, or as you started the podcast off with Trust No One, well, fact is it's totally impractical not to trust anyone. We have to. Otherwise we'd get nothing done.

PADRE: Right. I was actually just about to ask you that because, yes, if whoever's doing the authentication for you is compromised, you've got a problem. If the end device that you're using to receive your authentication is compromised, you've got a problem. But, I mean, even if you're using a time-based security, you still have a server that's running the time-based equation and then authenticating against your token. If that's compromised, then you've got a problem. At some point you have to assume, and I know that's a horrible, horrible thing to do in the security world, but you have to assume that some level of security is trustable. Right? And it can't just be you. It can't just be your equipment.

Steve: Correct.

PADRE: Something out there you have to trust.

Steve: Correct. And, for example, I've been deep into SQRL now for quite a while. What SQRL does is it is, as far as I know, the most minimal requirement for trust of anything we've got so far because it is only a compromise of the secret in your client which represents a vulnerability. The beauty with SQRL is that, because servers are dynamically challenging you to sign a secret, and all they have is a public key representing you, which is only valid for that domain name, you're not requiring them to keep any secrets.

So exactly in your example, Padre, they don't have a key which they're trying to keep secret which generates the one-time, the time-based token. And if that got loose, then anybody else would know what it was. So the vulnerability you point out is exactly correct. And SQRL doesn't have it. So ultimately what I think is going to happen is we'll end up with something like SQRL for YubiKey, where that one master secret which we have to keep secret ends up being in a little piece of easily managed hardware, and then it's just not possible for it to get loose. And you don't want to lose it, though, either.

PADRE: But that's the right way to think about it. I don't trust you; but I know that, even if you let all this out, it doesn't compromise me entirely.

Steve: Right.

PADRE: That's kind of - I think that's the best-case scenario.

Steve: Right. Sort of in terms of trust perimeter, SQRL has - or maybe diameter - has

the smallest trust diameter of any of these technologies that we've been talking about so far.

PADRE: Wait, did you just coin a new phrase? Can we TM that, trademark it?

Steve: I like that.

PADRE: Trust diameter. I like this.

Steve: The trust diameter, yeah.

PADRE: Hmm. Someone, oh, hold on, no, don't, don't [crosstalk] the domain.

Steve: Maybe radius. Trust radius?

PADRE: Trust, oh, trust radius, oh, that's even better. Trust radius.

Steve: That's better.

PADRE: How about a trust sphere?

Steve: Because you get a little radius in there, too. You want radius.

PADRE: Of course. All right. Yeah, you can't trust SMS. But you know what I can trust, Steve? I can trust my home security cameras. I know they're rock-solid, designed from the ground up to keep me safe and not let anyone else into the most private parts of my life.

Steve: Okay. So the timing of this could not have been better. The IoT Infancy - oh, by the way, we did coin an acronym there, and that was I Don't IoT, which of course is I-D-I-O-T.

PADRE: We used to call that an ID-10-T error. When you had a user who just wasn't there, and you needed to convince them that it wasn't their fault, you said, "Oh, yeah, see this all time, it's an ID-10-T."

Steve: That's good. So the original IoT infancy podcast was going to be about baby monitors. But I ended up breaking it up into two podcasts because there was just too much to talk about, and the details were just so juicy of the ridiculous lack of security that these technologies had, things like the URL that you used to go to a website to view your baby cam on the 'Net. The URL had a serial number, and you could just change some digits and view other people's babies. It was just, I mean, just horrifying lack of security.

So then when this popped up on Reddit, I thought, oh, this is wonderful. So the title of this article on Reddit was "I bought and returned a set of WiFi-connected home security cameras. Forgot to delete my account, and I can now watch the new owner." So this guy, oh, it's just horrifying. This guy wrote, he writes: "A few months back I purchased a Netgear Arlo home security camera set. I set up an online account, connected the cameras, tried them out for a few days, and ultimately changed my mind. They were returned to the store, and I never gave it another thought - until today.

"I got a random email alerting me that the camera had detected motion, but I don't have

any cameras. So I logged into my online account, and I can see the new owner, their house, and everything they're doing. Netgear obviously doesn't have a system in place to prevent cameras on multiple accounts. If I'm not mistaken, anyone could get the serial number off your cameras and link them to their online account, to watch and record every move without your permission. A creepier dream. Does anyone else see this as a serious security flaw on Netgear's behalf?" And then he says: "I'm even happier that I returned them now."

And then in a subsequent edit he said: "I left a message with Netgear this morning," which was yesterday, 6/20. He said: "Received a return call from a Senior Support Engineer saying they're aware of this issue. Since the cameras aren't supposed to be resold," he says, "I suppose they didn't think it would be an issue." What, are you supposed to destroy them and get credit? Anyway, he says: "I was assured they were working on a fix within the next three weeks to prevent cameras on multiple accounts and force a hard reset on the cameras, if cameras were previously registered in the system." Oh, and I should note that I tweeted this earlier, and several followers said, "You know, I have those cameras, and the same thing happened. You just plug them in, and they don't ask any questions. Everything just works." And, oh.

PADRE: Here's the sad thing, Steve. This is not just a Netgear problem. I have seen this on so many cameras where the only thing that protects you is either a registration number that they've created and put into the firmware, or more likely it's the MAC address. You have to know the MAC address. And I've seen way too many supposedly secure products that use a MAC address as a layer of security, which it was never designed to be.

Steve: No, you're right.

PADRE: In fact, JammerB, if you go back to the screen real quick, this is an actual - this is posted on that Reddit page. So if people want to go and find out what this owner is doing right now, they could just use that. The information that's on this label right now you could use to get into that camera. In fact, you could go down to your local Best Buy and take pictures of all of these tags and then just wait for them to get bought. And unfortunately I've seen this, and I won't name any of the other manufacturers, but a lot of the other companies that compete in the same space as Netgear are doing the same low-cost authentication, which is - it's sad.

Steve: Yeah. So this returns to the theme of IoT infancy. The only way to regard this is, first of all, we're in an incredibly early stage where companies are rushing to get their products on the market. And I'm sure that the front of the box says "Secure encrypted webcam system." I'm sure that's a bullet point. And here's the consequence. I mean, here's the reality of it is that the architecture offers none of the secure encryption. Maybe the connection is. But obviously anybody who's able to get on the website and see what's going on in the new owner's home, oh, wow.

PADRE: You know what I think, when I think about this, I think of the GM hack that they did. Was it Black Hat and Defcon last year? I think it was last year where they found out that the only security was the IP address range. So they just thought no one's going to guess the IP address. Well, I mean, once you had one device, you knew all the other devices are probably going to be near that, so you just kept scanning it until something responded. And it's this sort of lazy security that lets people think that, well, I paid a lot of money for this device. Therefore, it must be secure, which is completely not the case.

Steve: In fact, that's exactly the way ShieldsUP! first got created because I had the first, back in the early days, the first - it was a DSL or IDSL connection for GRC. And I don't

remember if we had multiple IPs or just one, and maybe an early NAT router. But I had a public Internet Protocol address, an IP address. And I thought, I wonder what's in the neighborhood? And so I scanned, like, plus or minus a hundred IPs and found all these C: drives, all these other Windows machines with their C: drives exposed. And it was like, oh, my lord. Someone's got to do - someone's got to raise the flag about this. So of course I created ShieldsUP!. And, boy, in the early days of ShieldsUP! people went there, and I showed them their C: drive. You could browse your own C: directory on my web page. It was just horrifying back then. So there was an example of another aspect of infancy.

That was the infancy of Windows machines' first contact to the Internet, and that was the reality. Now, as a consequence, all ISPs now block ports 137 through 139 and 440 - is it 445 or 443? I always get those confused.

PADRE: 443?

Steve: One is secure email.

PADRE: Right, 443.

Steve: I think it's 445, and 443 is email.

PADRE: Okay, thank you.

Steve: Anyway, so all ISPs block that in order to protect their own clients, who may still somehow have a drive on the 'Net not behind NAT, not behind a software firewall, just exposed and flapping in the breeze. But anyway, so the good news is there's hope. We are at this stage now with webcams and home security systems and so forth. We were once at that stage with people's hard drives on their computers. And we're finally, it took a long time, but now we're secure.

Unfortunately, I think this is going to be similar escapades for a long time on IoT, until we finally get some standards. I think what we're going to need is some sort of standards body to come in and establish the way this stuff has to work and come up with a protocol. Because right now it's the Wild West. Every manufacturer just does whatever they want to, as little as they want to, and then stamps "It's secure" on the box. And people at Kmart say, "Oh, let's get some cameras," and stick it in their cart, and off they go, and now everybody else can see what they're doing.

PADRE: I remember a few years ago it was Access Communications, which sold high-end cameras, I mean, way more expensive than the ones you would buy at a consumer Best Buy. But unfortunately, the login page was searchable, so Google could index it. And you could look for a particular page that would bypass security. So you would just do a Google search for this one page name, and it would list all the cameras, the Access cameras, that could be bypassed, just by clicking on the link. It was a little bit scary.

But Steve, it's funny because all of these zero-configuration devices - and again, Netgear, D-Link, think of the lower cost hardware. Most of them are built off of a singular reference design. I mean, if you crack them open, they vary very little on the inside. And the reference design was created, along with its firmware, for you to improve upon. And some of these businesses have not improved upon it. All they've done is they've added their branding. And then they add a password screen, which is really just a couple of lines of code that compare one string against another string. And that's it.

And I'd love to get your input on this. We've had talks at the last three CESes, the last two Black Hats, the last two Defcons, from high-level officials in the U.S. government who are saying, look, we understand that you're in a rush to get your creations out into the market, but security can no longer be an afterthought. Because that's what it is right now. It's get it working, and then I'll think about security later. Well, unfortunately, it stops at the "get it working" part, and security is tacked on as an afterthought. How do we change that? Because it's so difficult for some people to think security first.

Steve: Say, for example, and while you were talking I was trying to come up with a way, something that would be foolproof, that would allow a device to know if it should randomize something in itself. And so, for example, imagine that, okay, because we know there's a lot of technology in these things. Often there's a little Linux microkernel. They've got firmware. They've got nonvolatile storage.

Imagine if, when power comes up on this thing, it reaches out to well-known NTP servers and checks the date. And that allows it to know autonomously how long it's been powered off. And if it's been powered off more than some period of time, like 24 hours, it has to be repaired in some fashion. It just decides, okay, I'm no longer going to trust my own current configuration. The user's going to have to do something in order to reassociate the camera with their account, something like that, so that it can still be simple, yet it can protect users from themselves because that's what they need.

PADRE: Right. I want to bring up something from the chatroom real quick because [pschops] brings up a good point, something that a lot of us think, which is, and I quote: "If only companies were liable for lapses in security, things would get straight pretty quickly." I understand that. I understand that sentiment. I feel that way sometimes. However, there is the flipside to it, which is you want to reward companies for being honest. And if a company thinks it's going to get dinged for a security lapse, they are much more likely to sit on a potential security hole, rather than announce it immediately.

Steve: Ah, good point, yes.

PADRE: So many of the best policy examples I've seen have been, if you announce a security hole within the first 10 days of you learning about it, then you get sort of a blanket immunity because you were trying to do the right thing.

Steve: Or how about if we add to that concept, if they open source the software, then they're also let off the hook.

PADRE: Right. Precisely. Yeah, if you allow it to be vetted...

Steve: [Crosstalk] the ability.

PADRE: And that's, I think, because we can talk about security vulnerabilities all day, and they're fun because we get very creative when we look at ways of getting around authentication.

Steve: It's a target-rich environment.

PADRE: It is a very target-rich environment. But ultimately, as security professionals, we want our fellow professionals to have a reason to do the right thing.

Steve: So...

PADRE: What's next?

Steve: So this story, I don't really know what the truth is because GoToMyPC - and they've been a sponsor of the TWiT Network through the years. I don't know if they are still today. But of course their parent company, Citrix, has been a longtime sponsor. Again, this felt like clickbait. But it looked like it was clickbait, well, it looked like it was a problem maybe of their own initial disclosure. So the headline read: "After suffering a 'very sophisticated' attack," which is their words, "GoToMyPC forces all users to reset their passwords."

So of course a lot of people use GoToMyPC. I imagine a lot of our listeners do because they've been an advertiser on the TWiT Network for a long time. So everybody knows that they would have gotten a password reset email; or, when they have attempted to use GoToMyPC, they then were told you must reset your password in order to proceed. So what's weird is that - oh, and in the context of maybe there being a really bad breach of all their passwords, it occurred to me that they didn't mean GoToMyPC quite as literally as it was perhaps now being used.

But what they wrote in an incident report under status.gotomypc.com was: "Dear Valued Customer" - now, this is their own disclosure. "Unfortunately, the GoToMyPC service has been targeted by a very sophisticated password attack." It's like, whoa. And of course the good news it wasn't a junior attack because then that would be embarrassing. It's a very sophisticated attack. This is big guns were required. And continuing: "To protect you, the security team recommended that we reset all customer passwords immediately. Effective immediately, you will be required to reset your GoToMyPC password before you can log in again."

And then further down in this status report of incidents they said: "We have experienced" - it was under Investigating. "We have experienced an issue which requires you to reset your password if you are having trouble logging into your account." Okay. "Please reset your password through the Forgot Password link if you are having trouble logging into your account." Okay, so that seems a little inconsistent with what was said at the beginning of the same page.

But then subsequently they say: "John Bennett, product line director at Citrix, said that once the company learned about the attack, it took immediate action. But contrary to previous published reports" - and it was their own published report. Apparently he says: "There is no indication Citrix or its platforms have been compromised." He said: "Citrix can confirm the recent incident was a password reuse attack, where attackers used usernames and passwords leaked from other websites to access the accounts of GoToMyPC users." So not so very sophisticated after all.

PADRE: Steve, what you don't understand is it is so sophisticated that not even they understand what's been going on. So that's how - yeah, crazy sophisticated, really.

Steve: Yeah, so then he concludes his emailed statement, saying: "At this time, the response includes a mandatory password reset for all GoToMyPC users." And so I sort of - I pushed back from this; and I thought, okay, wait a minute. Now we're at a state where, if other companies suffer major publicly disclosed breaches, all the other companies on the Internet have to force all of their users to reset all of their passwords, too.

PADRE: Yeah.

Steve: Sad state of affairs.

PADRE: It's a sad state. But, I mean, if you look at the Next Steps page that Citrix put up, it's most of the same things that we've been saying for years. Don't use words from the dictionary. Select strong passwords that can't be guessed, eight characters or more. Make it complex with random capital letters, punctuation, or symbols. Then do the whole substitution, zero for "o," three for "e," two-step verification. So the problem is people read this, and they go, "Yeah, yeah, yeah, yeah." And then they promptly reuse that password on another site. And that's human nature. We can't change that.

Steve: Yeah.

PADRE: Oh, my. And actually, this is the second high-profile case of a potential breach that was caused because some other site lost their passwords. Twitter just suffered this. People said that, well, there's a lot of Twitter users being attacked. And Twitter was actually able to look at the accounts that were affected. They found out - because they segment their authentication databases. And they were saying, like, a few here and few here and a few here. So they said, look, they didn't get the database because, if they did, you'd have contiguous blocks of users.

Steve: Ah.

PADRE: This is obviously a case that someone has been compiling these passwords, and then they just threw them up against Twitter to see which ones would work.

Steve: Right. Well, and of course we talked about how Zuckerberg was one of the people who got compromised.

PADRE: Right, right.

Steve: With the rather embarrassing password "dadada." It was just D-A-D-A-D. So it was like, oh, okay. But it was from a 2012, was it, or even older, it was from an old Twitter account that he hadn't used. It had been just sitting idle for a long time. But, oh, I am sorry, it was from an old LinkedIn account, and we believe it was part of the LinkedIn breach. And then they tried to reuse that because he's a high-profile person, and they were able to compromise his Twitter account with it.

PADRE: Right.

Steve: Okay. So this one was the most tweeted story, with lots of people saying, Steve, can't wait to get your take on this on this week's or next week's, depending upon when they sent this, podcast. And this, again, was - this was an article on BoingBoing that was really over the top, with the headline "Intel x86s" - and actually it's not x86s because those chips, if they're only x86s, they don't have this. It's newer processors, so it's going to be x64s. And it's not actually in the CPU, either, it's in the chipset, it's in one of the outboard chips - "hide another CPU that can take over your machine." And then "(you can't audit it)." So this is a guy, Damien Zammit, who has sort of made this his personal crusade. And the problem is this generated a great deal of concern. And it's not that some concern isn't warranted. What I wish is that there was a little switch on the motherboard where you could turn this off. And that's what we're lacking.

So here's the background. So, okay, first of all, the story starts out: "Recent Intel x86 processors implement a secret, powerful, control mechanism that runs on a separate chip, that no one is allowed to audit or examine. When these are eventually

compromised," as he puts it, "they'll expose all affected systems to nearly unkillable, undetectable rootkit attacks. I've," writes Damien, "made it my mission to open up this system and make free, open replacements before it's too late."

Now, first of all, good luck with that, Damien. I'm going to explain why you're going to have to have some serious voodoo powers in order to make that dream come true. So here's what's going on. First of all, there is sort of an acronym stew. We have something known as the Intel Management Engine, sometimes referred to as IME, or sometimes ME. And then this outboard thing is known as the ARC, the A-R-C processor, which is a 32-bit RISC chip. So it's not like another x86 Intel, well, it is an Intel processor. They produce a chipset. But it's not Intel instructions. It's a RISC processor.

Then there's something called AMT, Active Management Technology, which IME implements. And this all replaces a previous IPMI, which is the Intelligent Platform Management Interface. So Intel's intention here is to create a - we could call it sort of base band. It's on the motherboard. It is built in. It is, where present, it is ubiquitous, meaning that users don't have any control. BIOS doesn't have any control. You can't turn it off.

When I was setting up my big new box a couple months ago where I put Windows 7 in it, and it was a Haswell chip, and I wanted to get that because of the news that future hardware platforms would not necessarily be backward compatible to Windows 7, that essentially Microsoft was saying we're not going to keep making our newer - Microsoft was saying we're not necessarily going to be providing drivers for newer hardware on our older OSes. So I said, okay, I can't wait for my current system to die. I need to get one now that I like a lot, that I'll be able to run Windows 7 on it, because I never want to have to go further than Windows 7. And I made the comment that I had removed the Intel Management Engine. What I had done was I had uninstalled the Windows drivers for it, where it allows Windows to interface with this. But it can't itself be disabled or turned off.

So the concern is, my concern is that not only is this thing responsible for setting the bus clocks, it's a processor that gets things going. It sets the various counters on the master clocks that run the bus that starts the system going so that then the big expensive Intel x86 or 64 or whatever processor that you've got is able to come to life. So it's sort of the pre-life, get things going processor.

PADRE: Steve?

Steve: The really annoying thing is that it has deliberate access to the motherboard's network interfaces.

PADRE: That was the question I was going to ask, yeah.

Steve: Yes. And that's deeply disturbing. Now, apparently this is what Intel - this is technology Intel is providing for corporate enterprise-level management, the idea being that using some ports at the enterprise level, regardless of whether your computer is on or not - and that's the other creepy thing. And I'm sure anyone who's been using or who's looked inside their machines anytime in the last 10 years or so, even when the machine is off, sometimes you look in, and there's a little LED staring back at you, glowing, on the motherboard. And it's like, wait. It's off. Fans aren't spinning. Everything's quiet. Yet there's a little green LED. Or even worse, if you look around the back, if there are lights on the network connector, they're sometimes on and flashing. Like something is alive in here, even though all is quiet; no heat's being generated; no fans are being spun.

And so that's this thing. That's the Intel Management Engine running and apparently listening on the network and able to respond affirmatively, even with the computer off. And so that's what's got Damien all worked up here is that what if Intel has made a mistake? What if their code is not perfect? And as he says, "when these are eventually compromised," dot dot dot. "They'll expose all affected systems to nearly unkillable, undetectable rootkit attacks."

And in fact it is the case that this Management Engine technology has, while the system is running, complete access to everything, unrestricted access to the system's main memory so that it can, if it were compromised, snoop. And we don't even know how it works, what it does. It is completely closed by Intel. It is not documented. It is, essentially, it is as secret as they're able to make it. And the people who've been looking at this consider this Ring -3, like the deepest, darkest Ring level of access, like what was it, some rings of - I'm trying to think of...

PADRE: Rings of hell, actually.

Steve: The seventh ring, yeah, the seventh ring of hell or something. So everyone's familiar with the notion probably of Ring 0. We talk about that a lot because that's where the kernel typically operates. This notion of rings is the privilege at which the instructions are executing at that time on the processor. And the Intel hardware supports four rings. So it's got two bits, two binary bits; so there's four different possibilities. So that gives us Ring 0, Ring 1, Ring 2, and Ring 3. And the original architects said, yeah, that ought to be enough. Well, it turns out that two rings was enough. They could have spared themselves a bit and just had one bit because nobody uses Ring 1 and 2. You're either in Ring 3, which I userland, or you're in Ring 0, which is the kernel.

So there is this notion of a ring transition where you go back and forth. And that's something that, due to the Intel architecture, is a little bit painful in terms of overhead, which is, for example, why Microsoft, in their less than wonderful wisdom, moved GDI, the Graphics Device Interface, from Ring 3 down into Ring 0 because there are so many calls to GDI by Windows that it was expensive to have it up in Ring 3. And so they made the mistake in retrospect of putting it in the kernel. And that then allowed things like JPEGs to take over your computer because the graphics device interface would be interpreting JPEGs, and they made mistakes, and that allowed your computer to get taken over.

So in this ring terminology we have the positive rings, Ring 3, and we're not using 1 and 2, and then 0. And then people consider hypervisors, which run underneath multiple Ring 0 kernels, they consider that to be Ring -1. And then there's the SMM, the System Management Mode. That's considered Ring -2 because that's even deeper than the hypervisor. And that's where we get to Ring -3, which is sort of what this is considered, this whole management engine thing, because it's arguably deep - it's not system management mode. It's even deeper than that, where it really provides the low-level control over the whole system and, unnervingly, is also monitoring the network.

Now, Intel has done arguably everything modern design permits them to do. I mean, they understand they don't want anybody messing with this. So first of all, it cannot be disabled on systems running the Core 2 series processors. It is kept absolutely secret. Damien notes that there's no way for the main CPU - there's total visibility from it, from this IME, like so-called Ring 3, up into the higher rings, but none in the reverse. There's no way for the main CPU to detect whether this management engine may have been compromised - no way for it to access it at all, no way for it to repair a compromised management engine, no way to detect if malicious entities have been able to compromise

it or infect it.

On the other hand, Intel has done everything to lock it down, everything possible. There's a public key in the hardware, not even in the firmware. I mean, there's a public key in the silicon of this management engine. And there is an SHA-256 hash which is taken of the public key to verify that it hasn't been changed. And maybe the public key is in the firmware, and the hash is in the silicon. Again, details are sketchy because Intel doesn't want to share any of this. The only thing we know is from deep reverse-engineering. And again, it's like almost pop the lid on the chip in order to reverse-engineer this. There's little known about it.

But so the public key is verified with an SHA-256 hash in a way that firmware can't change it. Some components are in the silicon at manufacture. Then the signature of what is in the firmware, because it is firmware updateable, which is a concern, the signature of the firmware is verified using that public key. So we know that only Intel will have the matching private key to allow them to create firmware that this IME will ever consider running. And that firmware image has custom compression which the hardware decompresses. It's not even decompressed through any known algorithm. And there's, I guess, 11 versions of it which have subtly changed over time.

So that part has been reverse-engineered. The people have extracted the compressed firmware and arranged to decompress it into the RISC processor's instructions. But it's not possible to change even a single bit of that, even knowing what the compression algorithm is, because then that ARC, that 32-bit ARC RISC processor will not run it when the system is first plugged into the wall, and that little green LED on the motherboard first turns on, and all of this begins to come alive.

So they've done everything they can. They've locked it down utterly. Nobody can change a bit. People - and this is why I say Damien wants to come up with a public domain open source version? Well, okay, who's going to sign it? Intel's not going to sign that. And if Intel doesn't sign it, this hardware will not run it. I mean, maybe you can get in there with your soldering iron and fuse the silicon. But it's not going to be a widely applicable, download this updated IME firmware sort of thing. And believe me, I mean, it is a concern that this thing is on our network interfaces. And apparently independent of IP address or MAC address or anything else, it's there sniffing traffic coming in and out. But it comes along with the hardware.

So again, it's like, yes, it's a little annoying. But all you can do is pull the plug. And I mean literally the plug, either the power plug or the network plug, out of the back of your machine. If both the power plug is plugged in, and the network is there, there's a link up, and this thing is waiting to receive instructions from the mothership.

PADRE: Now, as an enterprise guy, I understand why they built this, because...

Steve: Good, tell us.

PADRE: ...we've been dealing with out-of-band management forever. That's how I've always been taught to build. So you have the main network, and then you always have a way to get into your devices, even when the main network is broken or not working properly. So this is a way for you to get management that bypasses even the hypervisor inside of an Intel chip. So especially if you're running a server that's running a lot of VMs, or you're running it as a container box, this gives you the ability to get in even below that in case something goes tragically wrong. So rather than walking to the box, you actually have the ability to get in there and not just power cycle, because we can already do that with power distribution units, but to actually figure out what state a computer is

in. So that's, from the enterprise guy, that's incredibly fantastic. I love this solution.

However, and I think this is the same concern you have, the problem I have is that it has no audit. There's no way to find out what's going on inside. I would never let that on my network because it means, well, I now have an unknown, an always unknown on the network. I just have to assume that no one has broken their way into one of these processors. Now, as you describe, it is a very difficult, complex technical task to do so. Right now I can't think of a way to do it. But we've seen in the past that hardware security protection, where it's baked in, is always the most secure until the first time that it's broken, and then it's absolutely worthless and a huge security risk.

Steve: Okay. So now we have this 32-bit ARC RISC processor, and these guys have figured out, they reverse-engineered the hardware-based decompressor. So they're now able - and then through the 11 versions of this, so obviously Intel trying to change it. That's not going to help. I mean, it may be the case that nobody can change the firmware, no third party can change the firmware. But now look what we've got. Now we've got a group that are figuring out what the code is. We know that, once you know what the code is, you can then - you disassemble that and come up with the, what is this chip doing with the NIC? What is it doing on the network?

My point is, if it has preemptive control over the system, that is, if it's not just read-only, if it's not just a monitoring tool, I mean, that's bad enough, depending upon how deep and what it monitors. But what this means is that third parties will know everything Intel knows about this thing's capabilities. They will reverse-engineer the language. They will reverse-engineer the instruction set, that is, the high-level, network-level instructions that this thing uses for communicating. So now you've got the possibility that something malicious that knows this gets onto a large enterprise network and has the free rein of the place, even at 2:00 a.m. when all the computers are off. It just seems like a bad idea.

PADRE: It does.

Steve: To have something that is in - we are in the process of reverse-engineering it. So anything Intel has enabled that to do, we will know. And if anyone knows it, everyone knows it, just exactly as you said. And it just seems troublesome.

PADRE: I would actually like to get one of these processors, so a processor that I can confirm has this built into it, and then run it on an OS that I strip out all network controls. So you take out all the stacks, so it doesn't have the ability to communicate with the network adapter, and then just sit on the line and listen. I want to see if this is actually talking to something, rather than just waiting, because I believe the article mentioned that, in order for this to be useful, there has to be some sort of TCP/IP service running on that little processor. There has to be...

Steve: Yes, in fact the article did say what's most creepy is that it runs a full TCP/IP server on the network interface, and packets entering and leaving the machine on certain ports bypass any firewall running on the system. Of course, because it's hardware. The firewall is in your OS. And so really the only way to protect yourself would be to use an outboard physical firewall that is drop all, and then where you specifically allow your own higher layer traffic to get through.

PADRE: At the very least this could be a couple of interesting weekends, just sniffing around. Actually, I'd sniff first. I'm betting it's probably not sending out any traffic. It's probably just waiting. But then I'd start bombarding it with traffic, just to see what it listens to. And that is kind of fascinating. I look forward to seeing how this develops over

the next couple of months.

Steve: So there's the behavioral approach, which you talk about. And if you can get the firmware, and you can decompress it, and you can decompile it or disassemble it, then you just work your way through it, and you end up with a complete lexicon of exactly what this thing does. And so at some point we're going to know. And, I mean, maybe it's already known. Who knows? I mean, again, it's a little annoying that this thing is sitting in our machines. I just hope, I mean, so, for example, is it Ethernet level? It says TCP/IP, which presumes that it's got a TCP/IP stack. And TCP/IP, as we know, is Internet protocol, so it's routable. So what IP does it use? Yeah, it really, I mean, it raises a lot of really intriguing questions.

PADRE: Is there an easy way for me to find out which processors are affected?

Steve: There's more detail in the story. There was something called a vPro, and I think it has to be vPro processors, and non-vPro processors are not.

PADRE: I got it. Okay. But a recent vPro processor.

Steve: Yes. Well, because older ones had the previous, the IM whatever that was, the older technology. Intel's been doing this for a while. They've been up to this. The IPMI was the Intel - the Intelligent Platform Management Interface was their earlier one. And I think I've got servers, I think my servers at Level 3, those are from '04, and they've got that in it.

So again, there's 12 years ago Intel was like, oh, we've got to create this. So, I mean, you know, maybe if it's for inventory management, like the corporation is able to scan their network and get - because I know that there's like all kinds of funky serial numbers and things in the BIOS of these older servers that I have, where there are FRUs and all kinds of strange stuff that I never bothered with because it wasn't about running my stuff. But it seemed that there was a lot of enterprise-class management stuff. And that's apparently what Intel's doing. But the question then is who else knows about this? This isn't of value if Intel is the only one who knows. It must be that they provide some enterprise-level interconnectivity that allows corporations to do something with their fleets of machines that are all enabled with this.

PADRE: Right. But, I mean, like for asset management, which is one of the biggest things that we have to do in enterprise, I can do that just by looking at the NIC because all NICs will stay in low power state as long as the machine is still plugged into power somewhere, which will allow me to see if it's still connected to my network. And I can even get a warning if a device suddenly is removed from the network. So I don't need a second processor to do that.

Steve: Right, right.

PADRE: Which, I mean, again, I don't think it's a nefarious thing. I think it's something they thought was a feature. Maybe they didn't think it all the way through. But like you, I would love for Intel to come out and say, okay, this is exactly what it does. This is how we access it. This is how it's secured. And this is why we think it's still a useful feature for enterprise.

Steve: And why can't I, as the owner of my system, turn it off?

PADRE: Exactly.

Steve: Or at least turn off the COM part. Turn off the NIC interface. Like go set up the bus, get everything going, but stop, you know, don't even think about talking on my network interface without my permission.

PADRE: Steve, we've been talking about this a lot, but I could see this as becoming a thing past Intel. More and more of the devices that we use are always on, always connected. And I could see real benefit to having a management layer of hardware that I as an IT person could access that nobody else could. I'm actually pretty firm in my belief that this is going to start spreading to other product categories. So when it does, is there a safe way to do this?

Steve: The problem we have with the whole IoT concern, the reason it causes so much trouble, is the tradeoff between ease of use and security. Obviously, what Netgear was doing with Arlo made it extremely simple for their users to hook up their webcam systems. Look, oh, just plug it in, and go to the website, and it finds you. So what we're going to have to come up with is in general this Internet of connected things is some sort of compromise solution that gives us security and also ease of use. We don't have it yet.

PADRE: Oh, is that all. Oh, so I just want it secure, but I also want to make it dead simple to use.

Steve: That's right.

PADRE: That's the cry of every security professional from the dawn of security.

Steve: So just a couple last things. I did get a nice note from a Corey Grant, who's in Livingston, Texas, where the subject was "Testimony." And I thought, what? And he kind of meant testimonial, but thank you, Corey. He said: "Greetings. I'm a network engineer in rural East Texas and moonlight on the side for a few local businesses. I got a distress call about a PC that would not boot." And of course we all know where this is going. "This machine had lots of tax data from previous years that" - and he just wrote "she," meaning the person who called him, I guess - "was actively using for research."

He says: "I have used SpinRite many times to increase performance of lagging desktops, but this was the first time it actually resurrected a dead machine. She is happy, and I am a hero, thanks to you." So Corey, I really appreciate you sharing that with me and letting me share it with our listeners. So another SpinRite brings the drive, and in this case apparently years and years of tax data, back from the grave.

PADRE: Those stories aren't even really surprising anymore. We've heard them so often that we know. Actually, I got...

Steve: Yeah, yeah, yeah, another - so it's like, how do I make this more dramatic? Yeah, yeah, yeah, SpinRite fixed the drive, okay, fine.

PADRE: Well, I've got one for you.

Steve: Oh.

PADRE: TWiT.tv was covering the Electronic Entertainment Exposition down in Los Angeles at the Staples Center. And I was at a booth. I won't say which booth. I will just say it's a company that makes operating systems as well as gaming consoles. But we got an invite into the lounge, this sort of second-floor type thing. And there were a couple of workstations up there that were controlling some of the major displays that are going on

around the booth. And one of them was down. And as we were recording, I kept sort of glancing over. And then I realized it was running SpinRite, and they were running SpinRite on the internal SSD. I guess they had had some issues right before the show started. And then by the time my interview was done, it was booted back up. So I'm like, oh, okay, well, evidently this company that makes operating systems and game consoles has gotten the word that this is the software to use.

Steve: Interesting. Very nice.

PADRE: So you saved E3. Steve Gibson saved E3.

Steve: Well, and what's funny is that we've heard anecdotally that companies that are professional data recovery firms, they use SpinRite as the first thing they do because most customers don't know, and the bill is going to be often north of a thousand dollars, right, for like a professional data recovery firm. But they don't want to put their good guys on it if SpinRite'll fix it. So they just run SpinRite. And they still charge the customer an arm and a leg, but they didn't have to expend the expensive heavy-gun resources to do platter change or PC board swap-out or all the things that can be required if, well, if the hardware's really dead, and then SpinRite can't do anything to fix it.

PADRE: All I know is that it's been in my toolkit for the last two decades, just about.

Steve: Yes. Thank you.

PADRE: So if it's a hard drive, if it's an SSD, there's a reason for you to have SpinRite, folks. And that reason is Steve Gibson. Shall we get into some miscellany?

Steve: Yes. Well, I just have one piece. This is something I've mentioned before, but it was triggered by somebody tweeting me and didn't remember what it was. He said, what was that thing you recommended years ago for archiving all your email? And what happened was there was some event, I think it was just that I had - I'm still using old Eudora v7.1, which keeps on going. I like the UI. I've tried other email clients. They just - they're not the same. So I've stayed with it.

But I realized Eudora was having a horrible problem with a million years, or at least two decades, worth of email that I was just dragging along behind it. So I went looking for a solution, and I found a free solution. It is still free. The company's called MailStore, and it's called MailStore Home. Now, I was using v8, and they're now at 9.7.1. And it is free for personal use. And so I wanted to renew my recommendation. I'm still using it. I still love it. And it is amazing.

It now has 2.5GB of my past email archived. I can put in - in fact, I used it to find you, Robert, a couple days ago, when I first sent you mail, or I guess it was last week when I knew that you were going to be my co-host. I didn't still have you in Eudora. So I fired up MailStore, and I put in "Father Robert," and bang, it found all of our previous communication, and then I grabbed your email address and addressed a piece of email. So it is itself a pop3 and/or IMAP client, so you can just aim it at your server, and it'll suck things in and index them. It can be a central archive for all your email. It can do Gmail, Yahoo Mail. It knows about all versions of Microsoft Outlook, Windows Mail, Windows Live Mail, Exchange Server, Office through 365, Mozilla Thunderbird, Sea Monkey, also PST, EML, and other files. It can suck those in, as well. And it runs perfectly next to my creaky old Eudora v7.1.

It's funny, too, because when I fired it up, it noted that there was a new version. Or

maybe I checked. I don't remember. But it's like, oh, yeah, there's a v9.7.1. And so I thought, okay, fine. So I tried to update, but it said no, not compatible with your OS. So it's like, oh, okay. So I'm using 8. And I did note, when I went to their site in order to bring it up - by the way, it's just MailStore.com, M-A-I-L-S-T-O-R-E dot com - that 9.7.1 is Windows 7, 8, and 10. Of course I'm still on XP, so I'm going to stay with 8, which works perfectly. So anyway, I just wanted to remind people about it because I did get a lot of positive feedback from those who adopted it after I first mentioned it. So I thought it was useful to say, hey, I'm still using it. I'm still loving it. And it's hard to beat the price.

PADRE: I still have a few old versions of Eudora that will work. They actually connect and work just fine. But I'm doing it the more advanced way now, Steve. I have about 30GB worth of old Outlook PST files just sitting on a SkyDrive, a OneDrive somewhere. Not quite as efficient, really. Go figure. So MailStore? That's my new joint? I'm going to be doing that one?

Steve: I think you ought to check it out because you could feed it those PST files. It'll suck them in and index them. And then you can really give them the Deep Six. Maybe keep them around. But this gives you total access to their contents. And as I said, I don't have as much as you. I'm at 2.5GB. But again, instantaneous index keyword search of all of my back email, which is really handy.

PADRE: I have a lot of attachments. A lot of attachments. Now, actually, let me ask you about that. Because my old policy, back when I was still using Eudora, was nothing gets deleted. Everything gets saved. I might refer to it later on. But it's gotten to the sheer volume of mail, and it's not all spam, some of it's actually decent, means I delete probably about 96, 97% of it, and I save about 3%. What's that ratio for you?

Steve: I don't keep any attachments, but I do keep all the text. Well, because I've just got it set up as automated now. It's just it's all happening in the background. Any mail to me ends up being archived. So from Eudora I just delete things whenever I want to. I have no problem at all now with just wiping things out. Like sometimes, for example, I lost a few months ago a neighbor whom I met because we were both on the association board, and he died. He was in his late '70s. And so I sorted by name and marked them all and just deleted them because there was just no point in keeping it around. But I did so knowing that, if anything ever came up in the future where I needed to refer to some conversation that I'd had with Leonard, it was in MailStore, and I'd be able to grab it. But there was just no need to keep it online in Eudora because at some point Eudora does seem to get a little choked up when things get too big.

PADRE: I just checked. JammerB, if you look at the other screen, I'm really close to zero inbox, on the lower left-hand corner. I've only got 1,516 items in the main inbox. So I'm pretty good, actually. This is a good day for me. Oh, no.

Steve: No, I'm the same way. There's just too much incoming from all different directions. And so I deal with what I can.

PADRE: All right. Now let's get into the real meat. And it was fun talking about breaches, fun talking about SMS. It was fun talking about Intel's little hidden processor. But now we need to learn all about control-flow enforcement technology. What is that, Steve?

Steve: So I'm excited about this because this represents an evolution of - a simple and compatible evolution of the architecture of the Intel processors to directly address two of the biggest problems that we currently have with what hackers are able to do to abuse code that has errors. We're closing in on the end of Year 11 of this podcast. And for the

entire duration of that time, we've been talking about the famous buffer overrun errors. And then also more recently the so-called ROP (Return-Oriented Programming) hacks where - and in fact I'm sure it was a podcast you and I were doing. We were talking about an Adobe Flash exploit. And they used a return-oriented programming hack in order to run some code in order to get loose of their containment and run some code that was in Flash that allowed them then to essentially bootstrap themselves to full system privileges.

Okay. So Intel has added, or will be adding, two new features to all future processors. There's something called - they're adding a new instruction called the ENDBRANCH instruction, and a new feature of the architecture known as the "shadow stack." And of course, again, these problems have been plaguing the industry for, well, forever. And I remember talking about once Steve Ballmer had a meltdown during some sort of Microsoft security summit - I don't think it was a public event, but it ended up being public - where he was ranting, asking how it was - I think it was after XP's launch because of course he was very vocal in talking about how XP was going to be absolutely the most secure operating system ever. And of course it was after XP and due to XP that we had Code Red and Nimda and, I mean, those were really the last highly prevalent worms that got loose on the Internet because Windows XP was such a catastrophe. And so he was famously ranting, asking how can it be possible that we're still having these problems? He just didn't understand.

And of course what we've been talking about recently is that older software, which you leave alone, that you find the problems in, but you resist touching, ends up being more secure than newer software, even if the newer software is implementing new security features, because the software itself is the problem, rather than the features that it's trying to offer. What we find is, in the real world, any new code generally has problems. And so what that argues for is, if it's not broke, don't fix it. Leave it alone.

So I got a big kick out of this because I talked about last week how this will be the subject of this week's podcast. And I got a tweet from somebody from Intel, his name is Steve Fintel. I guess that's his real name. Maybe it's Steve F. at Intel. I don't know. But anyway, he tweeted me, he said: "I see your Security Now! topic for next week is Intel's CET. I," writes Steve, "was the Atom Processor CPU planner when we got approval to integrate CET into Atom, ahead of Core on Xeon. When I was preparing the material to pitch CET to management, I pointed people to Security Now! Episode 211 to explain what ROP was." So I got a big kick out of the fact that this podcast was used to explain to management why return-oriented programming was a problem, and that they were using this CET, this control-flow enforcement technology, to deal with it.

Okay. So, as I said, there are two components to CET. We'll tackle them separately. The first is ENDBRANCH, and the second one is the shadow stack. So the problem with return-oriented programming is that it's a clever way that hackers have figured out to get code executed that already has privilege to execute. What I mean is that one of the ways that - so there have been previous efforts, sort of an ongoing effort over time, to better secure and lock down our systems using hardware, that is, using architectural improvements.

One of them is the so-called NX bit, the No-eXecute bit. And that was added to systems after noticing that a frequently occurring problem with security, that is, that hackers were able to leverage, is that they were able to provide data to the target system, the exploit target, and get that data to execute. That is, there was no differentiation, there was no separation between the data and the instructions. And in fact, in classic architecture, standard von Neumann computer architecture, you do have a mixed data and instruction space. You have a single memory space, as opposed to a Harvard

architecture, where instructions are separated from data. They're completely separate. The architecture that has generally succeeded has data instructions existing in the same memory environment.

Well, that creates an inherent problem because it means that, if you can somehow get the chip, get the processor to execute a jump instruction into the data, it will execute data as if it's instructions. So the so-called NX, the No-eXecute bit, it's a flag which was added to the hardware, so it's enforced by hardware, which says this region of memory cannot be executed. So any time the instruction pointer in the processor is jumped into a region of memory with the No-eXecute bit set, it just causes an abort. Basically, it safely terminates the program, and without executing even that one single instruction that it was aimed at. So that was the first countermeasure against bad guys being able to use a buffer overrun to provide the data that would be then executed.

But they're very clever, these hackers. And so they said, okay, now the data is marked as non-executable. So what are we going to do? And they said, well, where can we find some executable instructions? And it's like, well, the computer's full of them. Look at that operating system. It's got all kinds of instructions, all over the place. So what they cleverly figured out was it was possible to execute instructions already there, that is, little snippets of code, typically at the end of a subroutine, because they generally don't want to do all of what an existing subroutine does. That's not going to be what they want. Typically, they'll find some instructions that already exist in Ring 0 in the kernel that are privileged, which when executed do something they need. And typically it's lift any other restrictions on them. That is, they can't do too much. But if they can simply find a little snippet of code that, for example, turns off the NX bit, then suddenly they can execute whatever they want to in the data space.

So what the hackers do is they arrange to jump to near the end of an existing subroutine of code, which naturally is executable because it's in the kernel. It is executable by definition. And those few instructions will be executed, and then the return instruction at the end of the subroutine is reached, which returns control to them - and normally, with whatever they wanted to get done, done. Maybe the NX bit has been flipped off, or the range of memory they can access has been extended to the entire range, so they're no longer restricted by the memory map of the system. Whatever. So Intel was saying, okay. How can we fix this problem? And they came up with just the cleverest solution.

PADRE: Would this be ASLR?

Steve: Well, okay. So good point. That was the first mitigation, was address space layout randomization. So here we had this problem of people jumping into known locations in the kernel and just, like, doing that with abandon. So the first mitigation, well, okay. The first was DEP, data execution prevention that we talked about, with the NX bit.

The second one was, after that sort of didn't solve the whole problem, they added ASLR, address space layout randomization, where the operating system at boot time, as it's loading, would deliberately randomize where the various blocks of its own code were loaded in memory. There's a whole process called "fixup," where the addresses of all the different modules are sort of established. But the loading order was normally fixed, that it was just sort of default. It didn't have to be that way. It was just that way because no one bothered to scramble it up.

So they said, well, let's scramble it up on purpose. So that's address space layout randomization. It turns out, though, that for architectural reasons, the granularity of their ability to place things in random places isn't very good. And it's possible for malicious code to probe for the location of code in the kernel, or just to not work as well.

For example, oftentimes the granularity is just eight bits, so 256 possible locations for the various modules. So if it just guesses one of the locations, it's going to be wrong 255 out of 256 times. But it's going to be right one out of 256 times. And so that's a low probability of success exploit, but it's better than zero. And if you've got enough machines in a huge environment on the Internet, and if you just - if somehow you can try again, even on the same machine, you're going to get lucky sooner or later.

PADRE: Steve, if it guesses wrong, if it's one of those in the 256 that it didn't get, does it just crash that thread? Does it crash the machine? What would a user see if it was an unsuccessful attempt to access code where it thought it was in memory?

Steve: It would be bad.

PADRE: Okay. It would be a bad thing. I got it. Okay.

Steve: Yeah. It would be like I don't know why my computer just froze.

PADRE: All right.

Steve: I don't know why my mouse stopped working. I got a Blue Screen of Death. So you go, oh, shoot. And so you reboot, and it doesn't happen again. It's like, huh. That was weird. I wonder what happened? And so you don't know that something just tried to own you, but guessed wrong.

PADRE: Right, right.

Steve: All you know is that your system went wonky, but now it seems fine. So okay.

PADRE: Let me back my propeller hat off a little bit because I love the execution of this. I mean, of course it can be used for nefarious reasons. But when we typically talk about exploits, we talk a lot about buffer overflows because it allows for arbitrary code execution, which I've always enjoyed. But this idea of being able to do what amounts to arbitrary code execution, using code that's already loaded into memory, that's far more elegant. I mean, this seems like something that would be incredibly difficult to figure out because it's not your code that you're running. You're running snippets of other people, other companies' codes, to get it to do what you want it to do.

Steve: Right. And I've looked at what they've actually done. And in some cases it's as clever as - okay. So instructions are typically multi-byte things. So the opcode, some opcodes are just, like operation codes, are just eight bits. But Intel is a variable instruction-length architecture, where instructions that are more complex or used less often will have a prefix that says the instruction is a member of this group, and then the succeeding bytes provide more definition. And then you often have arguments to that instruction.

So these guys, the hackers, are so good that they're not even jumping into the beginning of an instruction. In some cases they're jumping into the middle of a single instruction's multiple bytes and realizing that they want this opcode, which happens to be the third byte of a multi-byte instruction, but it's going to do what they want. I mean, it's just, when you look at it, you think, wow. I mean, we're talking serious deep voodoo in order to make this work.

PADRE: See, I don't even get that because - especially with ASLR. So since it's pseudorandomized, where those little bits and snippets will be placed inside of memory,

it only has to guess wrong once in that entire string of bytes that it's puling out of different available spaces of memory. It seems as if the odds are not just stacked against this working, but they're incredibly stacked against it working. And yet, as you're describing it, it seems to work. How do they do that? Can they just naturally assume that some spaces in memory are going to be safely where they think they'll be?

Steve: Often it's possible - well, for one, yes. But often it's possible to probe where these large block modules are located. So, for example, when you return from a subroutine, the stack - and we'll be talking about the stack when we talk about the second part of this, the so-called "shadow stack." The execution stack is where the return address is stored when you go to a subroutine. And so it uses the return address on the stack to get back to you. But it's often the case that you're able to look at the code that called the subroutine, and that completely decloaks it. If you're able to see a call to the subroutine, now you know exactly where at least that large module is located. And then you use an offset from that in order to execute the code you want.

PADRE: So once you get the proper response, you know how long that code snippet is, and you know exactly where you need to jump in to get the opcode you want.

Steve: Exactly.

PADRE: So how deep are we getting? Are we talking about like assembly type codes at that point?

Steve: Oh, yeah, yeah.

PADRE: Are they pulling, like, pushes and pops?

Steve: Yes.

PADRE: Really.

Steve: We are down at, for example, a return - okay. There's a no-op and a return. They're either 60 or 90, and I don't remember which is which. But, I mean, this is the way I code. So I don't actually always look at the machine language, but I'm used to seeing 90s and 60s, either as no-ops or returns. And so it's literally that pattern of bits is the computer interprets it that way. So here's the brilliance of what Intel's - the first part of this. They created and they defined a new instruction called the ENDBRANCH instruction. And it is simply this. That instruction must be the target of a call or a jump instruction. That is, in this new architecture, when this is enabled, that instruction, ENDBRANCH, is the only valid destination for a call or a jump.

PADRE: Okay.

Steve: So what that would mean is the beginning of every subroutine would start with an ENDBRANCH.

PADRE: Got it.

Steve: And what's so clever about the implementation, I just love this, is we talked about the architecture of processors years ago, did a whole series, followed the evolution of architectures all the way up. And at some point in any processor, you have an instruction pipeline. That is, you have a series of instructions, one after the other, that the processor is executing. And the brilliance of this ENDBRANCH is that the way Intel

implemented it, as the processor is reading through instructions, there's a little - it has a tiny little state machine. When it encounters a call or a jump, which are the two instructions - a call says I'm going to call this function and expect it to return to me. A jump is I'm going to jump somewhere, and then whatever happens, happens. But either of those have to have and have to land on an ENDBRANCH.

But think about what that means in terms of the instruction pipeline. It means, in the pipeline, there is going to be a call or a jump instruction. And the immediately following instruction has to be ENDBRANCH. That is, the thing fetching instructions on behalf of the processor, after it fetches a call or a jump and executes it, the next thing it fetches is an ENDBRANCH because that's where the call or jump have to jump to.

PADRE: It hands off control to that ENDBRANCH.

Steve: Exactly. And so the Intel design, they added a tiny little state machine that just sort of supervises the execution stream. And if it ever sees a jump or a call, it looks to verify that the immediately succeeding instruction is ENDBRANCH. If not, it raises an exception and aborts the process.

PADRE: Okay.

Steve: So this completely solves the ROP, the return-oriented programming problem. You can no longer jump. Nobody, not even good code, but good code doesn't. Good code has no reason to do a far jump or call into the middle of some subroutine somewhere. It always wants to come in at the top. So essentially what this does is, by putting ENDBRANCH instructions as the first instruction of all of your subroutines, it defines the single legal entry point for subroutines. And any attempt to jump into the middle or near the end, which is the way the ROP typically occurs, immediately aborts the process, and nothing bad can happen to your system.

PADRE: I love that. That's absolutely elegant.

Steve: So elegant. Oh, and Intel also chose the specific ENDBRANCH instruction so that it is a no-op, a no-operation, on all current and past generations of chips. It's an unused, do-nothing byte sequence. Which means that, when compilers begin compiling code with ENDBRANCH awareness turned on, so that they're sticking these little ENDBRANCH instructions at the beginning of all the subroutines in the system, that same code can run perfectly on older systems that don't have ENDBRANCH. The processors are just going to say, gee, that's weird. I wonder why all of the subroutines start with a no-operation? Oh, well. Who cares? There's nothing here to do, so we'll just go on to the next one. So it's beautifully backward-compatible to previous architectures.

PADRE: The only way you could exploit this is if somehow you exploited that stateless machine that's looking for the ENDBRANCH.

Steve: That's in the hardware.

PADRE: That's in the hardware, so good luck with that.

Steve: It's built, it's like, right down in the instruction set. The only thing I could think is that, if there was a full function that you could use, that is, not just a few instructions at the end of a subroutine, but if somehow the whole function was usable, but that's generally not doing what you want. Normally it's a clever re-use of existing code, very much near just before a return instruction, is what the bad guys have been using. And

this ends that.

PADRE: Besides, calling on a full function, if it will be allowed by the ENDBRANCH stateful processor, that's going to hand off control to whatever that function had originally been connected to. It's not going to come back to you as the attacker. I couldn't think of a complete function that you would be able to use to bypass authentication.

Steve: Yeah. It would be unexpected. But Part 2: The Shadow Stack fixes that.

PADRE: No. Why can't we just end with the happy news, Steve? Why do we have to get to the depressing stuff? All right. You've heard about the brilliant way to stop ROP, the brilliant use of ENDBRANCH, and now Steve Gibson is going to let it crash down all around us.

Steve: Okay. So the second part of this is another new feature that Intel put into this CET technology. To understand this we need to go back a little bit and look at the concept of a software-accessible stack. I would argue that the concept of a stack is probably one of the greatest innovations in computer science. Old machines didn't have a stack. These three PDP-8s have no stack. There was no concept of a stack in them. There's an instruction to allow a jump to a subroutine.

But the way it works is - so the problem is, if you jump to somewhere because you want a subroutine to be executed, how do you get back? If you just do a jump instruction, then you're somewhere else. But how does that place where you are know where you were? So on these older machines, on these 12-bit PDP-8s, there is a jump to subroutine. The way it works is the best they could do at the time, and that is the instruction that you jump to is where the address is stored of where you were. And then execution begins with the instruction afterwards.

So on those old machines, the first instruction of a subroutine is blank because that's going to actually be where the computer stores the address that you came from so that, at the end of the subroutine, you're able to go back. Now, that's cool, but it's got a problem. And that is that it doesn't allow recursion. That is, for example, that subroutine itself, nor any other subroutine it might call, could ever call it because, if they did, they would overwrite that return address that was stored at the top. In other words, that system it works, but you have to be very careful that no way for the code to execute would ever allow the subroutine to be called again before it had returned to its caller.

So that older system does not support any kind of recursion. We solved that problem not long after that, actually, by implementing a stack architecture in the machine. And, I mean, it's such a win that you don't - there just are none any longer, unless they're operating on my bookshelf, that don't have stacks.

So what is a stack? Everyone's sort of familiar with this notion of pushing and popping. You push something on the stack; you pop something from the stack. To really understand its value we have to go a little bit beyond that. But just looking at that, and using the example I was just giving with a non-stack machine, imagine that there's this thing, which we'll sort of leave undefined for a moment, called a "stack," where you can push and pop. That is, and you don't have to worry about any of the details.

So in a stack system, when you're jumping to a subroutine, you push on the stack typically the address of the next instruction, and then you jump to wherever you want to go. And then that subroutine is executed. And then the way that subroutine gets back to who called it is the stack is popped, which reveals the address of the instruction

underneath the one that called the subroutine. And so you do what's known as an indirect jump, that is, you jump to the address on the stack, which brings you home, brings you back.

Now, one of the beauties of that is now you can do recursion because, for example, that subroutine that you jump to, it could call itself, if it wanted to, because it would, if it called itself, it would push the instruction next in line for where it's calling itself on the stack and then call itself. And then if that subroutine then returned, it would pop the stack, which would reveal the instruction in that subroutine, returning to where it came from. And then if that returned, it would pop the stack again and come back to the original caller. In other words, it's this incredibly convenient scratchpad, but it is strictly sequence-based. That is, the order in which you push things, they are popped off in the reverse order.

And so that's one of the brilliant insights to this concept of a stack. And with that, for example, you get this amazing convenience of having subroutines able to call, you know, pretty much do anything they want to. They can call themselves; they can call other people that call them. And finally it all sort of unwinds itself back to the original caller by removing things in the reverse order that they were put on the stack.

Well, there's something more you can do with it that is even more clever. And that is you can use it to pass parameters to functions. So say that you wanted to put a circle on the screen at a certain coordinate where the upper left corner and the lower right corner were at two coordinates. And so you had a subroutine that could draw circles. But it didn't know where you wanted to draw them. So what you do is you push on the stack the coordinates for the upper left and the lower right. And then you call the circle function. The circle function knows to expect its parameters to be on the stack.

So in all these systems there's something known as the "stack pointer." It's a register that points to the top of the stack. Well, since several of these parameters were pushed down on the stack, then the return address was pushed on the stack, this subroutine is able to look at an offset from where the stack pointer currently is. That is, it's able to sort of look down into the stack, into the history, and find its parameters there. So it goes and looks deeper in the stack, finds its parameters, draws the circle. Then it returns, and the function which called it, which pushed those things on the stack, typically removes them from the stack.

So the calling function pushed some things on the stack; called the circle draw, which knew where to look for them. Then it returned, and then the caller says, oh, I put some stuff on the stack which the other guy used. Now I'm going to free them. So essentially the same number of things it pushed on the stack, it pops, in order to discard them, essentially. So this has been very, very clever.

Now, there's one more thing that the stack is typically used for. And that's local variables. We just talked about passing parameters on the stack. The final thing is, say that this circle draw subroutine needs to use some memory to do its work. It needs some scratchpad memory. It can use the stack, too. It essentially can do the equivalent of pushing some blanks, some blank space on the stack. And it knows where those blank spaces are. So it's freely able to use them as scratch memory. If it calls some other function, that's fine. It might push those parameters on the stack, then call the function, and so forth. The function comes back. It pops those parameters, and everything is sort of undone. So again, we have this incredibly convenient scratchpad facility where we're able to put parameters on the stack. The stack is able to store where we came from so we can get back there, and we're able to use that stack as local scratchpad.

Now, it turns out this is, clever as it is, convenient as it is, this is the cause for more of the pain. This is why Steve Ballmer had a meltdown. This is the buffer overrun problem. Why? Because, think about it, we are mixing in one place both data and execution pointers. Those return addresses that are on the stack that get us back to where we came from, the processor just assumes they're valid. It doesn't know. When you hit a return instruction, it pops it off the stack and goes where that says. But that return instruction was living right next to data, scratchpad data, that the program may have been using for its own purposes. What if the program allocates a buffer on the stack and then fills it with data that it receives from the Internet? And what if that...

PADRE: Oh, no.

Steve: Yes, yes. And it's so convenient. The stack is there. It grows and shrinks as you need it. Everybody uses it because it's just - it's self-serve. It's like, oh, I need a buffer. Allocate a block of memory on the stack. It's always going to be in memory because it's not going to get swapped out. It's local. It's fast to access. It's brilliant. But you have to use it perfectly. Otherwise you get in trouble. And so hackers have exploited mistakes in code to write their own data and even overwrite like four bytes past where you should. That's the return instruction from that subroutine, which the processor is going to believe. And so if a bad guy can supply data and manage a buffer overrun, that wipes out the proper return instruction. So when the processor tries to return from that subroutine, it goes somewhere else.

PADRE: And this would get them past ENDBRANCH; right? Because, I mean...

Steve: Precisely.

PADRE: It's just checking for the ENDBRANCH. It's not actually verifying that it's returning to what the ENDBRANCH said it should return to.

Steve: Right.

PADRE: Great.

Steve: Right. So it provides a way for malicious parties to disrupt the proper execution of the code. So, and where does the problem come from? The problem comes from the fact that we are mixing data and execution. We're mixing like program data and processor instruction data, mixing it all up in the same structure on the stack. So what Intel did is another piece of brilliance. They said, we're going to create a shadow stack.

The problem with the main stack is that, I mean, the beauty is it's so handy. You push parameters on it. You call a function. The function can allocate its own local variables, and they'll be discarded when the function exits because the stack will be popped. And then the return instruction returns to where it came from, and the caller pops its parameters that it had pushed for the callee, pops them, I mean, it's brilliant. But the problem is it's under program control, and it mixes data and execution.

So Intel with CET is creating a shadow stack that the programmer, the normal system, has no control over. And the only data that are pushed and popped are the ones that have implied use of the stack. That is, a call instruction has an implied use of the stack because the return from the call is automatically pushed on the stack. Similarly, the return instruction from a subroutine has implied use of the stack because the return instruction always gets the address it's going to return to from the stack.

So what the shadow stack does is it obeys the same implied activities for calls and returns. But the programmer has no access to it. That is, when you're pushing things on the stack, you're only pushing them on the visible stack. When you pop them, you're only popping them from the visible stack. But when you do a call, the visible stack and the shadow stack both store the return address. And here's the key. When you do a return, the system verifies that the shadow stack's return address matches the visible stack's return address.

PADRE: Ah, there we go.

Steve: If they don't match, something is wrong, and the process is terminated. So that completely prevents malicious use or even mistaken use. This will catch bugs faster than anyone's business, immediately catch the bug. But it will also immediately shut down anyone trying to use stack manipulation, buffer overruns, in order to get their own code to execute, to disrupt the function of a return instruction, to have that return instruction go somewhere else because it won't match what the shadow stack has because the shadow stack they have no control over, and it will always have the original true and correct return address. If the system tries to return to an address that the shadow stack doesn't agree with, it immediately produces a system exception and stops running. So it's just beautiful.

PADRE: I see how this works, Steve. But am I going to get a performance hit from checking two stacks?

Steve: No, it's all in the hardware.

PADRE: Oh, nice.

Steve: Yup. No effect whatsoever. The burden's on Intel. The chips are going to get bigger. The wattage is going to go up. They're going to burn more power. There's going to be another bazillion little transistors. But it just all happens by magic. So I just think it's very cool.

PADRE: [Vetman] in the chatroom asks if this requires OS support. It shouldn't, though. It's hardware control; correct? I mean, the OS just would be told it's an invalid instruction.

Steve: No, I think it would require at least a driver. Something in the kernel would need to turn this on. And Intel will be defining some new exception errors. And so the operating system would gain control and then decide what to do with it. So there would be new features in the chip. But it would probably be a matter of just adding a kernel driver to it instead of like a whole new operating system.

PADRE: Okay, so then that's what they would go after. They would go after the driver and basically just tell the driver to keep saying it's okay, it's okay, it's okay.

Steve: In order to disable the shadow stack.

PADRE: All right. So that will be on a future episode of Security Now!. Once this gets implemented perfectly, someone will have infiltrated the driver, and the shadow stack will no longer work. I love the game of cat and mouse, Steve, and no one plays it better than you. We've heard about SpinRite. We know that it's the tool that needs to be in all of our toolkits. Can you tell them where they can find you? Where they can find your work?

Steve: Well, after 1.26 million people, I think it is, downloaded Never10, everybody pretty much knows: GRC.com.

PADRE: I did not download Never10. And I actually had three different machines that were hit by just weird Windows 10 updates.

Steve: Yeah, 1.266411 million downloads.

PADRE: Wow. And how long did that take?

Steve: To do what?

PADRE: To get that 1.2?

Steve: Actually, I think this began in April. And it sort of - it cruised along for a while. But it was only like in the previous month, when Microsoft really put the screws on and really began to get more desperate, we saw a huge surge in downloads. But it is the fastest, most popular piece of freeware in the history of GRC. We have things that over two decades, things like - what was that little firewall tester? Leaktest. I don't know, like seven million downloads or something. But it's been decades. This thing is a few months old, and it's at 1.266 million downloads. There's never been a more popular piece of freeware that I've written.

PADRE: It just goes to show you, we've got empirical evidence on how many people are annoyed by Microsoft's super-aggressive Win10 upgrade.

Steve: Exactly.

PADRE: Steve Gibson, of course, the brain behind the GRC.com. He is my personal guru for all things security. And more than that, Steve, it is always a pleasure to be able to chat with you, just to sit back and let the propeller-head go. Sir, please, please, next time Leo goes away, please put in a good word for me.

Steve: It's been my pleasure, Padre, as always.

PADRE: Of course, that does it for this episode of Security Now! Don't forget that we're live here on TWiT.tv, live.twit.tv, every Tuesday at 13:30 Pacific time. Steve will always be here to inject you with some healthy paranoia and keep you safe in the wonderful world of insecurity.

Now, you can find all of our shows on the TWiT show page at TWiT.tv/sn, that's for Security Now!, as well as on iTunes, Stitcher, and wherever fine podcasts are aggregated. You can also find high-quality audio downloads on GRC.com, which is also where you'll find all the goodness that is SpinRite, ShieldsUP!, and coming soon SQRL. Oh, and don't forget Never10.

I'm Father Robert Ballecer, the Digital Jesuit, in for Leo Laporte, saying that, if you want to keep your data into the future, you've got to remember Security Now!.